Handbook of Philosophical Logic

2nd Edition

Volume 9

edited by Dov M. Gabbay and F. Guenthner

CONTENTS

PREFACE TO THE SECOND EDITION

It is with great pleasure that we are presenting to the community the second edition of this extraordinary handbook. It has been over 15 years since the publication of the first edition and there have been great changes in the landscape of philosophical logic since then.

The first edition has proved invaluable to generations of students and researchers in formal philosophy and language, as well as to consumers of logic in many applied areas. The main logic article in the Encyclopaedia Britannica 1999 has described the first edition as 'the best starting point for exploring any of the topics in logic'. We are confident that the second edition will prove to be just as good!

The first edition was the second handbook published for the logic community. It followed the North Holland one volume *Handbook of Mathematical Logic*, published in 1977, edited by the late Jon Barwise. The four volume *Handbook of Philosophical Logic*, published 1983–1989 came at a fortunate temporal junction at the evolution of logic. This was the time when logic was gaining ground in computer science and artificial intelligence circles.

These areas were under increasing commercial pressure to provide devices which help and/or replace the human in his daily activity. This pressure required the use of logic in the modelling of human activity and organisation on the one hand and to provide the theoretical basis for the computer program constructs on the other. The result was that the *Handbook of Philosophical Logic*, which covered most of the areas needed from logic for these active communities, became their bible.

The increased demand for philosophical logic from computer science and artificial intelligence and computational linguistics accelerated the development of the subject directly and indirectly. It directly pushed research forward, stimulated by the needs of applications. New logic areas became established and old areas were enriched and expanded. At the same time, it socially provided employment for generations of logicians residing in computer science, linguistics and electrical engineering departments which of course helped keep the logic community thriving. In addition to that, it so happens (perhaps not by accident) that many of the Handbook contributors became active in these application areas and took their place as time passed on, among the most famous leading figures of applied philosophical logic of our times. Today we have a handbook with a most extraordinary collection of famous people as authors!

The table below will give our readers an idea of the landscape of logic and its relation to computer science and formal language and artificial intelligence. It shows that the first edition is very close to the mark of what was needed. Two topics were not included in the first edition, even though

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, vii–ix.

they were extensively discussed by all authors in a 3-day Handbook meeting. These are:

- a chapter on non-monotonic logic
- a chapter on combinatory logic and λ-calculus

We felt at the time (1979) that non-monotonic logic was not ready for a chapter yet and that combinatory logic and λ-calculus was too far removed.[1] Non-monotonic logic is now a very major area of philosophical logic, alongside default logics, labelled deductive systems, fibring logics, multi-dimensional, multimodal and substructural logics. Intensive re-examinations of fragments of classical logic have produced fresh insights, including at time decision procedures and equivalence with non-classical systems.

Perhaps the most impressive achievement of philosophical logic as arising in the past decade has been the effective negotiation of research partnerships with fallacy theory, informal logic and argumentation theory, attested to by the Amsterdam Conference in Logic and Argumentation in 1995, and the two Bonn Conferences in Practical Reasoning in 1996 and 1997.

These subjects are becoming more and more useful in agent theory and intelligent and reactive databases.

Finally, fifteen years after the start of the Handbook project, I would like to take this opportunity to put forward my current views about logic in computer science, computational linguistics and artificial intelligence. In the early 1980s the perception of the role of logic in computer science was that of a specification and reasoning tool and that of a basis for possibly neat computer languages. The computer scientist was manipulating data structures and the use of logic was one of his options.

My own view at the time was that there was an opportunity for logic to play a key role in computer science and to exchange benefits with this rich and important application area and thus enhance its own evolution. The relationship between logic and computer science was perceived as very much like the relationship of applied mathematics to physics and engineering. Applied mathematics evolves through its use as an essential tool, and so we hoped for logic. Today my view has changed. As computer science and artificial intelligence deal more and more with distributed and interactive systems, processes, concurrency, agents, causes, transitions, communication and control (to name a few), the researcher in this area is having more and more in common with the traditional philosopher who has been analysing

[1] I am really sorry, in hindsight, about the omission of the non-monotonic logic chapter. I wonder how the subject would have developed, if the AI research community had had a theoretical model, in the form of a chapter, to look at. Perhaps the area would have developed in a more streamlined way!

such questions for centuries (unrestricted by the capabilities of any hardware).

The principles governing the interaction of several processes, for example, are abstract an similar to principles governing the cooperation of two large organisation. A detailed rule based effective but rigid bureaucracy is very much similar to a complex computer program handling and manipulating data. My guess is that the principles underlying one are very much the same as those underlying the other.

I believe the day is not far away in the future when the computer scientist will wake up one morning with the realisation that he is actually a kind of formal philosopher!

The projected number of volumes for this Handbook is about 18. The subject has evolved and its areas have become interrelated to such an extent that it no longer makes sense to dedicate volumes to topics. However, the volumes do follow some natural groupings of chapters.

I would like to thank our authors are readers for their contributions and their commitment in making this Handbook a success. Thanks also to our publication administrator Mrs J. Spurr for her usual dedication and excellence and to Kluwer Academic Publishers for their continuing support for the Handbook.

Dov Gabbay
King's College London

Logic	**IT**			
	Natural language processing	**Program control specification, verification, concurrency**	**Artificial intelligence**	**Logic programming**
Temporal logic	Expressive power of tense operators. Temporal indices. Separation of past from future	Expressive power for recurrent events. Specification of temporal control. Decision problems. Model checking.	Planning. Time dependent data. Event calculus. Persistence through time—the Frame Problem. Temporal query language. temporal transactions.	Extension of Horn clause with time capability. Event calculus. Temporal logic programming.
Modal logic. Multi-modal logics	generalised quantifiers	Action logic	Belief revision. Inferential databases	Negation by failure and modality
Algorithmic proof	Discourse representation. Direct computation on linguistic input	New logics. Generic theorem provers	General theory of reasoning. Non-monotonic systems	Procedural approach to logic
Non-monotonic reasoning	Resolving ambiguities. Machine translation. Document classification. Relevance theory	Loop checking. Non-monotonic decisions about loops. Faults in systems.	Intrinsic logical discipline for AI. Evolving and communicating databases	Negation by failure. Deductive databases
Probabilistic and fuzzy logic	logical analysis of language	Real time systems	Expert systems. Machine learning	Semantics for logic programs
Intuitionistic logic	Quantifiers in logic	Constructive reasoning and proof theory about specification design	Intuitionistic logic is a better logical basis than classical logic	Horn clause logic is really intuitionistic. Extension of logic programming languages
Set theory, higher-order logic, λ-calculus, types	Montague semantics. Situation semantics	Non-well-founded sets	Hereditary finite predicates	λ-calculus extension to logic programs

Imperative vs. declarative languages	**Database theory**	**Complexity theory**	**Agent theory**	**Special comments: A look to the future**
Temporal logic as a declarative programming language. The changing past in databases. The imperative future	Temporal databases and temporal transactions	Complexity questions of decision procedures of the logics involved	An essential component	Temporal systems are becoming more and more sophisticated and extensively applied
Dynamic logic	Database updates and action logic	Ditto	Possible actions	Multimodal logics are on the rise. Quantification and context becoming very active
Types. Term rewrite systems. Abstract interpretation	Abduction, relevance	Ditto	Agent's implementation rely on proof theory.	
	Inferential databases. Non-monotonic coding of databases	Ditto	Agent's reasoning is non-monotonic	A major area now. Important for formalising practical reasoning
	Fuzzy and probabilistic data	Ditto	Connection with decision theory	Major area now
Semantics for programming languages. Martin-Löf theories	Database transactions. Inductive learning	Ditto	Agents constructive reasoning	Still a major central alternative to classical logic
Semantics for programming languages. Abstract interpretation. Domain recursion theory.		Ditto		More central than ever!

Classical logic. Classical fragments	Basic background language	Program synthesis	A basic tool	
Labelled deductive systems	Extremely useful in modelling		A unifying framework. Context theory.	Annotated logic programs
Resource and substructural logics	Lambek calculus		Truth maintenance systems	
Fibring and combining logics	Dynamic syntax	Modules. Combining languages	Logics of space and time	Combining features
Fallacy theory				
Logical Dynamics	Widely applied here			
Argumentation theory games		Game semantics gaining ground		
Object level/ metalevel			Extensively used in AI	
Mechanisms: Abduction, default relevance			ditto	
Connection with neural nets				
Time-action-revision models			ditto	

	Relational databases	Logical complexity classes	The workhorse of logic	The study of fragments is very active and promising.
	Labelling allows for context and control.		Essential tool.	The new unifying framework for logics
Linear logic			Agents have limited resources	
	Linked databases. Reactive databases		Agents are built up of various fibred mechanisms	The notion of self-fibring allows for self-reference
				Fallacies are really valid modes of reasoning in the right context.
			Potentially applicable	A dynamic view of logic
				On the rise in all areas of applied logic. Promises a great future
			Important feature of agents	Always central in all areas
			Very important for agents	Becoming part of the notion of a logic
				Of great importance to the future. Just starting
			A new theory of logical agent	A new kind of model

NARCISO MARTÍ-OLIET AND JOSÉ MESEGUER

REWRITING LOGIC AS A LOGICAL AND SEMANTIC FRAMEWORK

1 INTRODUCTION

The relationships between logic and computation, and the mutual interactions between both fields, are becoming stronger and more pervasive than they have ever been. In fact, our way of thinking about both logic and computation is being altered quite strongly. For example, there is such an increasingly strong connection—in some cases to the point of complete identification—between computation and deduction, and such impressive progress in compilation techniques and computing power, that the frontiers between logical systems, theorem provers, and declarative programming languages are shifting and becoming more and more tenuous, with each area influencing and being influenced by the others.

Similarly, in the specification of languages and systems there is an increasing shift from mathematically precise but somewhat restricted formalisms towards specifications that are not only mathematical, but actually logical in nature, as exemplified, for example, by specification formalisms such as algebraic specifications and structural operational semantics. In this way, languages and systems that in principle may not seem to bear any resemblance to logical systems and may be completely "conventional" in nature, end up being conceptualized primarily as *formal* systems.

However, any important development brings with it new challenges and questions. Two such questions, that we wish to address in this paper are:

- *How can the proliferation of logics be handled?*
- *Can flexible logics allowing the specification and prototyping of a wide variety of languages and systems with naturalness and ease be found?*

Much fruitful research has already been done with the aim of providing adequate answers to these questions. Our aim here is to contribute in some measure to their ongoing discussion by suggesting that rewriting logic [Meseguer, 1992] seems to have particularly good properties recommending its use as both a *logical framework* in which many other logics can be represented, and as a general *semantic framework* in which many languages and systems can be naturally specified and prototyped.

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, 1–87.

1.1 Rewriting logic as a logical framework

In our view, the main need in handling the proliferation of logics is primarily conceptual. What is most needed is a *metatheory* of logics helping us to better understand and explore the boundaries of the "space" of all logics, present and future, and to relate in precise and general ways many of the logics that we know or wish to develop.

Following ideas that go back to the original work of Goguen and Burstall [1984] on *institutions*, we find very useful understanding the space of all logics as a *category*, with appropriate translations between logics as the arrows or morphisms between them. The work on institutions has been further developed by their original proponents and by others [Goguen and Burstall, 1986; Goguen and Burstall, 1992; Tarlecki, 1984; Tarlecki, 1985], and has influenced other notions proposed by different authors [Mayoh, 1985; Poigné, 1989; Fiadeiro and Sernadas, 1988; Meseguer, 1989; Harper *et al.*, 1989a; Salibra and Scollo, 1993; Ehrig *et al.*, 1991; Astesiano and Cerioli, 1993]. Some of the notions proposed are closely related to institutions; however, in other cases the main intent is to substantially expand the primarily model-theoretic viewpoint provided by institutions to give an adequate treatment of proof-theoretic aspects such as entailment and proof structures. The theory of general logics [Meseguer, 1989] that we present in summary form in Section 2 is one such attempt to encompass also proof-theoretic aspects, and suggests not just one space or category of logics, but several, depending on the proof-theoretic or model-theoretic aspects that we wish to focus on.

In our view, the quest for a *logical framework*, understood as a logic in which many other logics can be represented, is important but is not the primary issue. Viewed from the perspective of a general space of logics, such a quest can in principle—although perhaps not in all approaches—be understood as the search within such a space for a logic $\mathcal{F}$ such that many other logics $\mathcal{L}$ can be represented in $\mathcal{F}$ by means of mappings $\mathcal{L} \longrightarrow \mathcal{F}$ that have particularly nice properties such as being conservative translations.

Considered in this way, and assuming a very general axiomatic notion of logic and ambitious enough requirements for a framework, there is in principle no guarantee that such an $\mathcal{F}$ will necessarily be found. However, somewhat more restricted successes such as finding an $\mathcal{F}$ in which all the logics of "practical interest," having finitary presentations of their syntax and their rules, can be represented can be very valuable and can provide a great economy of effort. This is because, if an implementation for such a framework logic exists, it becomes possible to implement through it all the other "object logics" that can be adequately represented in the framework logic.

Much work has already been done in this area, including the Edinburgh logical framework LF [Harper *et al.*, 1993; Harper *et al.*, 1989; Gardner, 1992] and meta-theorem provers such as Isabelle [Paulson, 1989], λProlog

[Nadathur and Miller, 1988; Felty and Miller, 1990], and Elf [Pfenning, 1989], all of which adopt as framework logics different variants of higher-order logics or type theories. There has also been important work on what Basin and Constable [1993] call *metalogical* frameworks. These are frameworks supporting reasoning about the metalogical aspects of the logics being represented. Typically, this is accomplished by reifying as "data" the proof theory of the logic being represented in a process that is described in [Basin and Constable, 1993] as *externalizing* the logic in question. This is in contrast to the more *internalized* form in which logics are represented in LF and in meta-theorem provers, so that deduction in the object logic is mirrored by deduction—for example, type inference—in the framework logic. Work on metalogical frameworks includes the already mentioned paper by Basin and Constable [1993], who advocate constructive type theory as the framework logic, work of Matthews, Smaill, and Basin [1993], who use Feferman's FS_0 [Feferman, 1989], a logic designed with the explicit purpose of being a metalogical framework, earlier work by Smullyan [1961], and work by Goguen, Stevens, Hobley, and Hilberdink [1992] on the 2OBJ meta-theorem prover, which uses order-sorted equational logic [Goguen and Meseguer, 1992; Goguen *et al.*, 2000].

A difficulty with systems based on higher-order type theory such as LF is that it may be quite awkward and of little practical use to represent logics whose structural properties differ considerably from those of the type theory. For example, linear and relevance logics do not have adequate representations in LF, in a precise technical sense of "adequate" [Gardner, 1992, Corollary 5.1.8]. Since in metalogical frameworks a direct connection between deduction in the object and framework logics does not have to be maintained, they seem in principle much more flexible in their representational capabilities. However, this comes at a price, since the possibility of directly using an implementation of the framework logic to implement an object logic is compromised.

In relation to this previous work, rewriting logic seems to have great flexibility to represent in a natural way many other logics, widely different in nature, including equational, Horn, and linear logics, and any sequent calculus presentation of a logic under extremely general assumptions about such a logic. Moreover, quantifiers can also be treated without problems. More experience in representing other logics is certainly needed, but we are encouraged by the naturalness and directness—often preserving the original syntax and rules—with which the logics that we have studied can be represented. This is due to the great simplicity and generality of rewriting logic, since in it all syntax and structural axioms are user-definable, so that the abstract syntax of an object logic can be represented as an algebraic data type, and is also due to the existence of only a few general "meta" rules of deduction relative to the rewrite rules given by a specification, where such a specification can be used to describe with rewrite rules the rules of deduc-

tion of the object logic in question. In addition, the direct correspondence between proofs in object logics and proofs in the framework logic can often be maintained in a *conservative* way by means of maps of logics, so that an implementation of rewriting logic can directly support an implementation of an object logic. Furthermore, given the directness with which logics can be represented, the task of proving conservativity is in many cases straightforward. Finally, although we do not discuss this aspect which is left for a subsequent paper, externalization of logics to support metalogical reasoning is also possible in rewriting logic.

Another important difference is that most approaches to logical frameworks are proof-theoretic in nature, and thus they do not address the model theories of the logics being represented. By contrast, several of the representations into rewriting logic that we consider—such as those for equational logic, Horn logic, and linear logic—involve both models and proofs and are therefore considerably more informative than purely proof-theoretic representations.

The fact that rewriting logic is *reflective* [Clavel and Meseguer, 1996; Clavel and Meseguer, 1996a] has very important practical consequences for its use as a logical framework. Note that a representation map $\Psi : \mathcal{L} \to \mathit{RWLogic}$ for a logic $\mathcal{L}$ is by its very nature a *metatheoretic* construction above the object levels of both $\mathcal{L}$ and *RWLogic*. In particular, Ψ includes as one of its key components a function $\Psi_{\mathbf{Th}} : \mathbf{Th}_{\mathcal{L}} \to \mathbf{Th}_{\mathit{RWLogic}}$ translating theories in $\mathcal{L}$ into rewrite theories. However, thanks to the fact that the finitely presentable rewrite theories can be reified as an abstract data type `RWL-ADT`, for $\mathcal{L}$ a logic having a finitary presentation of its syntax and its deduction rules, and such that Ψ maps finitely presented theories in $\mathcal{L}$ to finitely presented rewrite theories, we can often *reify* a metatheoretic construction such as Ψ inside rewriting logic by first defining an abstract data type $\mathcal{L}$`-ADT` representing the finitely presentable theories of $\mathcal{L}$, and then reifying Ψ itself as an equationally defined function $\overline{\Psi} : \mathcal{L}\texttt{-ADT} \longrightarrow \texttt{RWL-ADT}$. In this way, the translation Ψ becomes itself expressible and executable inside rewriting logic.

1.2 Rewriting logic as a semantic framework

As we have already mentioned, the distinction between a logical system and a language or a model of computation is more and more in the eyes of the beholder, although of course efficiency considerations and the practical uses intended may indeed strongly influence the design choices. A good case in point is the isomorphism between the Petri net model of concurrent computation [Reisig, 1995] and the tensor fragment of linear logic [Girard, 1987] (see [Martí-Oliet and Meseguer, 1991] and references therein). Therefore, even though at the most basic mathematical level there may be little distinction between the general way in which a logic, a programming language,

a system, or a model of computation are represented in rewriting logic, the criteria and case studies to be used in order to judge the merits of rewriting logic as a semantic framework are different from those relevant for its use as a logical framework.

One important consideration is that, from a computational point of view, rewriting logic deduction is intrinsically *concurrent*. In fact, it was the search for a general concurrency model that would help unify the somewhat bewildering heterogeneity of existing models that provided the original impetus for the first investigations on rewriting logic [Meseguer, 1992]. Since the generality and naturalness with which many concurrency models can be expressed in rewriting logic has already been illustrated at length in [Meseguer, 1992], only a brief summary is given in this paper. However, the CCS [Milner, 1989] and the concurrent object-oriented programming models are discussed in some detail to provide relevant examples.

Concurrent object-oriented programming is of particular interest. Given that the semantics of object-oriented programs is still poorly understood, and that the semantics of concurrent object-oriented systems is even less well understood, the ease with which rewriting logic can be used to give a precise semantics to concurrent object-oriented programs and to make such programs declarative is quite encouraging. In this paper, only the basic ideas of such a semantics are sketched; a much more detailed account can be found in [Meseguer, 1993].

The similarities between rewriting logic and structural operational semantics [Plotkin, 1981; Kahn, 1987] already noted in [Meseguer, 1992] are further explored in this paper. We give examples showing that different styles of structural operational semantics can be regarded as special cases of rewriting logic. The two main differences are the greater expressive power of rewriting logic due to the ability for rewriting modulo user-definable axioms, and the fact that rewriting logic is a full-fledged logic with both a proof and a model theory, whereas structural operational semantics accounts are only proof-theoretic.

Deduction with constraints can greatly increase the efficiency of theorem provers and logic programming languages. The most classical constraint solving algorithm is syntactic unification, which corresponds to solving equations in a free algebra, the so-called Herbrand model, and is used in resolution. However, much more efficient deduction techniques than those afforded by resolution can be obtained by building in additional knowledge of special theories in the form of constraint solving algorithms such as, for example, semantic unification, or equalities and inequalities in a numerical domain. In the past few years many authors have become aware that many constraint solving algorithms can be specified declaratively using rewrite rules. However, since constraint solving is usually nondeterministic, the usual equational logic interpretation of rewrite rules is clearly inadequate as a mathematical semantics. By contrast, rewriting logic completely avoids

such inadequacies and can serve as a semantic framework for logical systems and languages using constraints, including parallel ones.

The frame problem in artificial intelligence is caused by the need, typical of classical logic representations, to specify changes of state by stating not only what changes, but also what does not change. This is basically due to the essentially Platonic character of classical logic. Since rewriting logic is by design a logic of change that allows sound and complete deductions about the transitions of a system whose basic changes are axiomatized by rewrite rules, the difficulties associated with the frame problem disappear [Martí-Oliet and Meseguer, 1999]. In addition, the conservative mappings of Horn logic with equality and of linear logic studied in Sections 4.2 and 4.3, respectively, directly show how other logics of change recently proposed [Hölldobler and Schneeberger, 1990; Große *et al.*, 1996; Große *et al.*, 1992; Masseron *et al.*, 1990; Masseron *et al.*, 1993] can be subsumed as special cases. Added benefits include the straightforward support for concurrent change and the logical support for object-oriented representation.

The paper begins with a summary of the theory of general logics proposed in [Meseguer, 1989] that provides the conceptual basis for our discussion of logical frameworks. Then the rules of deduction and the model theory of rewriting logic are introduced, and the Maude and MaudeLog languages based on rewriting logic are briefly discussed. This is followed by a section presenting examples of logics representable in the rewriting logic framework. The role of rewriting logic as a semantic framework is then discussed and illustrated with examples. The paper ends with some concluding remarks.

2 GENERAL LOGICS

A general axiomatic theory of logics should adequately cover all the key ingredients of a logic. These include: a *syntax*, a notion of *entailment* of a sentence from a set of axioms, a notion of *model*, and a notion of *satisfaction* of a sentence by a model. A flexible axiomatic notion of a *proof calculus*, in which proofs of entailments, not just the entailments themselves, are first class citizens should also be included. This section gives a brief review of the required notions and axioms that will be later used in our treatment of rewriting logic as a logical framework; a more detailed account with many examples can be found in [Meseguer, 1989].

2.1 Syntax

Syntax can typically be given by a *signature* Σ providing a grammar on which to build *sentences*. For first-order logic, a typical signature consists of a list of function symbols and a list of predicate symbols, each with a prescribed number of arguments, which are used to build up sentences by

means of the usual logical connectives. For our purposes, it is enough to assume that for each logic there is a category **Sign** of possible signatures for it, and a functor *sen* assigning to each signature Σ the set $sen(\Sigma)$ of all its sentences.

2.2 Entailment systems

For a given signature Σ in **Sign**, *entailment* (also called *provability*) of a sentence $\varphi \in sen(\Sigma)$ from a set of axioms $\Gamma \subseteq sen(\Sigma)$ is a relation $\Gamma \vdash \varphi$ which holds if and only if we can prove φ from the axioms Γ using the rules of the logic. We make this relation relative to a signature.

In what follows, $|\mathcal{C}|$ denotes the collection of objects of a category $\mathcal{C}$.

DEFINITION 1. [Meseguer, 1989] An *entailment system* is a triple $\mathcal{E} = (\mathbf{Sign}, sen, \vdash)$ such that

- **Sign** is a category whose objects are called *signatures*,
- $sen : \mathbf{Sign} \longrightarrow \mathbf{Set}$ is a functor associating to each signature Σ a corresponding set of Σ-*sentences*, and
- $\vdash$ is a function associating to each $\Sigma \in |\mathbf{Sign}|$ a binary relation $\vdash_\Sigma \subseteq \mathcal{P}(sen(\Sigma)) \times sen(\Sigma)$ called Σ-*entailment* such that the following properties are satisfied:
  1. *reflexivity*: for any $\varphi \in sen(\Sigma)$, $\{\varphi\} \vdash_\Sigma \varphi$,
  2. *monotonicity*: if $\Gamma \vdash_\Sigma \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \varphi$,
  3. *transitivity*: if $\Gamma \vdash_\Sigma \varphi_i$, for all $i \in I$, and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$,
  4. $\vdash$-*translation*: if $\Gamma \vdash_\Sigma \varphi$, then for any $H : \Sigma \to \Sigma'$ in **Sign**, $sen(H)(\Gamma) \vdash_{\Sigma'} sen(H)(\varphi)$.

Except for the explicit treatment of syntax translations, the axioms are very similar to Scott's axioms for a consequence relation [Scott, 1974].

DEFINITION 2. [Meseguer, 1989] Given an entailment system $\mathcal{E}$, its category **Th** of *theories* has as objects pairs $T = (\Sigma, \Gamma)$ with Σ a signature and $\Gamma \subseteq sen(\Sigma)$. A *theory morphism* $H : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$ is a signature morphism $H : \Sigma \to \Sigma'$ such that if $\varphi \in \Gamma$, then $\Gamma' \vdash_{\Sigma'} sen(H)(\varphi)$.

A theory morphism $H : (\Sigma, \Gamma) \longrightarrow (\Sigma', \Gamma')$ is called *axiom-preserving* if it satisfies the condition that $sen(H)(\Gamma) \subseteq \Gamma'$. This defines a subcategory $\mathbf{Th}_0$ with the same objects as **Th** but with morphisms restricted to be axiom-preserving theory morphisms. Notice that the category $\mathbf{Th}_0$ does not depend at all on the entailment relation $\vdash$.

2.3 Institutions

The axiomatization of a model theory is due to the seminal work on *institutions* by Goguen and Burstall [1984; 1992].

DEFINITION 3. [Goguen and Burstall, 1984] An *institution* is a 4-tuple $\mathcal{I} = (\mathbf{Sign}, sen, \mathbf{Mod}, \models)$ such that

- **Sign** is a category whose objects are called *signatures*,
- $sen : \mathbf{Sign} \longrightarrow \mathbf{Set}$ is a functor associating to each signature Σ a set of Σ-*sentences*,
- $\mathbf{Mod} : \mathbf{Sign} \longrightarrow \mathbf{Cat}^{op}$ is a functor that gives for each signature Σ a category whose objects are called Σ-*models*, and
- $\models$ is a function associating to each $\Sigma \in |\mathbf{Sign}|$ a binary relation $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times sen(\Sigma)$ called Σ-*satisfaction* satisfying the following *satisfaction condition* for each $H : \Sigma \to \Sigma'$ in **Sign**: for all $M' \in |\mathbf{Mod}(\Sigma')|$ and all $\varphi \in sen(\Sigma)$,

$$M' \models_{\Sigma'} sen(H)(\varphi) \iff \mathbf{Mod}(H)(M') \models_\Sigma \varphi.$$

The satisfaction condition just requires that, for any syntax translation between two signatures, a model of the second signature satisfies a translated sentence if and only if the translation of this model satisfies the original sentence. Note that **Mod** is a contravariant functor, that is, translations of models go backwards.

Given a set of Σ-sentences Γ, we define the category $\mathbf{Mod}(\Sigma, \Gamma)$ as the full subcategory of $\mathbf{Mod}(\Sigma)$ determined by those models $M \in |\mathbf{Mod}(\Sigma)|$ that satisfy all the sentences in Γ, i.e., $M \models_\Sigma \varphi$ for each $\varphi \in \Gamma$.

Since the definition above of the category of theories $\mathbf{Th}_0$ only depends on signatures and sentences, it also makes sense for an institution.

2.4 Logics

Defining a *logic* is now almost trivial.

DEFINITION 4. [Meseguer, 1989] A *logic* is a 5-tuple $\mathcal{L} = (\mathbf{Sign}, sen, \mathbf{Mod}, \vdash, \models)$ such that:

- $(\mathbf{Sign}, sen, \vdash)$ is an entailment system,
- $(\mathbf{Sign}, sen, \mathbf{Mod}, \models)$ is an institution,

and the following *soundness condition* is satisfied: for any $\Sigma \in |\mathbf{Sign}|$, $\Gamma \subseteq sen(\Sigma)$, and $\varphi \in sen(\Sigma)$,

$$\Gamma \vdash_\Sigma \varphi \implies \Gamma \models_\Sigma \varphi,$$

where, by definition, the relation $\Gamma \models_\Sigma \varphi$ holds if and only if $M \models_\Sigma \varphi$ holds for any model M that satisfies all the sentences in Γ.

The logic is called *complete* if the above implication is in fact an equivalence.

2.5 Proof calculi

A given logic may admit many different proof calculi. For example, in first-order logic we have Hilbert style, natural deduction, and sequent calculi among others, and the way in which proofs are represented and generated by rules of deduction is different for each of these calculi. It is useful to make proofs relative to a given theory T whose axioms we are allowed to use in order to prove theorems.

A proof calculus associates to each theory T a *structure* $P(T)$ of proofs that use axioms of T as hypotheses. The structure $P(T)$ typically has an *algebraic structure* of some kind so that we can obtain new proofs out of previously given proofs by operations that mirror the rules of deduction of the calculus in question. We need not make a choice about the particular types of algebraic structures that should be allowed for different proof calculi; we can abstract from such choices by simply saying that for a given proof calculus there is a category **Str** of such structures and a functor $P : \mathbf{Th}_0 \longrightarrow \mathbf{Str}$ assigning to each theory T its structure of proofs $P(T)$. Of course, it should be possible to extract from $P(T)$ the underlying set $\mathit{proofs}(T)$ of all the proofs of theorems of the theory T, and this extraction should be functorial. Also, each proof, whatever it is, should contain information about what theorem it is a proof of; this can be formalized by postulating a "projection function" π_T (parameterized by T in a natural way) that maps each proof $p \in \mathit{proofs}(T)$ to the sentence φ that it proves. Of course, each theorem of T must have at least one proof, and sentences that are not theorems should have no proof. To summarize, a *proof calculus* [Meseguer, 1989] consists of an entailment system together with:

- A functorial assignment P of a structure $P(T)$ to each theory T.
- An additional functorial assignment of a set $\mathit{proofs}(T)$ to each structure $P(T)$.
- A natural function π_T assigning a sentence to each proof $p \in \mathit{proofs}(T)$ and such that, for Γ the axioms of T, a sentence φ is in the image of π_T if and only if $\Gamma \vdash \varphi$.

It is quite common to encounter proof systems of a specialized nature. In these calculi, only certain signatures are admissible as syntax—e.g., finite signatures—, only certain sentences are allowed as axioms, and only certain sentences—possibly different from the axioms—are allowed as conclusions. The obvious reason for introducing such specialized calculi is that

proofs are simpler under the given restrictions. In computer science the choice between an efficient and an inefficient calculus may have dramatic practical consequences. For logic programming languages, such calculi do (or should) coincide with what is called their *operational semantics*, and mark the difference between a hopelessly inefficient theorem prover and an efficient programming language. In practice, of course, we are primarily interested in proof calculi and proof subcalculi that are computationally effective. This is axiomatized by the notion of an *(effective) proof subcalculus* which can be found in [Meseguer, 1989].

2.6 Mapping logics

The advantage of having an axiomatic theory of logics is that the "space" of all logics (or that of all entailment systems, institutions, proof calculi, etc.) becomes well understood. This space is not just a collection of objects bearing no relationship to each other. In fact, the most interesting fruit of the theory of general logics outlined in this section is that it gives us a method for *relating* logics in a general and systematic way, and to exploit such relations in many applications. The simplest kind of relation is a *sublogic* (subentailment system, etc.) relation. Thus, first-order equational logic and Horn logic are both sublogics of first-order logic with equality. However, more subtle and general ways of relating logics are possible. For example, we may want to represent the universal fragment of first-order logic in a purely functional way by taking all the predicates and formulas to be *functions* whose value is either *true* or *false* so that a universal formula then becomes an equation equating a given term to *true*. The general way of relating logics (entailment systems, etc.) is to consider *maps* that interpret one logic into another. A detailed treatment of such maps is given in [Meseguer, 1989]; here we summarize some of the key ideas.

Let us first discuss in some detail *maps of entailment systems*. Basically, a map of entailment systems $\mathcal{E} \longrightarrow \mathcal{E}'$ maps the language of $\mathcal{E}$ to that of $\mathcal{E}'$ in a way that respects the entailment relation. This means that signatures of $\mathcal{E}$ are functorially mapped to signatures of $\mathcal{E}'$, and that sentences of $\mathcal{E}$ are mapped to sentences of $\mathcal{E}'$ in a way that is coherent with the mapping of their corresponding signatures. In addition, such a mapping α must respect the entailment relations $\vdash$ of $\mathcal{E}$ and $\vdash'$ of $\mathcal{E}'$, i.e., we must have $\Gamma \vdash \varphi \;\Rightarrow\; \alpha(\Gamma) \vdash' \alpha(\varphi)$. It turns out that for many interesting applications, including the functional representation of first-order logic sketched above, one wants to be more general and allow maps that send a signature of $\mathcal{E}$ to a *theory* of $\mathcal{E}'$. These maps extend to maps between theories, and in this context the coherence with the mapping at the level of signatures is expressed by the notion of *sensible functor* defined in [Meseguer, 1989].

DEFINITION 5. [Meseguer, 1989] Given entailment systems $\mathcal{E} = (\mathbf{Sign}, sen, \vdash)$ and $\mathcal{E}' = (\mathbf{Sign}', sen', \vdash')$, a *map of entailment systems* $(\Phi, \alpha) : \mathcal{E} \longrightarrow \mathcal{E}'$ consists of a natural transformation $\alpha : sen \Rightarrow \Phi; sen'$ and an α-sensible functor[1] $\Phi : \mathbf{Th}_0 \longrightarrow \mathbf{Th}'_0$ satisfying the following property:

$$\Gamma \vdash_\Sigma \varphi \implies \Gamma' \cup \alpha_\Sigma(\Gamma) \vdash'_{\Sigma'} \alpha_\Sigma(\varphi),$$

where, by convention, $(\Sigma', \Gamma') = \Phi(\Sigma, \Gamma)$.

We say that (Φ, α) is a *conservative map of entailment systems* when the above implication is an equivalence.

The property of being conservative may be essential for many applications. For example, since proof calculi are in a sense computational engines on which the design and implementation of theorem provers and logic programming languages can be based, we can view the establishment of a map of proof calculi having nice properties, such as conservativity, as a proof of correctness for a *compiler* that permits implementing a system based on the first calculus in terms of another system based on the second. Besides establishing correctness, the map itself specifies the compilation function.

A *map of institutions*[2] $\mathcal{I} \longrightarrow \mathcal{I}'$ is similar in its syntax part to a map of entailment systems. In addition, for models we have a natural functor $\beta : \mathbf{Mod}'(\Phi(\Sigma)) \longrightarrow \mathbf{Mod}(\Sigma)$ "backwards" from the models in $\mathcal{I}'$ of a translated signature $\Phi(\Sigma)$ to the models in $\mathcal{I}$ of the original signature Σ, and such a mapping respects the satisfaction relations $\models$ of $\mathcal{I}$ and $\models'$ of $\mathcal{I}'$, in the sense that $M' \models' \alpha(\varphi) \iff \beta(M') \models \varphi$.

DEFINITION 6. [Meseguer, 1989] Given institutions $\mathcal{I} = (\mathbf{Sign}, sen, \mathbf{Mod}, \models)$ and $\mathcal{I}' = (\mathbf{Sign}', sen', \mathbf{Mod}', \models')$, a *map of institutions* $(\Phi, \alpha, \beta) : \mathcal{I} \longrightarrow \mathcal{I}'$ consists of a natural transformation $\alpha : sen \Rightarrow \Phi; sen'$, an α-sensible functor $\Phi : \mathbf{Th}_0 \longrightarrow \mathbf{Th}'_0$, and a natural transformation $\beta : \Phi^{op}; \mathbf{Mod}' \Rightarrow \mathbf{Mod}$ such that for each $\Sigma \in |\mathbf{Sign}|$, $\varphi \in sen(\Sigma)$, and $M' \in |\mathbf{Mod}'(\Phi(\Sigma, \emptyset))|$ the following property is satisfied:

$$M' \models'_{\Sigma'} \alpha_\Sigma(\varphi) \iff \beta_{(\Sigma,\emptyset)}(M') \models_\Sigma \varphi,$$

where Σ' is the signature of the theory $\Phi(\Sigma, \emptyset)$.

A *map of logics* has now a very simple definition. It consists of a pair of maps: one for the underlying entailment systems, and another for the underlying institutions, such that both maps agree on how they translate signatures and sentences.

[1] We refer to [Meseguer, 1989] for the detailed definition of α-sensible functor. Basically, what is required is that the provable consequences of the theory $\Phi(\Sigma, \Gamma)$ are entirely determined by $\Phi(\Sigma, \emptyset)$ and by $\alpha(\Gamma)$. Note that α depends only on signatures, not theories.

[2] Such maps are different from the "institution morphisms" considered by Goguen and Burstall in [1984; 1992].

DEFINITION 7. [Meseguer, 1989] Given logics $\mathcal{L} = (\mathbf{Sign}, sen, \mathbf{Mod}, \vdash, \models)$ and $\mathcal{L}' = (\mathbf{Sign}', sen', \mathbf{Mod}', \vdash', \models')$, a *map of logics* $(\Phi, \alpha, \beta) : \mathcal{L} \longrightarrow \mathcal{L}'$ consists of a functor $\Phi : \mathbf{Th}_0 \longrightarrow \mathbf{Th}'_0$, and natural transformations $\alpha : sen \Rightarrow \Phi; sen'$ and $\beta : \Phi^{op}; \mathbf{Mod}' \Rightarrow \mathbf{Mod}$ such that:

- $(\Phi, \alpha) : (\mathbf{Sign}, sen, \vdash) \longrightarrow (\mathbf{Sign}', sen', \vdash')$ is a map of entailment systems, and
- $(\Phi, \alpha, \beta) : (\mathbf{Sign}, sen, \mathbf{Mod}, \models) \longrightarrow (\mathbf{Sign}', sen', \mathbf{Mod}', \models')$ is a map of institutions.

We say that (Φ, α, β) is *conservative* when if (Φ, α) is so as a map of entailment systems.

There is also a notion of map of proof calculi, for which we refer the reader to [Meseguer, 1989].

2.7 The idea of a logical framework

As we have already explained in the introduction, viewed from the perspective of a general space of logics that can be related to each other by means of mappings, the quest for a *logical framework* can be understood as the search within such a space for a logic $\mathcal{F}$ (the *framework* logic) such that many other logics (the *object* logics) such as, say, $\mathcal{L}$ can be represented in $\mathcal{F}$ by means of mappings $\mathcal{L} \longrightarrow \mathcal{F}$ that have good enough properties. The minimum requirement that seems reasonable to make on a representation map $\mathcal{L} \longrightarrow \mathcal{F}$ is that it should be a *conservative* map of entailment systems. Under such circumstances, we can reduce issues of provability in $\mathcal{L}$ to issues of provability in $\mathcal{F}$, by mapping the theories and sentences of $\mathcal{L}$ into $\mathcal{F}$ using the conservative representation map. Given a computer implementation of deduction in $\mathcal{F}$, we can use the conservative map to prove theorems in $\mathcal{L}$ by proving the corresponding translations in $\mathcal{F}$. In this way, the implementation for $\mathcal{F}$ can be used as a generic theorem prover for many logics.

However, since maps between logics can, as we have seen, respect additional logical structure such as the model theory or the proofs, in some cases a representation map into a logical framework may be particularly informative because, in addition to being a conservative map of entailment systems, it is also a map of institutions, or a map of proof calculi. For example, when rewriting logic is chosen as a logical framework, appropriate representation maps for equational logic, Horn logic, and propositional linear logic can be shown to be maps of institutions also (see Section 4). In general, however, since the model theories of different logics can be very different from each other, it is not reasonable to expect or require that the representation maps into a logical framework will always be maps of institutions. Nevertheless,

what it can always be done is to "borrow" the additional logical structure that $\mathcal{F}$ may have (institution, proof calculus) to endow $\mathcal{L}$ with such a structure, so that the representation map does indeed preserve the extra structure [Cerioli and Meseguer, 1996].

Having criteria for the adequacy of maps representing logics in a logical framework is not enough. An equally important issue is having criteria for the *generality* of a logical framework, so that it is in fact justified to call it by that name. That is, given a candidate logical framework $\mathcal{F}$, how many logics can be adequately represented in $\mathcal{F}$? We can make this question precise by defining the *scope* of a logical framework $\mathcal{F}$ as the class of entailment systems $\mathcal{E}$ having conservative maps of entailment systems $\mathcal{E} \longrightarrow \mathcal{F}$. In this regard, the axioms of the theory of general logics that we have presented are probably too general; without adding further assumptions it is not reasonable to expect that we can find a logical framework $\mathcal{F}$ whose scope is the class of *all* entailment systems. A much more reasonable goal is finding an $\mathcal{F}$ whose scope includes all entailment systems of "practical interest," having finitary presentations of their syntax and their rules of deduction. Axiomatizing such finitely presentable entailment systems and proof calculi so as to capture—in the spirit of the more general axioms that we have presented, but with stronger requirements—all logics of "practical interest" (at least for computational purposes) is a very important research task.

Another important property that can help measuring the suitability of a logic $\mathcal{F}$ as a logical framework is its *representational adequacy*, understood as the naturalness and ease with which entailment systems can be represented, so that the representation $\mathcal{E} \longrightarrow \mathcal{F}$ mirrors $\mathcal{E}$ as closely as possible. That is, a framework requiring very complicated encodings for many object logics of interest is less representationally adequate than one for which most logics can be represented in a straightforward way, so that there is in fact little or no "distance" between an object logic and its corresponding representation. Although at present we lack a precise definition of this property, it is quite easy to observe its absence in particular examples. We view representational adequacy as a very important practical criterion for judging the relative merits of different logical frameworks.

In this paper, we present rewriting logic as a logic that seems to have particularly good properties as a logical framework. We conjecture that the scope of rewriting logic contains all entailment systems of "practical interest" for a reasonable axiomatization of such systems.

2.8 Reflection

We give here a brief summary of the notion of a universal theory in a logic and of a reflective entailment system introduced in [Clavel and Meseguer, 1996]. These notions axiomatize reflective logics within the theory of general

logics [Meseguer, 1989]. We focus here on the simplest case, namely entailment systems. However, reflection at the proof calculus level—where not only sentences, but also proofs are reflected—is also very useful; definitions for that case are also in [Clavel and Meseguer, 1996].

A reflective logic is a logic in which important aspects of its metatheory can be represented at the object level in a consistent way, so that the object-level representation correctly simulates the relevant metatheoretic aspects. Two obvious metatheoretic notions that can be so reflected are theories and the entailment relation $\vdash$. This leads us to the notion of a universal theory. However, universality may not be absolute, but only relative to a class $\mathcal{C}$ of *representable* theories. Typically, for a theory to be representable at the object level, it must have a finitary description in some way—say, being recursively enumerable—so that it can be represented as a piece of language.

Given an entailment system $\mathcal{E}$ and a set of theories $\mathcal{C}$, a theory U is $\mathcal{C}$-*universal* if there is a recursive injective function, called a *representation function*,

$$\overline{(\_ \vdash \_)} : \bigcup_{T \in \mathcal{C}} \{T\} \times sen(T) \longrightarrow sen(U)$$

such that for each $T \in \mathcal{C}$, $\varphi \in sen(T)$,

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi}.$$

If, in addition, $U \in \mathcal{C}$, then the entailment system $\mathcal{E}$ is called $\mathcal{C}$-*reflective*.

Note that in a reflective entailment system, since U itself is representable, representation can be iterated, so that we immediately have a "reflective tower"

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi} \iff U \vdash \overline{U \vdash \overline{T \vdash \varphi}} \ \ldots$$

3 REWRITING LOGIC

This section gives the rules of deduction and semantics of rewriting logic, and explains its computational meaning. The Maude and MaudeLog languages, based on rewriting logic, are also briefly discussed.

3.1 Basic universal algebra

Rewriting logic is parameterized with respect to the version of the underlying equational logic, which can be unsorted, many-sorted, order-sorted, or the recently developed membership equational logic [Bouhoula *et al.*, 2000; Meseguer, 1998]. For the sake of simplifying the exposition, we treat here the *unsorted* case.

A set Σ of function symbols is a ranked alphabet $\Sigma = \{\Sigma_n \mid n \in \mathbb{N}\}$. A Σ-algebra is then a set A together with an assignment of a function

$f_A : A^n \longrightarrow A$ for each $f \in \Sigma_n$ with $n \in \mathbb{N}$. We denote by T_Σ the Σ-algebra of ground Σ-terms, and by $T_\Sigma(X)$ the Σ-algebra of Σ-terms with variables in a set X. Similarly, given a set E of Σ-equations, $T_{\Sigma,E}$ denotes the Σ-algebra of equivalence classes of ground Σ-terms modulo the equations E; in the same way, $T_{\Sigma,E}(X)$ denotes the Σ-algebra of equivalence classes of Σ-terms with variables in X modulo the equations E. Let $[t]_E$ or just $[t]$ denote the E-equivalence class of t.

Given a term $t \in T_\Sigma(\{x_1, \ldots, x_n\})$ and terms $u_1, \ldots, u_n \in T_\Sigma(X)$, we denote $t(u_1/x_1, \ldots, u_n/x_n)$ the term in $T_\Sigma(X)$ obtained from t by *simultaneously substituting* u_i for x_i, $i = 1, \ldots, n$. To simplify notation, we denote a sequence of objects $a_1, \ldots, a_n$ by $\overline{a}$; with this notation, $t(u_1/x_1, \ldots, u_n/x_n)$ can be abbreviated to $t(\overline{u}/\overline{x})$.

3.2 The rules of rewriting logic

A *signature* in rewriting logic is a pair (Σ, E) with Σ a ranked alphabet of function symbols and E a set of Σ-equations. Rewriting will operate on equivalence classes of terms modulo the set of equations E. In this way, we free rewriting from the syntactic constraints of a term representation and gain a much greater flexibility in deciding what counts as a *data structure*; for example, string rewriting is obtained by imposing an associativity axiom, and multiset rewriting by imposing associativity and commutativity. Of course, standard term rewriting is obtained as the particular case in which the set E of equations is empty. Techniques for rewriting modulo equations have been studied extensively [Dershowitz and Jouannaud, 1990] and can be used to implement rewriting modulo many equational theories of interest.

Given a signature (Σ, E), *sentences* of rewriting logic are "sequents" of the form $[t]_E \longrightarrow [t']_E$, where t and t' are Σ-terms possibly involving some variables from the countably infinite set $X = \{x_1, \ldots, x_n, \ldots\}$. A *theory* in this logic, called a rewrite theory, is a slight generalization of the usual notion of theory as in Definition 2 in that, in addition, we allow the axioms—in this case the sequents $[t]_E \longrightarrow [t']_E$—to be labelled. This is very natural for many applications, and customary for automata—viewed as labelled transition systems—and for Petri nets, which are both particular instances of our definition.

DEFINITION 8. A *rewrite theory* $\mathcal{R}$ is a 4-tuple $\mathcal{R} = (\Sigma, E, L, R)$ where Σ is a ranked alphabet of function symbols, E is a set of Σ-equations, L is a set of *labels*, and R is a set of pairs $R \subseteq L \times T_{\Sigma,E}(X)^2$ whose first component is a label and whose second component is a pair of E-equivalence classes of terms, with $X = \{x_1, \ldots, x_n, \ldots\}$ a countably infinite set of variables. Elements of R are called *rewrite rules*[3]. We understand a

[3] To simplify the exposition the rules of the logic are given for the case of *unconditional* rewrite rules. However, all the ideas presented here have been extended to conditional

rule $(r, ([t], [t']))$ as a labelled sequent and use for it the notation $r : [t] \longrightarrow [t']$. To indicate that $\{x_1, \ldots, x_n\}$ is the set of variables occurring in either t or t', we write $r : [t(x_1, \ldots, x_n)] \longrightarrow [t'(x_1, \ldots, x_n)]$, or in abbreviated notation $r : [t(\overline{x})] \longrightarrow [t'(\overline{x})]$.

Given a rewrite theory $\mathcal{R}$, we say that $\mathcal{R}$ *entails* a sequent $[t] \longrightarrow [t']$ and write $\mathcal{R} \vdash [t] \longrightarrow [t']$ if and only if $[t] \longrightarrow [t']$ can be obtained by finite application of the following *rules of deduction*:

1. **Reflexivity**. For each $[t] \in T_{\Sigma,E}(X)$,
$$\frac{}{[t] \longrightarrow [t]}.$$
2. **Congruence**. For each $f \in \Sigma_n$, $n \in \mathbb{N}$,
$$\frac{[t_1] \longrightarrow [t'_1] \quad \ldots \quad [t_n] \longrightarrow [t'_n]}{[f(t_1, \ldots, t_n)] \longrightarrow [f(t'_1, \ldots, t'_n)]}.$$
3. **Replacement**. For each rewrite rule in the theory R of the form $r : [t(x_1, \ldots, x_n)] \longrightarrow [t'(x_1, \ldots, x_n)]$,
$$\frac{[w_1] \longrightarrow [w'_1] \quad \ldots \quad [w_n] \longrightarrow [w'_n]}{[t(\overline{w}/\overline{x})] \longrightarrow [t'(\overline{w'}/\overline{x})]}.$$
4. **Transitivity**.
$$\frac{[t_1] \longrightarrow [t_2] \qquad [t_2] \longrightarrow [t_3]}{[t_1] \longrightarrow [t_3]}.$$

Equational logic (modulo a set of axioms E) is obtained from rewriting logic by adding the following rule:

5. **Symmetry**.
$$\frac{[t_1] \longrightarrow [t_2]}{[t_2] \longrightarrow [t_1]}.$$

With this new rule, sequents derivable in equational logic are *bidirectional*; therefore, in this case we can adopt the notation $[t] \leftrightarrow [t']$ throughout and call such bidirectional sequents *equations*.

A nice consequence of having defined rewriting logic is that concurrent rewriting, rather than emerging as an operational notion, actually coincides with deduction in such a logic.

DEFINITION 9. Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, a (Σ, E)-sequent $[t] \longrightarrow [t']$ is called a *concurrent $\mathcal{R}$-rewrite* (or just a *rewrite*) if and only if it can be derived from $\mathcal{R}$ by means of the rules 1-4, i.e., $\mathcal{R} \vdash [t] \longrightarrow [t']$.

---

rules in [Meseguer, 1992] with very general rules of the form
$$r : [t] \longrightarrow [t'] \ \textit{if} \ [u_1] \longrightarrow [v_1] \wedge \ldots \wedge [u_k] \longrightarrow [v_k].$$
This increases considerably the expressive power of rewrite theories, as illustrated by several of the examples presented in this paper.

3.3 The meaning of rewriting logic

A logic worth its salt should be understood as a method of correct reasoning about some class of entities, not as an empty formal game. For equational logic, the entities in question are sets, functions between them, and the relation of identity between elements. For rewriting logic, the entities in question are *concurrent systems* having *states*, and evolving by means of *transitions*. The *signature* of a rewrite theory describes a particular structure for the states of a system—e.g., multiset, binary tree, etc.—so that its states can be distributed according to such a structure. The *rewrite rules* in the theory describe which *elementary local transitions* are possible in the distributed state by concurrent local transformations. The rules of rewriting logic allow us to reason correctly about which *general* concurrent transitions are possible in a system satisfying such a description. Clearly, concurrent systems should be the *models* giving a semantic interpretation to rewriting logic, in the same way that algebras are the models giving a semantic interpretation to equational logic. A precise account of the model theory of rewriting logic, giving rise to an initial model semantics for Maude modules and fully consistent with the above system-oriented interpretation, is sketched in Section 3.5 and developed in full detail for the more general conditional case in [Meseguer, 1992].

Therefore, in rewriting logic a sequent $[t] \longrightarrow [t']$ should not be read as "$[t]$ *equals* $[t']$," but as "$[t]$ *becomes* $[t']$." Clearly, rewriting logic is a logic of *becoming* or *change*, not a logic of equality in a static sense. The apparently innocent step of adding the symmetry rule is in fact a *very strong* restriction, namely assuming that *all change is reversible*, thus bringing us into a timeless Platonic realm in which "before" and "after" have been identified.

A related observation, which is particularly important for the use of rewriting logic as a logical framework, is that $[t]$ should not be understood as a *term* in the usual first-order logic sense, but as a *proposition* or *formula*—built up using the *connectives* in Σ—that asserts being in a certain *state* having a certain *structure*. However, unlike most other logics, the logical connectives Σ and their structural properties E are entirely *user-definable*. This provides great flexibility for considering many different state structures and makes rewriting logic very general in its capacity to deal with many different types of concurrent systems, and also in its capacity to represent many different logics. For the case of concurrent systems, this generality is discussed at length in [Meseguer, 1992] (see also [Martí-Oliet and Meseguer, 1999] for the advantages of this generality in the context of unifying AI logics of action). In a similar vein, but with a broader focus, Section 5 discusses the advantages of rewriting logic as a general semantic framework in which to specify and prototype languages and systems. Finally, Section 4 explores the generality of rewriting logic as a logical framework in which logics can

be represented and prototyped.

In summary, the rules of rewriting logic are rules to reason about *change in a concurrent system*, or, alternatively, metarules for reasoning about *deduction in a logical system*. They allow us to draw valid conclusions about the evolution of the system from certain basic types of change known to be possible, or, in the alternative viewpoint, about the correct deductions possible in a logical system. Our present discussion is summarized as follows:

State	$\leftrightarrow$	*Term*	$\leftrightarrow$	*Proposition*
Transition	$\leftrightarrow$	*Rewriting*	$\leftrightarrow$	*Deduction*
Distributed Structure	$\leftrightarrow$	*Algebraic Structure*	$\leftrightarrow$	*Propositional Structure*

Section 4 will further clarify and illustrate each of the correspondences in the last two columns of the diagram, and Section 5 will do the same for the first two columns.

3.4 The Maude and MaudeLog languages

Rewriting logic can be used directly as a wide spectrum language supporting specification, rapid prototyping, and programming of concurrent systems. As explained later in this paper, rewriting logic can also be used as a logical framework in which other logics can be naturally represented, and as a semantic framework for specifying languages and systems. The Maude language [Meseguer, 1993; Clavel *et al.*, 1996] supports all these uses of rewriting logic in a particularly modular way in which modules are rewrite theories and in which functional modules with equationally defined data types can also be declared in a functional sublanguage. The examples given later in this paper illustrate the syntax of Maude. Details about the language design, its semantics, its parallel programming and wide spectrum capabilities, and its support of object-oriented programming can be found in [Meseguer, 1992; Meseguer and Winkler, 1992; Meseguer, 1993; Meseguer, 1993b]. Here we provide a very brief sketch that should be sufficient for understanding the examples presented later.

In Maude there are three kinds of *modules*:

1. *Functional modules*, introduced by the keyword `fmod`,
2. *System modules*, introduced by the keyword `mod`, and
3. *Object-oriented modules*, introduced by the keyword `omod`.

Object-oriented modules can be reduced to a special case of system modules for which a special syntax is used; therefore, in essence we only have functional and system modules. Maude's functional and system modules are respectively of the form

- `fmod` $\mathcal{E}$ `endfm`, and
- `mod` $\mathcal{R}$ `endm`,

for $\mathcal{E}$ an equational theory and $\mathcal{R}$ a rewrite theory[4]. In functional modules, equations are declared with the keywords `eq` or `ceq` (for conditional equations), and in system or object-oriented modules with the keywords `ax` or `cax`. In addition, certain equations, such as any combination of associativity, commutativity, or identity, for which rewriting modulo is provided, can be declared together with the corresponding operator using the keywords `assoc`, `comm`, `id`. Rules can only appear in system or object-oriented modules, and are declared with the keywords `rl` or `crl`.

In Maude a module can have *submodules*, which can be imported with either `protecting` or `including` qualifications stating the degree of integrity enjoyed by the submodule when imported by the supermodule.

The version of rewriting logic used for Maude in this paper is *order-sorted*[5]. This means that rewrite theories are typed (types are called *sorts*) and can have subtypes (subsorts), and that function symbols can be overloaded. In particular, functional modules are order-sorted equational theories [Goguen and Meseguer, 1992] and they form a sublanguage similar to OBJ [Goguen *et al.*, 2000].

Like OBJ, Maude has also *theories* to specify semantic requirements for interfaces and to make high level assertions about modules. They are of the three kinds:

1. *Functional theories*, introduced by the keyword `fth`,
2. *System theories*, introduced by the keyword `th`, and
3. *Object-oriented theories*, introduced by the keyword `oth`.

Also as OBJ, Maude has *parameterized modules* and *theories*, again of the three kinds, and *views* that are theory interpretations relating theories to modules or to other theories.

Maude can be further extended to a language called MaudeLog that unifies the paradigms of functional programming, Horn logic programming, and concurrent object-oriented programming. In fact, Maude's design is based on a general axiomatic notion of "logic programming language" based on the general axiomatic theory of logic sketched in Section 2 [Meseguer, 1989; Meseguer, 1992b]. Technically, a unification of paradigms is achieved by mapping the logics of each paradigm into a richer logic in which the

[4] This is somewhat inaccurate in the case of system modules having functional submodules because we have to "remember" that the submodule is functional.

[5] The latest version of Maude [Clavel *et al.*, 1996] is based on the recently developed membership equational logic, which extends order-sorted equational logic and at the same time has a simpler and more general model theory [Bouhoula *et al.*, 2000; Meseguer, 1998].

paradigms are unified. In the case of Maude and MaudeLog, what is done is to define a new logic (rewriting logic) in which concurrent computations, and in particular concurrent object-oriented computations, can be expressed in a natural way, and then to formally relate this logic to the logics of the functional and relational paradigms, i.e., to equational logic and to Horn logic, by means of maps of logics that provide a simple and rigorous unification of paradigms. As it has already been mentioned, we actually assume an order-sorted structure throughout, and therefore the logics in question are: order-sorted rewriting logic, denoted *OSRWLogic*, order-sorted equational logic, denoted *OSEqtl*, and order-sorted Horn logic, denoted *OSHorn*.

The logic of equational programming can be embedded within (order-sorted) rewriting logic by means of a map of logics

$$OSEqtl \longrightarrow OSRWLogic.$$

The details of this map of logics are discussed in Section 4.1. At the programming language level, such a map corresponds to the inclusion of Maude's functional modules (essentially identical to OBJ modules) within the language.

Since the power and the range of applications of a multiparadigm logic programming language can be substantially increased if it is possible to solve queries involving *logical variables* in the sense of relational programming, as in the Prolog language, we are naturally led to seek a unification of the three paradigms of functional, relational and concurrent object-oriented programming into a single multiparadigm logic programming language. This unification can be attained in a language extension of Maude called MaudeLog. The integration of Horn logic is achieved by a map of logics

$$OSHorn \longrightarrow OSRWLogic$$

that systematically relates order-sorted Horn logic to order-sorted rewriting logic. The details of this map are discussed in Section 4.2.

The difference between Maude and MaudeLog does not consist of any change in the underlying logic; indeed, both languages are based on rewriting logic, and both have rewrite theories as programs. It resides, rather, in an enlargement of the set of *queries* that can be presented, so that, while keeping the same syntax and models, in MaudeLog we also consider queries involving existential formulas of the form

$$\exists \overline{x} \quad [u_1(\overline{x})] \longrightarrow [v_1(\overline{x})] \ \wedge \ \ldots \ \wedge \ [u_k(\overline{x})] \longrightarrow [v_k(\overline{x})].$$

Therefore, the sentences and the deductive rules and mechanisms that are now needed require further extensions of rewriting logic deduction. In particular, solving such existential queries requires performing *unification*, specifically, given Maude's typing structure, order-sorted E-unification for a set E of structural axioms [Meseguer *et al.*, 1989].

3.5 The models of rewriting logic

We first sketch the construction of initial and free models for a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$. Such models capture nicely the intuitive idea of a "rewrite system" in the sense that they are systems whose states are E-equivalence classes of terms, and whose transitions are concurrent rewrites using the rules in R. By adopting a logical instead of a computational perspective, we can alternatively view such models as "logical systems" in which formulas are validly rewritten to other formulas by concurrent rewrites which correspond to proofs for the logic in question. Such models have a natural *category* structure, with states (or formulas) as objects, transitions (or proofs) as morphisms, and sequential composition as morphism composition, and in them dynamic behavior exactly corresponds to deduction.

Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, the model that we are seeking is a category $\mathcal{T}_{\mathcal{R}}(X)$ whose objects are equivalence classes of terms $[t] \in T_{\Sigma,E}(X)$ and whose morphisms are equivalence classes of "proof terms" representing proofs in rewriting deduction, i.e., concurrent $\mathcal{R}$-rewrites. The rules for generating such proof terms, with the specification of their respective domain and codomain, are given below; they just "decorate" with proof terms the rules 1-4 of rewriting logic. Note that we always use "diagrammatic" notation for morphism composition, i.e., $\alpha; \beta$ always means the composition of α *followed by* β.

1. **Identities**. For each $[t] \in T_{\Sigma,E}(X)$,

$$\frac{}{[t] : [t] \longrightarrow [t]}.$$

2. Σ**-structure**. For each $f \in \Sigma_n$, $n \in \mathbb{N}$,

$$\frac{\alpha_1 : [t_1] \longrightarrow [t'_1] \quad \ldots \quad \alpha_n : [t_n] \longrightarrow [t'_n]}{f(\alpha_1, \ldots, \alpha_n) : [f(t_1, \ldots, t_n)] \longrightarrow [f(t'_1, \ldots, t'_n)]}.$$

3. **Replacement**. For each rewrite rule $r : [t(\overline{x}^n)] \longrightarrow [t'(\overline{x}^n)]$ in R,

$$\frac{\alpha_1 : [w_1] \longrightarrow [w'_1] \quad \ldots \quad \alpha_n : [w_n] \longrightarrow [w'_n]}{r(\alpha_1, \ldots, \alpha_n) : [t(\overline{w}/\overline{x})] \longrightarrow [t'(\overline{w'}/\overline{x})]}.$$

4. **Composition**.

$$\frac{\alpha : [t_1] \longrightarrow [t_2] \qquad \beta : [t_2] \longrightarrow [t_3]}{\alpha; \beta : [t_1] \longrightarrow [t_3]}.$$

Convention. In the case when the same label r appears in two different rules of R, the "proof terms" $r(\overline{\alpha})$ can sometimes be ambiguous. We assume that such ambiguity problems have been resolved by disambiguating the

label r in the proof terms $r(\overline{\alpha})$ if necessary; with this understanding, we adopt the simpler notation $r(\overline{\alpha})$ to ease the exposition.

Each of the above rules of generation defines a different operation taking certain proof terms as arguments and returning a resulting proof term. In other words, proof terms form an algebraic structure $\mathcal{P}_{\mathcal{R}}(X)$ consisting of a graph with nodes $T_{\Sigma,E}(X)$, with identity arrows, and with operations f (for each $f \in \Sigma$), r (for each rewrite rule), and $\_;\_$ (for composing arrows). Our desired model $\mathcal{T}_{\mathcal{R}}(X)$ is the quotient of $\mathcal{P}_{\mathcal{R}}(X)$ modulo the following equations[6]:

1. **Category**.

   (a) *Associativity.* For all α, β, γ,

   $$(\alpha;\beta);\gamma = \alpha;(\beta;\gamma).$$

   (b) *Identities.* For each $\alpha : [t] \longrightarrow [t']$,

   $$\alpha;[t'] = \alpha \quad \text{and} \quad [t];\alpha = \alpha.$$

2. **Functoriality of the Σ-algebraic structure**. For each $f \in \Sigma_n$, $n \in \mathbb{N}$,

   (a) *Preservation of composition.* For all $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$,

   $$f(\alpha_1;\beta_1,\ldots,\alpha_n;\beta_n) = f(\alpha_1,\ldots,\alpha_n);f(\beta_1,\ldots,\beta_n).$$

   (b) *Preservation of identities.*

   $$f([t_1],\ldots,[t_n]) = [f(t_1,\ldots,t_n)].$$

3. **Axioms in E**. For each axiom $t(x_1,\ldots,x_n) = t'(x_1,\ldots,x_n)$ in E, for all $\alpha_1,\ldots,\alpha_n$,

   $$t(\alpha_1,\ldots,\alpha_n) = t'(\alpha_1,\ldots,\alpha_n).$$

4. **Exchange**. For each rule $r : [t(x_1,\ldots,x_n)] \longrightarrow [t'(x_1,\ldots,x_n)]$ in R,

   $$\frac{\alpha_1 : [w_1] \longrightarrow [w_1'] \quad \ldots \quad \alpha_n : [w_n] \longrightarrow [w_n']}{r(\overline{\alpha}) = r(\overline{[w]});t'(\overline{\alpha}) = t(\overline{\alpha});r(\overline{[w']})}.$$

[6] In the expressions appearing in the equations, when compositions of morphisms are involved, we always implicitly assume that the corresponding domains and codomains match.

Note that the set X of variables is actually a parameter of these constructions, and we need not assume X to be fixed and countable. In particular, for $X = \emptyset$, we adopt the notation $\mathcal{T}_{\mathcal{R}}$. The equations in 1 make $\mathcal{T}_{\mathcal{R}}(X)$ a category, the equations in 2 make each $f \in \Sigma$ a functor, and 3 forces the axioms E. The exchange law states that any rewrite of the form $r(\overline{\alpha})$—which represents the *simultaneous* rewriting of the term at the top using rule r *and* "below," i.e., in the subterms matched by the variables, using the rewrites $\overline{\alpha}$—is equivalent to the sequential composition $r(\overline{[w]}); t'(\overline{\alpha})$, corresponding to first rewriting on top with r and then below on the subterms matched by the variables with $\overline{\alpha}$, and is also equivalent to the sequential composition $t(\overline{\alpha}); r(\overline{[w']})$ corresponding to first rewriting below with $\overline{\alpha}$ and then on top with r. Therefore, the exchange law states that rewriting at the top by means of rule r and rewriting "below" using $\overline{\alpha}$ are processes that are independent of each other and can be done either simultaneously or in any order. Since $[t(x_1, \ldots, x_n)]$ and $[t'(x_1, \ldots, x_n)]$ can be regarded as functors $\mathcal{T}_{\mathcal{R}}(X)^n \longrightarrow \mathcal{T}_{\mathcal{R}}(X)$, from the mathematical point of view the exchange law just asserts that r is a *natural transformation*, i.e.,

LEMMA 10. [Meseguer, 1992] For each rewrite rule $r : [t(x_1, \ldots, x_n)] \longrightarrow [t'(x_1, \ldots, x_n)]$ in R, the family of morphisms

$$\{r(\overline{[w]}) : [t(\overline{w}/\overline{x})] \longrightarrow [t'(\overline{w}/\overline{x})] \mid \overline{[w]} \in T_{\Sigma,E}(X)^n\}$$

is a natural transformation $r : [t(x_1, \ldots, x_n)] \Rightarrow [t'(x_1, \ldots, x_n)]$ between the functors $[t(x_1, \ldots, x_n)], [t'(x_1, \ldots, x_n)] : \mathcal{T}_{\mathcal{R}}(X)^n \longrightarrow \mathcal{T}_{\mathcal{R}}(X)$.

The exchange law provides a way of *abstracting* a rewriting computation by considering immaterial the order in which rewrites are performed "above" and "below" in the term; further abstraction among proof terms is obtained from the functoriality equations. The equations 1-4 provide in a sense the *most abstract* "true concurrency" view of the computations of the rewrite theory $\mathcal{R}$ that can reasonably be given.

The category $\mathcal{T}_{\mathcal{R}}(X)$ is just one among many *models* that can be assigned to the rewrite theory $\mathcal{R}$. The general notion of model, called an $\mathcal{R}$*-system*, is defined as follows:

DEFINITION 11. Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, an $\mathcal{R}$*-system* $\mathcal{S}$ is a category $\mathcal{S}$ together with:

- a (Σ, E)-algebra structure given by a family of functors

  $$\{f_{\mathcal{S}} : \mathcal{S}^n \longrightarrow \mathcal{S} \mid f \in \Sigma_n, n \in \mathbb{N}\}$$

  satisfying the equations E, i.e., for any $t(x_1, \ldots, x_n) = t'(x_1, \ldots, x_n)$ in E we have an identity of functors $t_{\mathcal{S}} = t'_{\mathcal{S}}$, where the functor $t_{\mathcal{S}}$ is defined inductively from the functors $f_{\mathcal{S}}$ in the obvious way.

- for each rewrite rule $r : [t(\overline{x})] \longrightarrow [t'(\overline{x})]$ in R a natural transformation $r_{\mathcal{S}} : t_{\mathcal{S}} \Rightarrow t'_{\mathcal{S}}$.

An $\mathcal{R}$-*homomorphism* $F : \mathcal{S} \longrightarrow \mathcal{S}'$ between two $\mathcal{R}$-systems is then a functor $F : \mathcal{S} \longrightarrow \mathcal{S}'$ such that it is a Σ-algebra homomorphism—i.e., $f_{\mathcal{S}} * F = F^n * f_{\mathcal{S}'}$, for each f in Σ_n, $n \in \mathbb{N}$—and such that "F preserves R," i.e., for each rewrite rule $r : [t(\overline{x})] \longrightarrow [t'(\overline{x})]$ in R we have the identity of natural transformations[7] $r_{\mathcal{S}} * F = F^n * r_{\mathcal{S}'}$, where n is the number of variables appearing in the rule. This defines a category $\mathcal{R}$-**Sys** in the obvious way.

The above definition captures formally the idea that the models of a rewrite theory are *systems*. By a "system" we mean a machine-like entity that can be in a variety of *states*, and that can change its state by performing certain *transitions*. Such transitions are transitive, and it is natural and convenient to view states as "idle" transitions that do not change the state. In other words, a system can be naturally regarded as a *category*, whose objects are the states of the system and whose morphisms are the system's transitions.

For *sequential* systems such as labelled transition systems this is in a sense the end of the story; such systems exhibit *nondeterminism*, but do not have the required algebraic structure in their states and transitions to exhibit true concurrency. Indeed, what makes a system *concurrent* is precisely the existence of an additional *algebraic structure* [Meseguer, 1992]. First, the states themselves are distributed according to such a structure; for example, for Petri nets [Reisig, 1995] the distribution takes the form of a multiset. Second, concurrent transitions are themselves distributed according to the same algebraic structure; this is what the notion of $\mathcal{R}$-system captures, and is for example manifested in the concurrent firing of Petri nets, the evolution of concurrent object-oriented systems [Meseguer, 1993] and, more generally, in any type of concurrent rewriting.

The expressive power of rewrite theories to specify concurrent transition systems is greatly increased by the possibility of having not only transitions, but also *parameterized transitions*, i.e., *procedures*. This is what rewrite rules with variables provide. The family of states to which the procedure applies is given by those states where a component of the (distributed) state is a substitution instance of the lefthand side of the rule in question. The rewrite rule is then a *procedure* which transforms the state *locally*, by replacing such a substitution instance by the corresponding substitution instance of the righthand side. The fact that this can take place concurrently with other transitions "below" is precisely what the concept of a *natural transformation* formalizes. The following table summarizes our present discussion:

[7] Note that we use diagrammatic order for the *horizontal*, $\alpha * \beta$, and *vertical*, $\gamma ; \delta$, composition of natural transformations [Mac Lane, 1971].

System	$\longleftrightarrow$	*Category*
State	$\longleftrightarrow$	*Object*
Transition	$\longleftrightarrow$	*Morphism*
Procedure	$\longleftrightarrow$	*Natural Transformation*
Distributed Structure	$\longleftrightarrow$	*Algebraic Structure*

A detailed proof of the following theorem on the existence of initial and free $\mathcal{R}$-systems for the more general case of conditional rewrite theories is given in [Meseguer, 1992], where the soundness and completeness of rewriting logic for $\mathcal{R}$-system models is also proved.

THEOREM 12. $\mathcal{T}_{\mathcal{R}}$ is an initial object in the category $\mathcal{R}$-**Sys**. More generally, $\mathcal{T}_{\mathcal{R}}(X)$ has the following universal property: Given an $\mathcal{R}$-system $\mathcal{S}$, each function $F : X \longrightarrow |\mathcal{S}|$ extends uniquely to an $\mathcal{R}$-homomorphism $F^{\natural} : \mathcal{T}_{\mathcal{R}}(X) \longrightarrow \mathcal{S}$.

Preorder, poset, and algebra models

Since $\mathcal{R}$-systems are an "essentially algebraic" concept[8], we can consider classes Θ of $\mathcal{R}$-systems defined by the satisfaction of additional equations. Such classes give rise to full subcategory inclusions $\Theta \hookrightarrow \mathcal{R}$-**Sys**, and by general universal algebra results about essentially algebraic theories [Barr and Wells, 1985] such inclusions are *reflective* [Mac Lane, 1971], i.e., for each $\mathcal{R}$-system $\mathcal{S}$ there is an $\mathcal{R}$-system $R_\Theta(\mathcal{S}) \in \Theta$ and an $\mathcal{R}$-homomorphism $\rho_\Theta(\mathcal{S}) : \mathcal{S} \longrightarrow R_\Theta(\mathcal{S})$ such that for any $\mathcal{R}$-homomorphism $F : \mathcal{S} \longrightarrow \mathcal{D}$ with $\mathcal{D} \in \Theta$ there is a unique $\mathcal{R}$-homomorphism $F^{\diamondsuit} : R_\Theta(\mathcal{S}) \longrightarrow \mathcal{D}$ such that $F = \rho_\Theta(\mathcal{S}); F^{\diamondsuit}$. The assignment $\mathcal{S} \longmapsto R_\Theta(\mathcal{S})$ extends to a functor $\mathcal{R}$-**Sys** $\longrightarrow \Theta$, called the *reflection functor*.

Therefore, we can consider subcategories of $\mathcal{R}$-**Sys** that are defined by certain equations and be guaranteed that they have initial and free objects, that they are closed by subobjects and products, etc. Consider for example the following equations:

$$\forall f, g \in \mathit{Arrows},\ f = g \ \ \mathit{if} \ \ \partial_0(f) = \partial_0(g) \wedge \partial_1(f) = \partial_1(g)$$
$$\forall f, g \in \mathit{Arrows},\ f = g \ \ \mathit{if} \ \ \partial_0(f) = \partial_1(g) \wedge \partial_1(f) = \partial_0(g)$$
$$\forall f \in \mathit{Arrows},\ \partial_0(f) = \partial_1(f),$$

where $\partial_0(f)$ and $\partial_1(f)$ denote the source and target of an arrow f respectively. The first equation forces a category to be a preorder, the addition of the second requires this preorder to be a poset, and the three equations

[8] In the precise sense of being specifiable by an "essentially algebraic theory" or a "sketch" [Barr and Wells, 1985]; see [Meseguer, 1992] for more details.

together force the poset to be *discrete*, i.e., just a set. By imposing the first one, the first two, or all three, we get full subcategories

$$\mathcal{R}\text{-}\mathbf{Alg} \subseteq \mathcal{R}\text{-}\mathbf{Pos} \subseteq \mathcal{R}\text{-}\mathbf{Preord} \subseteq \mathcal{R}\text{-}\mathbf{Sys}.$$

A routine inspection of $\mathcal{R}$-**Preord** for $\mathcal{R} = (\Sigma, E, L, R)$ reveals that its objects are preordered Σ-algebras $(A, \leq)$—i.e., preordered sets with a Σ-algebra structure such that all the operations in Σ are monotonic—that satisfy the equations E and such that for each rewrite rule $r : [t(\overline{x})] \longrightarrow [t'(\overline{x})]$ in R and for each $\overline{a} \in A^n$ we have $t_A(\overline{a}) \leq t'_A(\overline{a})$. The poset case is entirely analogous, except that the relation $\leq$ is a partial order instead of being a preorder. Finally, $\mathcal{R}$-**Alg** is the category of ordinary Σ-algebras that satisfy the equations $E \cup eq(R)$, where $eq(r : [t] \longrightarrow [t']) = \{t_1 = t_2 \mid t_1 \in [t] \text{ and } t_2 \in [t']\}$, and $eq(R) = \bigcup\{eq(r : [t] \longrightarrow [t']) \mid [t] \longrightarrow [t'] \in R\}$.

The reflection functor associated with the inclusion $\mathcal{R}$-**Preord** $\subseteq$ $\mathcal{R}$-**Sys** sends $\mathcal{T}_{\mathcal{R}}(X)$ to the familiar $\mathcal{R}$-*rewriting relation*[9] $\rightarrow_{\mathcal{R}(X)}$ on E-equivalence classes of terms with variables in X. Similarly, the reflection associated to the inclusion $\mathcal{R}$-**Pos** $\subseteq$ $\mathcal{R}$-**Sys** maps $\mathcal{T}_{\mathcal{R}}(X)$ to the partial order $\leq_{\mathcal{R}(X)}$ obtained from the preorder $\rightarrow_{\mathcal{R}(X)}$ by identifying any two $[t], [t']$ such that $[t] \rightarrow_{\mathcal{R}(X)} [t']$ and $[t'] \rightarrow_{\mathcal{R}(X)} [t]$. Finally, the reflection functor into $\mathcal{R}$-**Alg** maps $\mathcal{T}_{\mathcal{R}}(X)$ to $T_{\mathcal{R}}(X)$, the free Σ-algebra on X satisfying the equations $E \cup eq(R)$; therefore, the classical *initial algebra semantics* of (functional) equational specifications reappears here associated with a very special class of models which—when viewed as systems—have only trivial identity transitions.

4 REWRITING LOGIC AS A LOGICAL FRAMEWORK

The adequacy of rewriting logic as a logical framework in which other logics can be represented by means of maps of logics or of entailment systems is explored by means of relevant examples, including equational, Horn, and linear logic, a general approach to the treatment of quantifiers, and a very general method for representing sequent presentations of a logic.

4.1 Mapping equational logic

As mentioned in Section 3.2, one can get equational logic from rewriting logic by adding the symmetry rule. Moreover, the syntax of rewriting logic includes equations in order to impose structural axioms on terms. Therefore, it should not be surprising to find out that there are many connections between both logics.

[9] It is perhaps more suggestive to call $\rightarrow_{\mathcal{R}(X)}$ the *reachability relation* of the system $\mathcal{T}_{\mathcal{R}}(X)$.

Even in the case of equational logic it can be convenient to allow sometimes a distinction between structural axioms and equations, so that an equational theory can then be described as a triple (Σ, E, Q), with Q a set of equations of the form $[u]_E = [v]_E$. This increases the expressiveness of equational theories, because we can allow more flexible description of equations—for example, omitting parentheses in the case when E contains an associativity axiom—and also supports a built-in treatment of the structural axioms in equational deduction. Indeed, this is fully consistent with the distinction made in OBJ3 and in Maude's functional modules between the equational *attributes* of an operator—such as associativity, commutativity, etc.—which are declared together with the operator, and the equations given, which are used modulo such attributes.

In order to define a map of entailment systems

$$(\Phi, \alpha) : ent(OSEqtl) \longrightarrow ent(OSRWLogic)$$

in principle we need to map an equation $[u]_E = [v]_E$ to a sequent, and the obvious choices are either $[u]_E \longrightarrow [v]_E$ or $[v]_E \longrightarrow [u]_E$. However this choice involves giving a fixed orientation to an equation, with the well-known problems that this causes. To avoid this choice, we would like to give the equation *both* orientations. We can achieve this by slightly generalizing Definition 5 of map of entailment systems in such a way that a sentence is mapped to a set of sentences[10]. In our case, α maps an equation $[u]_E = [v]_E$ to the set of sequents $\{[u]_E \longrightarrow [v]_E, [v]_E \longrightarrow [u]_E\}$, and Φ maps an equational theory $T = (\Sigma, E, Q)$ to the rewrite theory $\Phi(T) = (\Sigma, E, L, \alpha(Q))$, where $\alpha(Q) = \bigcup\{\alpha(e) \mid e \in Q\}$, and L is a labelling of the rewrite rules such that, for example, each rule is labelled by itself. This map satisfies

$$(\Sigma, E, Q) \vdash_{EL} e \iff (\Sigma, E, L, \alpha(Q)) \vdash_{RL} \alpha(e).$$

This can be easily proved by induction on the deduction rules of equational logic, using the fact that all the rules of rewriting logic are also rules of equational logic and the following lemma.

LEMMA 13.

$$(\Sigma, E, L, \alpha(Q)) \vdash_{RL} [u] \to [v] \iff (\Sigma, E, L, \alpha(Q)) \vdash_{RL} [v] \to [u].$$

Therefore, we have a *conservative* map of entailment systems.

In order to extend this map to a map of logics, a simple idea concerning models is to send a $\Phi(T)$-system $\mathcal{C}$ to $R_{\mathbf{Alg}}(\mathcal{C})$, where $R_{\mathbf{Alg}}$ is the reflection functor associated with the inclusion $\Phi(T)$-**Alg** $\subseteq$ $\Phi(T)$-**Sys**, as discussed

[10] This generalization is also very useful in relating other logics; see for example [Meseguer, 1998].

in Section 3.5. By definition, $R_{\mathbf{Alg}}(\mathcal{C})$ is a model of the equational theory T. However, this map does *not* satisfy the condition in Definition 6 of map of institutions. The difficulty is that, in general, from an equation $t = t'$ one can deduce that there is a chain $t \to t_1 \leftarrow t_2 \cdots t_n \leftarrow t'$, but not that $t \to t'$, as the reader familiar with term rewriting knows. To solve this problem, we consider a different quotient of the underlying (Σ, E)-algebra $|\mathcal{C}|$ in which two objects A and B are identified if and only if there exist morphisms $f : A \to B$ and $g : B \to A$ in $\mathcal{C}$. In this way, we obtain a (Σ, E)-algebra $\beta_T(\mathcal{C})$ that satisfies all the sentences in Q. Moreover, the condition in Definition 6 of map of institutions holds for this map. In short, we have obtained a conservative map of logics

$$(\Phi, \alpha, \beta) : \mathit{OSEqtl} \longrightarrow \mathit{OSRWLogic}.$$

There is also another map of logics

$$(\Phi', \alpha', \beta') : \mathit{OSEqtl} \longrightarrow \mathit{OSRWLogic}$$

that, instead of sending equations to sequents, sends equations to equations. This requires making explicit the fact, left implicit in Section 3, that equations can also be considered as sentences of rewriting logic, where, by definition,

$$(\Sigma, E, L, R) \vdash_{RL} t = t' \iff E \vdash_{EL} t = t'.$$

From this point of view, Φ' maps an equational theory (Σ, E) to the rewrite theory $(\Sigma, E, \emptyset, \emptyset)$, and at the level of sentences α' is just an inclusion, trivially satisfying the requirement for a map of entailment systems. Note that in this context the distinction between structural axioms and equations is not necessary.

With respect to the models, β'_T maps a $(\Sigma, E, \emptyset, \emptyset)$-system $\mathcal{C}$ to the underlying (Σ, E)-algebra structure on $|\mathcal{C}|$, trivially satisfying also the condition in Definition 6 and being therefore a map of institutions. Notice that (Φ', α', β') is *conservative* in a straightforward way.

On the opposite direction there is also a map of logics

$$(\Psi, \gamma, \delta) : \mathit{OSRWLogic} \longrightarrow \mathit{OSEqtl}$$

mapping a rewrite theory (Σ, E, L, R) to the equational theory $(\Sigma, E, \gamma(R))$ where γ removes the labels from the rules and turns the sequent signs "$\longrightarrow$" into equality signs. For the models, $\delta_{\mathcal{R}}$ is the inclusion $\mathcal{R}$-**Alg** $\subseteq$ $\mathcal{R}$-**Sys** defined in Section 3.5.

Notice that the composition of maps of logics $(\Phi, \alpha, \beta); (\Psi, \gamma, \delta)$ is the identity.

4.2 Mapping Horn logic

Horn logic signatures are of the form (F, P), with F a set of function symbols and P a set of predicate symbols. In the order-sorted case such symbols have ranks $f : s_1 \dots s_n \to s$, and $p : s_1 \dots s_n$, specified by strings of sorts in the poset of sorts S. Models are F-algebras M together with, for each predicate symbol $p : s_1 \dots s_n$, a subset $p_M \subseteq M_{s_1} \times \dots \times M_{s_n}$, which can alternatively be viewed as a characteristic function $p_M : M_{s_1} \times \dots \times M_{s_n} \longrightarrow \mathit{Bool}$ to the two element Boolean algebra *Bool*. Satisfaction of a Horn clause

$$q_1(\overline{u_1}), \dots, q_n(\overline{u_n}) \;\Rightarrow\; p(\overline{t})$$

in a model M can be expressed as either the subset containment of the intersection of the interpretations of $q_1(\overline{u_1}), \dots, q_n(\overline{u_n})$ in M inside the corresponding interpretation of $p(\overline{t})$, or, in a characteristic function description, as the functional inequality

$$q_1(\overline{u_1})_M \;\; \mathit{and} \;\; \dots \;\; \mathit{and} \;\; q_n(\overline{u_n})_M \leq p(\overline{t})_M$$

between the corresponding interpretations in M of the conjunction of the premises and of the conclusion as characteristic functions, where the inequality between the functions means inequality of their values for each of the arguments in the Boolean algebra ordering. A *homomorphism* $f : M \to M'$ between two such models is an F-homomorphism which in addition satisfies $(f_{s_1} \times \dots \times f_{s_n})(p_M) \subseteq p_{M'}$ for each $p : s_1 \dots s_n$, or in characteristic function form the functional inequality

$$p_M \leq (f_{s_1} \times \dots \times f_{s_n}); p_{M'}.$$

Horn logic is a particularly simple logic that does not use the full power of classical first-order logic and is in fact compatible with a variety of other nonclassical interpretations such as for example intuitionistic logic. It is therefore reasonable to enlarge the class of models just described by keeping the F-algebra parts as before, but allowing instead interpretations of the predicate symbols p as "characteristic functions"

$$p_M : M_{s_1} \times \dots \times M_{s_n} \longrightarrow M_{Prop}$$

into a partially ordered set M_{Prop} of "propositions" which is not required to be fixed, i.e., it can vary from model to model. We require of any such poset the "bare minimum" structure of having a top element *true*: *Prop* and a binary associative and commutative "conjunction" operator $\_,\_$: *Prop Prop* $\longrightarrow$ *Prop* that is monotonic and has *true* as its neutral element. Of course, *Bool* is one such poset, where conjunction is interpreted as *and*. Satisfaction of Horn clauses can be defined by a functional inequality just as before, but changing *Bool* by the appropriate poset M_{Prop} being chosen for the model.

The natural generalization of the notion of homomorphism $f : M \longrightarrow M'$ is to again require an F-homomorphism for the operations in F, whereas for predicate symbols $p : s_1 \dots s_n$ we require the functional inequality

$$(\dagger) \qquad p_M ; f_{Prop} \leq (f_{s_1} \times \dots \times f_{s_n}) ; p_{M'}$$

where $f_{Prop} : M_{Prop} \longrightarrow M'_{Prop}$ is an additional component of the homomorphism, namely, a monotonic function preserving *true* and conjunction "up to inequality" between the posets of propositions M_{Prop} and M'_{Prop} chosen for the models M and M', in the sense that we have $f_{Prop}(true_M) \leq true_{M'}$, and $f_{Prop}(x, y) \leq f_{Prop}(x), f_{Prop}(y)$, for $x, y \in M_{Prop}$. This defines a category of models (F, P)-**Mod**.

In addition, we can consider the generalization to Horn theories of the form (F, P, E, H) where E is a set of F-equations, and H is a set of Horn clauses involving the predicates in P but not equations (again, equations in E can be viewed as structural axioms forming part of the signature). A model satisfies this theory when the underlying F-algebra satisfies all the equations in E and the model satisfies the Horn clauses in H, defining in this way a full subcategory (F, P, E, H)-**Mod** of (F, P)-**Mod**. We denote by $OSHorn^{=}$ the logic whose theories are such generalized Horn theories (F, P, E, H) with equational axioms E, and whose models we have just described.

The map of logics

$$(\Phi, \alpha, \beta) : OSHorn^{=} \longrightarrow OSRWLogic$$

that we define now is a considerable simplification and extension of the map described in [Meseguer, 1992b].

A Horn theory (F, P, E, H) is mapped to a rewrite theory

$$\Phi(F, P, E, H) = (F \cup P^{\diamond}, E \cup ACI, \{*\} \cup H, \{x_{Prop} \longrightarrow true\} \cup H^{\diamond}),$$

where

- $F \cup P^{\diamond}$ is the order-sorted signature that extends F by adding the additional sort *Prop*, a constant $true : Prop$, a binary operator $\_,\_$ on *Prop*, and, for each predicate symbol $p : s_1 \dots s_n$ in P, an operator $p : s_1 \dots s_n \longrightarrow Prop$;
- ACI is the set of associativity, commutativity, and identity (*true*) structural axioms for the conjunction operator $\_,\_$;
- "$*$" is the label for the rewrite rule $x_{Prop} \longrightarrow true$, where x_{Prop} is a variable of sort *Prop*;
- $H^{\diamond}$ is a set of rewrite rules labelled by the Horn clauses H themselves in such a way that a Horn clause of the form $q_1(\overline{u_1}), \dots, q_n(\overline{u_n}) \Rightarrow p(\overline{t})$ labels the rewrite rule $q_1(\overline{u_1}), \dots, q_n(\overline{u_n}) \longrightarrow p(\overline{t})$, whereas a Horn clause of the form $p(\overline{t})$ labels the rewrite rule $true \longrightarrow p(\overline{t})$.

At the level of sentences, α maps each Horn clause to its corresponding labelled rewrite rule in the above manner.

As to models, given a Horn theory T, a $\Phi(T)$-system consists of a category $\mathcal{C}_s$ for each sort s in the poset S, and a category $\mathcal{P}$ for the sort *Prop*, together with a collection of functors satisfying the equations in $\Phi(T)$ and natural transformations interpreting the rewrite rules in $\Phi(T)$. The functor β_T sends such a system to the T-model consisting of the underlying (order-sorted) algebra structure on the family of sets $\{|\mathcal{C}_s| \mid s \in S\}$, and the poset $R_{\mathbf{Pos}}(\mathcal{P})$, where $R_{\mathbf{Pos}}$ is the reflection functor associated to the inclusion $\Phi(T)$-**Pos** $\subseteq$ $\Phi(T)$-**Sys**, discussed in Section 3.5. By definition of this reflection functor, $A \leq B$ in $R_{\mathbf{Pos}}(\mathcal{P})$ if and only if there is a morphism $A \to B$ in $\mathcal{P}$. Therefore, a Horn clause $q_1(\overline{u_1}), \ldots, q_n(\overline{u_n}) \Rightarrow p(\overline{t})$ is satisfied by this T-model if and only if there is a morphism in $\mathcal{P}$ interpreting the rewrite sequent $q_1(\overline{u_1}), \ldots, q_n(\overline{u_n}) \longrightarrow p(\overline{t})$ if and only if this sequent is satisfied by the original $\Phi(T)$-system. Thus, (Φ, α, β) is indeed a map of institutions.

Notice that, by the conditions for $\mathcal{R}$-homomorphisms in Definition 11, for the homomorphisms in the image of β_T the functional inequality (†) above becomes an equality. In addition, β_T maps free $\Phi(T)$-systems to (weakly) free Horn T-models; since the entailment relation coincides with satisfaction in free models (see the proof of Theorem 3.13 in [Meseguer, 1992]), this provides a short proof of the fact that (Φ, α) is indeed a map of entailment systems, and moreover, it is *conservative*.

The same discussion applies to the case of preorders instead of posets, by considering the reflection functor associated to the inclusion $\Phi(T)$-**Preord** $\subseteq$ $\Phi(T)$-**Sys**, which would have given a slightly more general notion of model for a Horn theory in which propositions would form a preorder.

4.3 Mapping linear logic

In this section, we describe a map of logics *LinLogic* $\longrightarrow$ *OSRWLogic* mapping theories in full quantifier-free first-order linear logic to rewrite theories. We do not provide much motivation for linear logic, referring the reader to [Girard, 1987; Troelstra, 1992; Martí-Oliet and Meseguer, 1991] for example. We need to point out, nonetheless, the way linear logic satisfies the conditions given in Definition 1 of entailment system. If one thinks of formulas as sentences and of the turnstile symbol "$\vdash$" in a sequent as the entailment relation, then this relation is not monotonic, because in linear logic the structural rules of weakening and contraction are forbidden, so that, for example, we have the sequent $A \vdash A$ as an axiom, but we cannot derive either $A, B \vdash A$ or even $A, A \vdash A$. The point is that, for Σ a linear logic signature, the elements of $sen(\Sigma)$ should not be identified with *formulas* but with *sequents*. Viewed as a way of generating sequents, i.e., identifying our entailment relation $\vdash$ with the closure

of the horizontal bar relation among linear logic sequents, the entailment of linear logic is indeed reflexive, monotonic, and transitive. This idea is also supported by the categorical models for linear logic [Seely, 1989; Martí-Oliet and Meseguer, 1991], in which sequents are interpreted as morphisms, and leads to a very natural correspondence between the models of rewriting and linear logic.

Expressing linear logic in rewriting logic

We use the syntax of the Maude language to write down the map of entailment systems from linear logic to rewriting logic. Note that any sequence of characters starting with either "`---`" or "`***`" and ending with "`end-of-line`" is a comment. Moreover, we usually drop the equivalence class square brackets, adopting the convention that a term t denotes the equivalence class $[t]_E$ for the appropriate set of structural axioms E.

We first define the *functional* theory `PROP0[X]` which introduces the syntax of propositions as a parameterized abstract data type. The parameterization permits having additional structure at the level of atoms if desired. In order to provide a proper treatment of negation, only equations are given, and no rewrite rules are introduced in this theory; they are introduced afterwards in the `LINLOG[X]` theory. The purpose of the equations in the `PROP0[X]` theory is to push negation to the atom level, by using the dualities of linear logic; this is a well-known process in classical and linear logic.

```
fth ATOM is
  sort Atom .
endft

--- linear logic syntax
fth PROP0[X :: ATOM] is
  sort Prop0 .
  subsort Atom < Prop0 .
  ops 1 0 ⊥ ⊤ : -> Prop0 .
  op _⊥ : Prop0 -> Prop0 .
  op _⊗_ : Prop0 Prop0 -> Prop0 [assoc comm id: 1] .
  op _⅋_ : Prop0 Prop0 -> Prop0 [assoc comm id: ⊥] .
  op _⊕_ : Prop0 Prop0 -> Prop0 [assoc comm id: 0] .
  op _&_ : Prop0 Prop0 -> Prop0 [assoc comm id: ⊤] .
  op !_ : Prop0 -> Prop0 .
  op ?_ : Prop0 -> Prop0 .

  vars A B : Prop0 .
  eq (A ⊗ B)⊥ = A⊥ ⅋ B⊥ .
  eq (A ⅋ B)⊥ = A⊥ ⊗ B⊥ .
  eq (A & B)⊥ = A⊥ ⊕ B⊥ .
```

```
  eq (A ⊕ B)^⊥ = A^⊥ & B^⊥ .
  eq (!A)^⊥ = ?(A^⊥) .
  eq (?A)^⊥ = !(A^⊥) .
  eq A^⊥^⊥ = A .
  eq 1^⊥ = ⊥ .
  eq ⊥^⊥ = 1 .
  eq ⊤^⊥ = 0 .
  eq 0^⊥ = ⊤ .
endft
```

Note that the equations can be used as oriented rules from left to right at the implementation level in order to obtain a canonical form for expressions in `Prop0`.

The `LINLOG[X]` theory introduces linear logic propositions and the rules of the logic. Propositions are of the form `[A]` for `A` an expression in `Prop0`. All logical connectives work similarly for `Prop0` expressions and for propositions, except negation, which is defined only for `Prop0` expressions.

Some presentations of linear logic are given in the form of one-sided sequents $\vdash \Gamma$ where negation has been pushed to the atom level, and there are no rules for negation in the sequent calculus [Girard, 1987]. In this section, in order to make the connections with category theory and with rewriting logic more direct, we prefer to use standard sequents of the more general form $\Gamma \vdash \Delta$. In a later section, we will also use one-sided sequents just in order to reduce the number of rules.

The style of our formulation adopts a categorical viewpoint for the proof theory and semantics of linear logic [Seely, 1989; Martí-Oliet and Meseguer, 1991]. This style exploits the close connection between the models of linear logic and those of rewriting logic which are also categories, as we have explained in Section 3.5. Without going into details that the reader can find for example in [Martí-Oliet and Meseguer, 1991] and the references therein, the tensor and linear implication connectives are interpreted in a closed symmetric monoidal category $\langle \mathcal{C}, \otimes, \multimap \rangle$. Negation is interpreted by means of a dualizing object $\bot$ and the definition $A^\bot = A \multimap \bot$ (with this definition of negation, $\mathcal{C}$ becomes a $*$-autonomous category [Barr, 1979]). The categorical product & interprets additive conjunction. The interpretation of the exponential ! is given by a comonad $\langle !A, !A \to A, !A \to !!A \rangle$ that maps the comonoid structure $\top \leftarrow A \to A \& A$ into a comonoid structure $1 \leftarrow !A \to !A \otimes !A$ via isomorphisms $!\top \cong 1$ and $!(A \& A) \cong !A \otimes !A$.

The dual connectives $\parr$, $\oplus$, and ? can be defined using negation: $A \parr B = (A^\bot \otimes B^\bot)^\bot = A^\bot \multimap B$, $A \oplus B = (A^\bot \& B^\bot)^\bot$, $?A = (!A^\bot)^\bot$. Without negation, $\oplus$ needs the presence of coproducts and ? is interpreted by means of a monad with a monoid structure.

When seeking the minimal categorical structure required for interpreting linear logic, an important question is how to interpret the connective $\parr$ without using negation, and how to axiomatize its relationship with the

tensor $\otimes$. Cockett and Seely have answered this question with the notion of a *weakly distributive category* [Cockett and Seely, 1992]. A weakly distributive category consists of a category $\mathcal{C}$ with two symmetric tensor products $\otimes, \parr : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, and a natural transformation $A \otimes (B \parr C) \longrightarrow (A \otimes B) \parr B$ (weak distributivity) satisfying some coherence equations[11]. Negation is added to a weakly distributive category by means of a function $(\_)^{\perp} : |\mathcal{C}| \to |\mathcal{C}|$ on the objects of $\mathcal{C}$, and natural transformations $1 \longrightarrow A \parr A^{\perp}$ and $A \otimes A^{\perp} \longrightarrow \perp$ satisfying some coherence equations. Cockett and Seely then prove that the concepts of weakly distributive category with negation and of $*$-autonomous category are equivalent, providing in this way a categorical semantics for linear logic in which the *par* connective $\parr$ is primitive and is not defined in terms of tensor and negation.

In the following theory, the rewrite rules for $\otimes$, $\parr$, and negation correspond to the natural transformations in the definition of a weakly distributive category, as explained above. The rules for & ($\oplus$, respectively) mirror the usual definition of final object and product (initial object and coproduct, respectively). Finally, the axioms and rules for the exponential ! (?, respectively) correspond to the comonad with a comonoid structure (monad with monoid structure, respectively). Note that some rules are redundant, but we have decided to include them in order to make the connectives less interdependent, so that, for example, if the connective & is omitted we do not need to add new rules for the modality !.

```
--- linear logic rules
th LINLOG[X :: ATOM] is
  protecting PROP0[X] .
  sort Prop .
  ops 1 0 ⊥ ⊤ : -> Prop .
  op _⊗_ : Prop Prop -> Prop [assoc comm id: 1] .
  op _⅋_ : Prop Prop -> Prop [assoc comm id: ⊥] .
  op _⊕_ : Prop Prop -> Prop [assoc comm id: 0] .
  op _&_ : Prop Prop -> Prop [assoc comm id: ⊤] .
  op !_ : Prop -> Prop .
  op ?_ : Prop -> Prop .

  op [_] : Prop0 -> Prop .

  vars A B : Prop0 .
  ax [A ⊗ B] = [A] ⊗ [B] .
  ax [A ⅋ B] = [A] ⅋ [B] .
  ax [A & B] = [A] & [B] .
  ax [A ⊕ B] = [A] ⊕ [B] .
  ax [!A] = ![A] .
```

[11] Cockett and Seely develop in [1992] the more general case in which the tensor products are not assumed to be symmetric.

```
  ax [?A] = ?[A] .
  ax [1] = 1 .
  ax [⊥] = ⊥ .
  ax [⊤] = ⊤ .
  ax [0] = 0 .

  *** [_] is injective
  cax A = B if [A] = [B] .

  *** Rules for negation
  rl 1 => [A] ⅋ [A^⊥] .
  rl [A] ⊗ [A^⊥] => ⊥ .

  vars P Q R : Prop .
  *** Rules for ⊗ and ⅋
  rl P ⊗ (Q ⅋ R) => (P ⊗ Q) ⅋ R .

  *** Rules for &
  rl P => ⊤ .                                  *** (1)
  rl P & Q => P .
  crl R => P & Q  if  R => P  and  R => Q .  *** (2)

  *** Rules for ⊕
  rl 0 => P .                                   *** (3)
  rl P => P ⊕ Q .
  crl P ⊕ Q => R  if  P => R  and  Q => R .  *** (4)

  *** Structural axioms and rules for !
  ax !(P & Q) = !P ⊗ !Q .                       *** (5)
  ax !⊤ = 1 .                                  *** (6)
  rl !P => P .
  rl !P => !!P .
  rl !P => 1 .        *** redundant from (1) and (6) above
  rl !P => !P ⊗ !P .  *** redundant from (2) and (5) above

  *** Structural axioms and rules for ?
  ax ?(P ⊕ Q) = ?P ⅋ ?Q .                       *** (7)
  ax ?0 = ⊥ .                                  *** (8)
  rl P => ?P .
  rl ??P => ?P .
  rl ⊥ => ?P .        *** redundant from (3) and (8) above
  rl ?P ⅋ ?P => ?P .  *** redundant from (4) and (7) above
endt
```

A linear logic formula is built from a set of propositional constants using the logical constants and connectives of linear logic. Notice that linear implication $A \multimap B$ is not necessary because it can be defined as $A^{\perp} \parr B$.

Representing a linear logic theory in rewriting logic

A *linear theory* T in propositional linear logic consists of a finite set C of propositional constants and a finite set S of double-sided sequents of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$, where each A_i and B_j is a linear logic formula built from the constants in C. Given such a theory T, it is interpreted in rewriting logic as follows.

First, we define a functional theory to interpret the propositional constants in C. For example, if $C = \{a, b, c\}$ we would define

```
fth C is
  sort Atom .
  ops a b c : -> Atom .
endft
```

Then, we can instantiate the parameterized theory `LINLOG[X]` using this functional theory, with the default view `ATOM` $\rightarrow$ `C`:

```
make LINLOG0 is LINLOG[C] endmk
```

A linear logic formula A (with constants in C) is interpreted in `LINLOG0` as the term `[A]` of sort `Prop`. For example, the formula $(a \otimes b)^{\perp} \oplus (!(a \,\&\, c^{\perp}))^{\perp}$ is interpreted as the term

```
[(a ⊗ b)^⊥ ⊕ (!(a & c^⊥))^⊥]
```

which, using the equations for negation in `PROPO[X]` and the structural axioms in `LINLOG[X]`, is equal to the term

```
([a^⊥] ⅋ [b^⊥]) ⊕ ?([a^⊥] ⊕ [c]).
```

Finally, we extend the theory `LINLOG0` by adding a rule

```
rl [A1] ⊗ ... ⊗ [An] => [B1] ⅋ ... ⅋ [Bm] .
```

for each sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ in the linear theory T. For example, if

$$\begin{aligned} T \;=\; \{ & a \otimes b, !c \oplus a \vdash a, (c \oplus b)^{\perp} \\ & a \parr b, ?(c^{\perp}) \vdash (?b \parr !c)^{\perp}, a \oplus b \}, \end{aligned}$$

the corresponding rewrite theory is

```
th LINLOG(T) is
  including LINLOG0 .
  rl [a] ⊗ [b] ⊗ (![c] ⊕ [a]) => [a] ⅋ ([c^⊥] & [b^⊥]) .
  rl ([a] ⅋ [b]) ⊗ ?[c^⊥] => (![b^⊥] ⊗ ?[c^⊥]) ⅋ ([a] ⊕ [b]) .
endt
```

Note that this technique can also be used to interpret quantifier-free first-order linear logic formulas, where, instead of propositional constants, we have literals built using functions and predicates. In general, we can allow any abstract data type `ADT` defining constants, functions and predicates. Then, we define the instantiation

```
make LINLOG0 is LINLOG[ADT] endmk
```

which is finally extended with the corresponding rules to a theory `LINLOG(T)` corresponding to the desired theory T.

The main result is the following conservativity theorem.

THEOREM 14. Given a linear theory T, a sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ is provable in linear logic from the axioms in T if and only if the sequent

`[A1]` $\otimes$ ... $\otimes$ `[An]` $\longrightarrow$ `[B1]` $\parr$ ... $\parr$ `[Bm]`

is a `LINLOG(T)`-rewrite, i.e., it is provable in rewriting logic from the rewrite theory `LINLOG(T)`.

To show that a linear logic proof can be translated into a rewriting logic proof, the idea is similar to the proof of the soundness theorem for the categorical semantics of linear logic, where a sequent is interpreted as a morphism (see [Martí-Oliet and Meseguer, 1991, Theorem 40]). What is important to realize is that the categorical constructions of these morphisms can be seen as rewriting logic proofs; for example, functoriality corresponds to the *Congruence* rule of rewriting logic, something made completely explicit in the categorical semantics of rewriting logic, as outlined in Section 3.5 and developed in detail in [Meseguer, 1992].

The map of logics

The fully detailed development in the previous sections provides a map of entailment systems between linear logic and rewriting logic, which is conservative because of Theorem 14. We have already discussed briefly the models of linear logic in Section 4.3 by way of motivation to the rules in the theory `LINLOG[X]`. Now, in order to complete the construction of the map of logics *LinLogic* $\longrightarrow$ *OSRWLogic*, we need a way of getting a (categorical) model of a linear theory T from a rewrite system that is a model of the rewrite theory `LINLOG(T)`.

The first thing to note, recalling the definition of $\mathcal{R}$-system in Section 3.5, is that for each rewrite rule in R we require just a natural transformation in the system, but we do not impose any coherence or uniqueness conditions on these natural transformations. For this reason, a `LINLOG(T)`-system interprets $A\&B$ as a weak product instead of a product, for example. A way of obtaining uniqueness would be considering the generalized rewrite

theories defined in [Meseguer, 1992b], but we do not need that for our purposes here. On the other hand, the attributes of the operations, like associativity or commutativity, are interpreted as identities, instead of the more general natural isomorphisms, thus satisfying all coherence conditions automatically.

In general, given a linear theory $T = (C, S)$, a `LINLOG(T)`-system consists of an algebra $\mathcal{A}$ interpreting all the structure of the functional theory `PROPO[C]`, a category $\mathcal{C}$ with all the morphisms necessary to interpret the rewrite rules in the theory `LINLOG[C]` and the rules corresponding to all the sequents in S, and an injective homomorphism $\mathcal{A} \to |\mathcal{C}|$ that, without loss of generality, we can consider to be an inclusion. Note that, as $\mathcal{A}$ is closed under all the operations in the theory `LINLOG[C]`, the full subcategory of $\mathcal{C}$ generated by $\mathcal{A}$ has the same structure as $\mathcal{C}$, and, in addition, there is a function $(\_)^{\perp} : \mathcal{A} \to \mathcal{A}$ interpreting negation. Therefore, this full subcategory is *almost* a weakly distributive category with negation, products, coproducts, a comonad with a comonoid structure, and a monad with a monoid structure. What is possibly missing is the satisfaction of a set of equations between morphisms which ensure that all this structure is really what we want.

Thus, in order to get a Girard category $\mathcal{L}$ from the original `LINLOG(T)`-system, we do the quotient of the full subcategory of $\mathcal{C}$ generated by $\mathcal{A}$ by this set of equations. Clearly, there is a morphism $A \to B$ in $\mathcal{L}$ if and only if there is a morphism $A \to B$ in $\mathcal{C}$, i.e., $\mathcal{L}$ satisfies a linear sequent if and only if $\mathcal{C}$ satisfies the rewriting logic version of that sequent. This is true because the constants in C are interpreted always as the corresponding constants in $\mathcal{A}$, and variables in a sequent are also interpreted as elements of $\mathcal{A}$ (note that variables appear in a theory `ADT` that is used to instantiate `PROPO[X]`). In summary, we have a *conservative* map of logics *LinLogic* $\longrightarrow$ *OSRWLogic*.

4.4 Quantifiers

In Section 4.3 we have defined a map of logics between quantifier-free linear logic and rewriting logic. In this section, we show how to extend that map at the level of entailment systems to quantifiers. The choice of linear logic to illustrate the treatment of quantifiers is irrelevant; we could have chosen any other logic. It has only the expository advantage of building upon an example already introduced in this paper. In fact, our equational treatment of quantification, inspired by ideas of Laneve and Montanari on the definition of the lambda calculus as a theory in rewriting logic [Laneve and Montanari, 1992; Laneve and Montanari, 1996], is very general and encompasses not only existential and universal quantification, but also lambda abstraction and other such binding mechanisms.

The main idea is to internalize as operations in the theory the notions

of free variables and substitution that are usually defined at the metalevel. Then, the typical definitions of such notions by structural induction on terms can be easily written down as equations in the theory, but, more importantly, we can consider terms modulo these axioms and we can also use the operation of substitution explicitly in the rules introducing or eliminating quantifiers. This is similar to the lambda calculus with explicit substitutions defined by Abadi, Cardelli, Curien, and Lévy in [1991], and to the work on binding structures by Talcott [1993].

We begin by presenting the example of the lambda abstraction binding mechanism in the lambda calculus, as defined by Laneve and Montanari in [1992] (see also [Laneve and Montanari, 1996], where this technique is generalized to combinatory reduction systems). Since in this case the syntax is much simpler, the main ideas can become more explicit and clearer to the reader.

We assume a parameterized functional module `SET[X]` that provides finite sets over a parameter set `X` with operations `_U_` for union, `_-_` for set difference, {_} for singleton, `emptyset` for the empty set, and a predicate `_is-in_` for membership.

```
--- variable names
fth VAR is
  sort Var .
  protecting SET[Var] .
  op new : Set -> Var .
  var S : Set .
  eq new(S) is-in S = false .  *** new variable
endft

--- lambda calculus syntax with substitution
fmod LAMBDA[X :: VAR] is
  including SET[X] .
  sort Lambda .
  subsort Var < Lambda .                      *** variables
  op λ_._ : Var Lambda -> Lambda .            *** lambda abstraction
  op __ : Lambda Lambda -> Lambda .           *** application
  op _[_/_] : Lambda Lambda Var -> Lambda .   *** substitution
  op fv : Lambda -> Set .                     *** free variables

  vars X Y : Var .
  vars M N P : Lambda .

  *** Free variables
  eq fv(X) = {X} .
  eq fv(λX.M) = fv(M) - {X} .
  eq fv(MN) = fv(M) U fv(N) .
  eq fv(M[N/X]) = (fv(M) - {X}) U fv(N) .
```

```
  *** Substitution equations
  eq X[N/X] = N .
  ceq Y[N/X] = Y  if  not(X == Y) .
  eq (MN)[P/X] = (M[P/X])(N[P/X]) .
  eq (λX.M)[N/X] = λX.M .
  ceq (λY.M)[N/X] = λY.(M[N/X])
      if not(X == Y) and (not(Y is-in fv(N)) or not(X is-in fv(M))) .
  ceq (λY.M)[N/X] = λ(new(fv(MN))).((M[new(fv(MN))/Y])[N/X])
      if not(X == Y) and Y is-in fv(N) and X is-in fv(M) .
endfm
```

Note that substitution is here another term constructor instead of a meta-syntactic operation. Of course, using the above equations, all occurrences of the substitution constructor can be eliminated. After having defined in the previous functional module the class of lambda terms with substitution, we just need to add the equational axiom of alpha conversion and the beta rule in the following module:

```
--- lambda calculus rules
mod ALPHA-BETA[X :: VAR] is
  including LAMBDA[X] .
  vars X Y : Var .
  vars M N : Lambda .

  *** Alpha conversion
  cax λX.M = λY.(M[Y/X])  if  not(Y is-in fv(M)) .

  *** Beta reduction
  rl (λX.M)N => M[N/X] .
endm
```

In order to introduce quantifiers, we can develop a similar approach, by first introducing substitution in the syntax together with the quantifiers, and then adding rewrite rules for the new connectives. In the same way that we had to duplicate the logical connectives in both theories `PROPO[X]` and `LINLOG[X]` in Section 4.3 in order to have a correct treatment of negation, we also have to duplicate the operations and equations for substitution in the two modules `FO-PROPO[X]` and `FO-LINLOG[X]` below. This technicality, due to the treatment of negation, makes the exposition somewhat longer, but should not obscure the main ideas about the treatment of quantification that have been illustrated more concisely before with the lambda calculus example.

We assume an abstract data type `ADT` defining constants, functions and predicates over a set `Var` of variable names. Substitution must also be defined in this module. For example, we can have something like the following module:

```
fmod ADT[X :: VAR] is
  including SET[X] .
  sort Term .                    *** terms
  subsort Var < Term .           *** variables are terms
  op c : -> Term .               *** constant symbol
  op f : Term Term -> Term .     *** function symbol
  sort Atom .                    *** atomic formulas
  op p : Term Term -> Atom .     *** predicate symbol

  op va : Term -> Set .          *** set of variables
  op va : Atom -> Set .          *** set of variables
  op _[_/_] : Term Term Var -> Term .  *** substitution
  op _[_/_] : Atom Term Var -> Atom .  *** substitution

  vars X Y : Var .    vars T U V : Term .
  var P : Atom .

  *** Set of variables
  eq va(X) = {X} .
  eq va(c) = emptyset .
  eq va(f(T,V)) = va(T) U va(V) .
  eq va(p(T,V)) = va(T) U va(V) .
  eq va(V[T/X]) = (va(V) - {X}) U va(T) .
  eq va(P[T/X]) = (va(P) - {X}) U va(T) .

  *** Substitution equations
  eq X[T/X] = T .
  ceq Y[T/X] = Y  if  not(X == Y) .
  eq c[T/X] = c .
  eq f(U,V)[T/X] = f(U[T/X],V[T/X]) .
  eq p(U,V)[T/X] = p(U[T/X],V[T/X]) .
endfm

--- linear logic syntax with quantifiers
fmod FO-PROP0[X :: VAR] is
  including PROP0[ADT[X]] .
  op _[_/_] : Prop0 Term Var -> Prop0 .  *** substitution
  op fv : Prop0 -> Set .                 *** free variables
  op ∀_._ : Var Prop0 -> Prop0 .         *** universal quantifier
  op ∃_._ : Var Prop0 -> Prop0 .         *** existential quantifier

  vars A B : Prop0 .    vars X Y : Var .
  var P : Atom .        var T : Term .

  *** Negation and quantifiers
  eq (∀X.A)^⊥ = ∃X.A^⊥ .
  eq (∃X.A)^⊥ = ∀X.A^⊥ .
```

```
  *** Free variables
  eq fv(P) = va(P) .
  eq fv(1) = emptyset .
  eq ...  *** similar equations for the other logical constants
  eq fv(A^⊥) = fv(A) .
  eq fv(A ⊗ B) = fv(A) U fv(B) .
  eq ...  *** similar equations for the other logical connectives
  eq fv(∀X.A) = fv(A) - {X} .
  eq fv(∃X.A) = fv(A) - {X} .
  eq fv(A[T/X]) = (fv(A) - {X}) U va(T) .

  *** Substitution equations
  eq 1[T/X] = 1 .
  eq ...  *** similar equations for the other logical constants
  eq A^⊥[T/X] = A[T/X]^⊥ .
  eq (A ⊗ B)[T/X] = A[T/X] ⊗ B[T/X] .
  eq ...  *** similar equations for the other logical connectives
  eq (∀X.A)[T/X] = ∀X.A .
  ceq (∀Y.A)[T/X] = ∀Y.(A[T/X])
      if not(X == Y) and (not(Y is-in va(T)) or not(X is-in fv(A))) .
  ceq (∀Y.A)[T/X] =
           ∀(new(va(T) U fv(A))).((A[new(va(T) U fv(A))/Y])[T/X])
      if not(X == Y) and Y is-in fv(T) and X is-in fv(A) .
  eq ...  *** similar equations for the existential quantifier
endfm

mod FO-LINLOG[X :: VAR] is
  including LINLOG[ADT[X]] .  ***
  protecting FO-PROP0[X] .    *** Note PROP0[ADT[X]] is shared

  op _[_/_] : Prop Term Var -> Prop0 .  *** substitution
  op fv : Prop -> Set .                 *** free variables
  op ∀_._ : Var Prop -> Prop .          *** universal quantifier
  op ∃_._ : Var Prop -> Prop .          *** existential quantifier

  var P Q : Prop .    var A : Prop0 .
  var X : Var .       var T : Term .

  ax [∀X.A] = ∀X.[A] .
  ax [∃X.A] = ∃X.[A] .

  *** Free variables
  ax fv(1) = emptyset .
  ax ...  *** similar axioms for the other logical constants
  ax fv(P ⊗ Q) = fv(P) U fv(Q) .
  ax ...  *** similar axioms for the other logical connectives
```

```
  ax fv(∀X.P) = fv(P) - {X} .
  ax fv(∃X.P) = fv(P) - {X} .
  ax fv(P[T/X]) = (fv(P) - {X}) U va(T) .
  ax fv([A]) = fv(A) .

  *** Substitution axioms
  ax 1[T/X] = 1 .
  ax ...  *** similar axioms for the other logical constants
  ax (P ⊗ Q)[T/X] = P[T/X] ⊗ Q[T/X] .
  ax ...  *** similar axioms for the other logical connectives
  ax (∀X.P)[T/X] = ∀X.P .
  cax (∀Y.P)[T/X] = ∀Y.(P[T/X])
      if not(X == Y) and (not(Y is-in va(T)) or not(X is-in fv(P))) .
  cax (∀Y.P)[Q/X] =
           ∀(new(va(T) U fv(P))).((P[new(va(T) U fv(P))/Y])[T/X])
      if not(X == Y) and Y is-in va(T) and X is-in fv(P) .
  ax ...  *** similar axioms for the existential quantifier
  ax [A][T/X] = [A[T/X]] .

  *** Rules for quantifiers
  rl ∀X.P => P[T/X] .
  rl P[T/X] => ∃X.P .
  crl P => ∀X.A ⅋ Q
      if  P => A ⅋ Q  and  not(X is-in fv(P ⊗ Q)) .
  crl P ⊗ ∃X.A => Q
      if  P ⊗ A => Q  and  not(X is-in fv(P ⊗ Q)) .
endm
```

In this way, we have defined a map of entailment systems

$$ent(FOLinLogic) \longrightarrow ent(OSRWLogic)$$

which is also *conservative*.

4.5 Mapping sequent systems

In Section 4.3, we have mapped linear logic formulas to terms, and linear logic sequents to rewrite rules in rewriting logic. There is another map of entailment systems between linear logic and rewriting logic in which linear sequents become also terms, and rewrite rules correspond to rules in a Gentzen sequent calculus for linear logic. In order to reduce the number of rules of this calculus, we consider one-sided linear sequents in this section, but a completely similar treatment can be given for two-sided sequents. Thus, a linear logic sequent will be a turnstile symbol "$\vdash$" followed by a multiset M of linear logic formulas, that in our translation to rewriting logic will be represented by the term `⊢ M`. Using the duality of linear logic

negation, a two-sided sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ can in this notation be expressed as the one-sided sequent $\vdash A_1^{\perp}, \ldots, A_n^{\perp}, B_1, \ldots, B_m$.

First, we define a parameterized module for multisets. The elements in the parameter are considered singleton multisets via a subsort declaration `Elem < Mset`, and there is a multiset union operator `_,_` which is associative, commutative, and has the empty multiset `null` as neutral element. Note that what makes the elements of `Mset` multisets instead of lists is the attribute `comm` of commutativity of the union operator `_,_`.

```
fth ELEM is
  sort Elem .
endft

fmod MSET[X :: ELEM] is
  sort Mset .
  subsort Elem < Mset .
  op null : -> Mset .
  op _,_ : Mset Mset -> Mset [assoc comm id: null] .
endfm
```

Now we can use this parameterized module to define the main module for sequents[12] and give the corresponding rules. A sequent calculus rule of the form

$$\frac{\vdash M_1, \ldots, \vdash M_n}{\vdash M}$$

becomes the rewrite rule

```
rl ⊢ M1 ... ⊢ Mn => ⊢ M .
```

on the sort `Configuration`. Recalling that "`---`" introduces a comment, this rule can be written as

```
rl    ⊢ M1 ... ⊢ Mn
   => --------------
          ⊢ M .
```

This displaying trick that makes it possible to write a sequent calculus rule in a similar way to the usual presentation in logical textbooks is due to K. Futatsugi.

[12] The multiset structure is one particular way of building in certain *structural rules*, in this case *exchange*. Many other such data structuring mechanisms are as well possible to build in, or to drop, desired structural properties. Appropriate parameterized data types can similarly be used for this purpose. For example, we use later a data type of lists to define 2-sequents in which exchange is not assumed.

```
--- one-sided sequent calculus for linear logic
mod LL-SEQUENT[X :: VAR] is
  protecting FO-PROP0[X] .
  including MSET[FO-PROP0[X]] .

  --- a configuration is a multiset of sequents
  sort Configuration .
  op ⊢_ : Mset -> Configuration .
  op empty : -> Configuration .
  op __ : Configuration Configuration -> Configuration
                                       [assoc comm id: empty] .
  op ?_ : Mset -> Mset .
  vars M N : Mset .
  ax ?null = null .
  ax ?(M,N) = (?M,?N) .
  op fv : Mset -> Set .
  ax fv(null) = emptyset .
  ax fv(M,N) = fv(M) U fv(N) .

  var P : Atom .    vars A B : Prop0 .
  var T : Term .    var X : Var .

  *** Identity
  rl      empty
     => --------
        ⊢ P,P^⊥ .

  *** Cut
  rl     (⊢ M,A) (⊢ N,A^⊥)
     => ------------------
              ⊢ M,N .

  *** Tensor
  rl     (⊢ M,A) (⊢ B,N)
     => ---------------
         ⊢ M,A ⊗ B,N .

  *** Par
  rl       ⊢ M,A,B
     => -----------
        ⊢ M,A ⅋ B .

  *** Plus
  rl        ⊢ M,A
     => -----------
        ⊢ M,A ⊕ B .
```

```
*** With
rl     (⊢ M,A) (⊢ M,B)
    => ---------------
         ⊢ M,A & B .

*** Weakening
rl       ⊢ M
    => --------
       ⊢ M,?A .

*** Contraction
rl     ⊢ M,?A,?A
    => ----------
        ⊢ M,?A .

*** Dereliction
rl      ⊢ M,A
    => --------
       ⊢ M,?A .

*** Storage
rl      ⊢ ?M,A
    => ---------
       ⊢ ?M,!A .

*** Bottom
rl       ⊢ M
    => -------
       ⊢ M,⊥ .

*** One
rl     empty
    => -----
       ⊢ 1 .

*** Top
rl      empty
    => --------
       ⊢ M,⊤ .

*** Universal
crl      ⊢ M,A
    => ---------
       ⊢ M,∀X.A
    if  not(X is-in fv(M)) .
```

```
  *** Existential
  rl     ⊢ M,A[T/X]
      => ----------
         ⊢ M.∃X.A .
endm
```

Note that in the module `FO-PROPO[X]` (via the reused theory `PROPO[X]`) we have imposed associativity and commutativity attributes for some connectives, making syntax a bit more abstract than usual. However, in this case, this has no significance at all, except for the convenient fact that we only need a rule for $\oplus$ instead of two; of course, these attributes can be removed if a less abstract presentation is preferred.

Given a linear theory $T = (C, S)$ (where we can assume that all the sequents in S are of the form $\vdash A_1, \ldots, A_n$), we instantiate the parameterized module `LL-SEQUENT[X]` using a functional module `C` that interprets the propositional constants in C, as in Section 4.3, and then extend it by adding a rule

```
rl empty => ⊢ A1,...,An .
```

for each sequent $\vdash A_1, \ldots, A_n$ in S, obtaining in this way a rewrite theory `LL-SEQUENT(T)`.

With this map we have also an immediate conservativity result:

THEOREM 15. Given a linear theory T, a linear logic sequent $\vdash A_1, \ldots, A_n$ is provable in linear logic from the axioms in T if and only if the sequent

```
empty ⟶ ⊢ A1,...,An
```

is provable in rewriting logic from the rewrite theory `LL-SEQUENT(T)`.

It is very important to realize that the technique used in this conservative map of entailment systems is very general and it is in no way restricted to linear logic. Indeed, it can be applied to any sequent calculus, be it for intuitionistic, classical or any other logic. In general, we need an operation

```
op _⊢_ : FormList FormList -> Sequent .
```

that turns two lists of formulas (multisets, or sets in some cases) into a term representing a sequent. Then we have a sort `Configuration` representing multisets of sequents, with a union operator written using empty syntax. A sequent calculus rule

$$\frac{G_1 \vdash D_1, \ldots, G_n \vdash D_n}{G \vdash D}$$

becomes a rewrite rule

```
rl (G1 ⊢ D1) ... (Gn ⊢ Dn) => (G ⊢ D) .
```

on the sort `Configuration`, that we have displayed above also as

```
rl    (G1 ⊢ D1) ... (Gn ⊢ Dn)
   => ------------------------
              (G ⊢ D) .
```

in order to make even clearer that the rewrite rule and the sequent notations in fact capture the same idea. In the particular case of linear logic the situation is somewhat simplified by the use of one-sided sequents. Notice also that sometimes the rewrite rule can be conditional to the satisfaction of some auxiliary side conditions like, for example, in the rule for the universal quantifier in the module above.

As another example illustrating the generality of this approach, we sketch a presentation in rewriting logic of the *2-sequent calculus* defined by Masini and Martini in order to develop a proof theory for modal logics [Masini, 1993; Martini and Masini, 1993]. In their approach, a *2-sequent* is an expression of the form $\Gamma \vdash \Delta$, where Γ and Δ are not lists of formulas as usual, but they are lists of lists of formulas, so that sequents are endowed with a vertical structure. For example,

$$\begin{array}{cc} A, B & \\ C & \end{array} \quad \vdash \quad \begin{array}{c} D \\ E, F \\ G \end{array}$$

is a 2-sequent, which will be represented in rewriting logic as

$$A, B; C \vdash D; E, F; G.$$

In order to define 2-sequents, we first need a parameterized module for lists, assuming a module `NAT` defining a sort `Nat` of natural numbers with zero `0`, a successor function `s_`, an addition operation `_+_`, and an order relation `_<=_`, as well as a module `BOOL` defining a sort `Bool` of truth values `true`, `false` and corresponding Boolean operations.

```
fmod LIST[X :: ELEM] is
  protecting NAT BOOL .
  sort List .
  subsort Elem < List .
  op nil : -> List .
  op _;_ : List List -> List [assoc id: nil] .
  op length : List -> Nat .
  op _in_ : Elem List -> Bool .

  vars E E' : Elem .
  vars L L' : List .
  eq length(nil) = 0 .
  eq length(E) = s0 .
```

```
  eq length(L;L') = length(L) + length(L') .
  eq E in nil = false .
  eq E in E' = if E == E' then true else false .
  eq E in (L;L') = (E in L) or (E in L') .
endfm
```

This module is instantiated twice in order to get the module of 2-sequents, using a sort of formulas `Form` whose definition is not presented here, and that should have an operation

```
  op []_ : Form -> Form .
```

corresponding to the modality $\Box$.

```
make 2-LIST is
  LIST[LIST[Form]*(op _;_ to _,_)]*(sort List to 2-List,
                                    op length to depth)
endmk
```

Note that in the `2-LIST` module the concatenation operation `_;_` is renamed to `_,_` in the case of lists of formulas, whereas in the case of lists of lists of formulas, called 2-lists, the notation `_;_` is kept. Also, to emphasize the vertical structure of 2-sequents, the operation `length` for 2-lists is renamed to `depth`.

Now we can define 2-sequents as follows:

```
fmod 2-SEQUENT is
  protecting 2-LIST .
  sort 2-Sequent .
  op _⊢_ : 2-List 2-List -> 2-Sequent .
endfm
```

The basic rules for the modality $\Box$ are

$$\frac{\begin{array}{c}\Gamma\\ \alpha\\ \beta, A\\ \Gamma'\end{array} \vdash \Delta}{\begin{array}{c}\Gamma\\ \alpha, \Box A\\ \beta\\ \Gamma'\end{array} \vdash \Delta}\ (\Box\text{-}L) \qquad \frac{\Gamma \vdash \begin{array}{c}\Delta\\ \alpha\\ A\end{array}}{\Gamma \vdash \begin{array}{c}\Delta\\ \alpha, \Box A\end{array}}\ (\Box\text{-}R)$$

where Γ, Γ', Δ denote 2-lists, α, β denote lists of formulas, and the rule $\Box$-R has the side condition that $\mathit{depth}(\Gamma) \leq \mathit{depth}(\Delta) + 1$, i.e., the formula A is the only formula in the last level of the 2-sequent.

These rules are represented in rewriting logic as follows.

```
mod 2-SEQUENT-RULES is
  protecting 2-SEQUENT .
  sort Configuration .
  subsort 2-Sequent < Configuration .
  op empty : -> Configuration .
  op __ : Configuration Configuration -> Configuration
                                       [assoc comm id: empty] .
  vars R R' S : 2-List .
  vars L L' : List .
  var A : Form .
  rl     R ; L ; L',A ; R' ⊢ S
     => ------------------------
        R ; L,[]A ; L' ; R' ⊢ S .

  crl   R ⊢ S ; L ; A
     => --------------
        R ⊢ S ; L,[]A
     if depth(R) <= s(depth(S)) .
endm
```

The dual rules for the modality $\Diamond$ are treated similarly.

This general method of viewing sequents as rewrite rules can even be applied to systems more general than traditional sequent calculi. Thus, besides the possibilities of being one-sided or two-sided, one-dimensional or two-dimensional, etc., a "sequent" can for example be a sequent presentation of natural deduction, a term assignment system, or even any predicate defined by structural induction in some way such that the proof is a kind of tree, as for example the operational semantics of CCS given later in Section 5.3 and any other use of the so-called structural operational semantics (see [Hennessy, 1990] and Section 5.4 later), including type-checking systems. The general idea is to map a rule in the "sequent" system to a rewrite rule over a "configuration" of sequents or predicates, in such a way that the rewriting relation corresponds to provability of such a predicate.

4.6 Reflection in rewriting logic

Clavel and Meseguer have shown in [1996; 1996a] that rewriting logic is reflective in the sense of Section 2.8. That is, there is a rewrite theory $\mathcal{U}$ with a finite number of operations and rules that can simulate any other finitely presentable rewrite theory $\mathcal{R}$ in the following sense: given any two terms t, t' in $\mathcal{R}$, there are corresponding terms $\langle \overline{\mathcal{R}}, \overline{t} \rangle$ and $\langle \overline{\mathcal{R}}, \overline{t'} \rangle$ in $\mathcal{U}$ such that we have

$$\mathcal{R} \vdash t \longrightarrow t' \iff \mathcal{U} \vdash \langle \overline{\mathcal{R}}, \overline{t} \rangle \longrightarrow \langle \overline{\mathcal{R}}, \overline{t'} \rangle.$$

Moreover, it is often possible to reify inside rewriting logic itself a representation map $\mathcal{L} \to \mathit{OSRWLogic}$ for the finitely presentable theories of $\mathcal{L}$.

Such a reification takes the form of a map between the abstract data types representing the finitary theories of $\mathcal{L}$ and of *OSRWLogic*. In this section we illustrate this powerful idea with the linear logic mapping defined in Section 4.3.

We have defined a linear theory T as a finite set C of propositional constants together with a finite set S of sequents of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$, where each A_i and B_j is a linear logic formula built from the constants in C. Note that with this definition, all linear theories are finitely presentable. First, we define an abstract data type `LL-ADT` to represent linear theories. A linear theory is represented as a term `<C | G>`, where `C` is a list of propositional constants (that is, identifiers), and `G` is a list of sequents written in the usual way. Moreover, all the propositional constants in `G` must be included in `C`. To enforce this condition, we use a sort constraint [Meseguer and Goguen, 1993], which is introduced with the keyword `sct` and defines a subsort `LLTheory` of a sort `LLTheory?` by means of the given condition. In the functional module below, we do not give the equations defining the auxiliary functions `const` that extracts the constants of a list of sequents, and the list containment predicate `_=<_`. These functions are needed to write down the sort constraint for theories.

```
fmod LL-ADT is
  protecting QID .
  sorts Ids Formula Formulas Sequent .
  sorts Sequents LLTheory? LLTheory .

  subsort Id < Formula .
  ops 1 0 ⊥ ⊤ : -> Formula .
  op _⊗_ : Formula Formula -> Formula .
  op _⅋_ : Formula Formula -> Formula .
  op _⊕_ : Formula Formula -> Formula .
  op _&_ : Formula Formula -> Formula .
  op !_ : Formula -> Formula .
  op ?_ : Formula -> Formula .
  op _⊥ : Formula -> Formula .

  subsort Formula < Formulas .
  op null : -> Formulas .
  op _,_ : Formulas Formulas -> Formulas [assoc comm id: null] .
  op (_⊢_) : Formulas Formulas -> Sequent .

  subsort Id < Ids .
  op nil : -> Ids .
  op _,_ : Ids Ids -> Ids [assoc id: nil] .

  subsort Sequent < Sequents .
  op nil : -> Sequents .
```

```
  op _,_ : Sequents Sequents -> Sequents [assoc id: nil] .
  op <_|_> : Ids Sequents -> LLTheory? .

  var C : Ids .
  var G : Sequents .
  sct <C | G> : LLTheory if const(G) =< C .
  eq ...
  *** several equations defining the auxiliary operations
  *** "const" and "_=<_" used in the sort constraint condition
  eq ...
endfm
```

An order-sorted rewrite theory has much more structure, and therefore the corresponding `RWL-ADT` is more complex, but the basic ideas are completely similar as we sketch here. First we have an order-sorted signature, declaring sorts, subsorts, constants, operations, and variables. Then, in addition, we have equations and rules. Thus, a finitely presentable rewrite theory is represented as a term `<S | E | R>`, where `S` is a term representing a signature, `E` is a list of equations, and `R` is a list of rules. In turn, the term `S` has the form `<T ; B ; C ; O ; V>` where each subterm corresponds to a component of a signature as mentioned before. In addition, several sort constraints are necessary to ensure for example that the variables used in equations and rules are included in the list of variables. Just to give the flavor of the construction, here is a small fragment of the module `RWL-ADT`, where we have omitted most of the list constructors, operations to handle conditional equations and rules, and sort constraints.

```
sorts Sort Subsort Constant Op Var .
sorts Term Equation Rule Signature RWLTheory .

op sort{_} : Id -> Sort .
subsort Sort < Sorts .
op nil : -> Sorts .
op __ : Sorts Sorts -> Sorts [assoc id: nil] .
op (_<_) : Id Id -> Subsort .
subsort Subsort < Subsorts .

op (cons{_}:sort{_}) : Id Id -> Constant .
subsort Constant < Constants .
op nil : -> Constants .
op _,_ : Constants Constants -> Sorts [assoc id: nil] .

op (op{_}:_->sort{_}) : Id Sorts Id -> Op .
subsort Op < Ops .
op (var{_}:sort{_}) : Id Id -> Var .
subsort Var < Vars .
```

```
op <_;_;_;_;_> : Sorts Subsorts Constants Ops Vars -> Signature .
subsort Var < Term .
subsort Constant < Term .
subsort Term < Terms .
op nil : -> Terms .
op op{_}[_] : Id Terms -> Term .
op _,_ : Terms Terms -> Terms [assoc id: nil] .

op (_=_) : Term Term -> Equation .
subsort Equation < Equations .
op (_=>_) : Term Term ->  Rule .
subsort Rule < Rules .
op <_|_|_> : Signature Equations Rules -> RWLTheory .
```

Having defined the abstract data types to represent both linear and rewrite theories, we define a function $\overline{\Phi}$ mapping a term in `LLTheory` representing a linear theory T to a term in `RWLTheory` representing the corresponding rewrite theory `LINLOG(T)` as defined in Section 4.3. First note that the rewrite theory `LINLOG` presented in Section 4.3 gives rise to a term in `RWLTheory` that we denote

$$\texttt{<<T}_{LL}\texttt{ ; B}_{LL}\texttt{ ; C}_{LL}\texttt{ ; O}_{LL}\texttt{ ; V}_{LL}\texttt{> | E}_{LL}\texttt{ | R}_{LL}\texttt{>}.$$

The representation `<C | F1` $\vdash$ `G1,...,Fn` $\vdash$ `Gn>` of a linear logic theory is then mapped by $\overline{\Phi}$ to the following term

```
<<T_LL ; B_LL ; cons(C),C_LL ; O_LL ; V_LL> | E_LL |
     R_LL,([tensor(F1)] => [par(G1)]),...,
          ([tensor(Fn)] => [par(Gn)])>
```

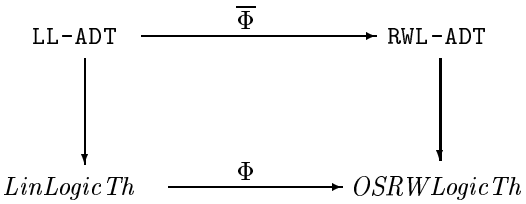
where the auxiliary operations `cons`, `tensor` and `par` are defined as follows, and correspond exactly to the description in Section 4.3.

```
op tensor : Formulas -> Formula .
op par : Formulas -> Formula .
op cons : Ids -> Constants .

var F : Formula .    vars F1 F2 : Formulas .
var I : Id .         var L : Ids .

eq tensor(null) = 1 .
eq tensor(F) = F .
eq tensor(F1,F2) = tensor(F1) ⊗ tensor(F2) .
eq par(null) = ⊥ .
eq par(F) = F .
eq par(F1,F2) = par(F1) ⅋ par(F2) .
eq cons(nil) = nil .
eq cons(I,L) = (cons{I}:sort{Atom}),cons(L) .
```

We can summarize the reification $\overline{\Phi}$: `LL-ADT` $\longrightarrow$ `RWL-ADT` of the map of logics Φ : *LinLogic* $\longrightarrow$ *OSRWLogic* we have just defined by means of the following commutative diagram:



This method is completely general, in that it should apply to any effectively presented map of logics $\Psi : \mathcal{L} \longrightarrow RWLogic$ that maps finitely presentable theories in $\mathcal{L}$ to finitely presentable theories in rewriting logic. Indeed, the effectiveness of Ψ should exactly mean that the corresponding $\overline{\Psi}$: $\mathcal{L}$`-ADT` $\longrightarrow$ `RWL-ADT` is a computable function and therefore, by the metatheorem of Bergstra and Tucker [1980], that it is specifiable by a finite set of Church-Rosser and terminating equations inside rewriting logic.

5 REWRITING LOGIC AS A SEMANTIC FRAMEWORK

After an overview of rewriting logic as a general model of computation that unifies many other existing models, the cases of concurrent object-oriented programming and of Milner's CCS are treated in greater detail. Structural operational semantics is discussed as a specification formalism similar in some ways to rewriting logic, but more limited in its expressive capabilities. Rewriting logic can also be very useful as a semantic framework for many varieties of constraint solving in logic programming and in automated deduction. Finally, the representation of action and change in rewriting logic and the consequent solution of the "frame problem" difficulties associated with standard logics are also discussed.

5.1 Generality of rewriting logic as a model of computation

Concurrent rewriting is a very general model of concurrency from which many other models can be obtained by specialization. Except for concurrent objet-oriented programming and CCS that are further discussed in Sections 5.2 and 5.3, respectively, we refer the reader to [Meseguer, 1992; Meseguer, 1996] for a detailed discussion of the remaining models, and summarize here such specializations using Figure 1, where RWL stands for rewriting logic, the arrows indicate specializations, and the subscripts $\emptyset$, AI, and ACI stand for syntactic rewriting, rewriting modulo associativity and identity, and rewriting modulo associativity, commutativity, and identity, respectively.

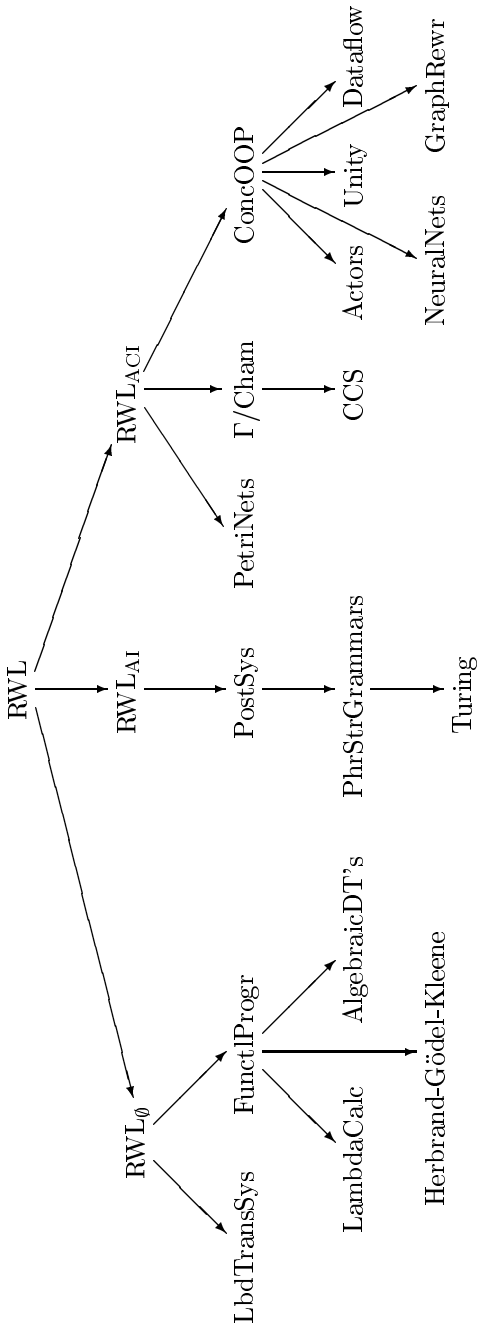


Figure 1. Unification of models of computation.

Within syntactic rewriting we have labelled transition systems, which are used in interleaving approaches to concurrency; functional programming (in particular Maude's functional modules) corresponds to the case of *confluent*[13] rules, and includes the lambda calculus and the Herbrand-Gödel-Kleene theory of recursive functions. Rewriting modulo AI yields Post systems and related grammar formalisms, including Turing machines. Besides the general treatment by ACI-rewriting of concurrent object-oriented programming, briefly described in Section 5.2, that contains Actors [Agha, 1986], neural networks, graph rewriting, and the dataflow model as a special case [Meseguer, 1996], rewriting modulo ACI includes Petri nets [Reisig, 1995], the Gamma language of Banâtre and Le Mètayer [1990], and Berry and Boudol's *chemical abstract machine* [1992] (which itself specializes to CCS [Milner, 1989]; see [Berry and Boudol, 1992] and also the treatment in Section 5.3), as well as Unity's model of computation [Chandy and Misra, 1988].

The ACI case is quite important, since it contains as special subcases a good number of concurrency models that have already been studied. In fact, the associativity and commutativity of the axioms appear in some of those models as "fundamental laws of concurrency." However, from the perspective of this work the ACI case, while being important and useful, does not have a monopoly on the concurrency business. Indeed, "fundamental laws of concurrency" expressing associativity and commutativity are only valid in this particular case. They are for example meaningless for the tree-structured case of functional programming. The point is that the laws satisfied by a concurrent system cannot be determined *a priori*. They essentially depend on the actual distributed structure of the system, which is its algebraic structure.

5.2 Concurrent object-oriented programming

Concurrent object-oriented programming is a very active area of research. An important reason for this interest is the naturalness with which this style of programming can model concurrent interactions between objects in the real world. However, the field of concurrent object-oriented programming seems at present to lack a clear, agreed-upon semantic basis.

Rewriting logic supports a logical theory of concurrent objects that addresses these conceptual needs in a very direct way. We summarize here the key ideas regarding Maude's object-oriented modules; a full discussion of Maude's object-oriented aspects can be found in [Meseguer, 1993; Meseguer, 1993b].

An *object* in a given state can be represented as a term

[13] Although not reflected in the picture, rules confluent *modulo* equations E are also functional.

```
< O : C | a1 : v1, ... , an : vn >
```

where `O` is the object's name, belonging to a set `OId` of object identifiers, `C` is its class, the `ai`'s are the names of the object's *attributes*, and the `vi`'s are their corresponding values, which typically are required to be in a sort appropriate for their corresponding attribute. The *configuration* is the distributed state of the concurrent object-oriented system and is represented as a multiset of objects and messages according to the following syntax:

```
subsorts Object Message < Configuration .
op __ : Configuration Configuration -> Configuration
                                    [assoc comm id: null] .
```

where the operator `__` is associative and commutative with identity `null` and is interpreted as multiset union, and the sorts `Object` and `Message` are subsorts of `Configuration` and generate data of that sort by multiset union. The system evolves by concurrent ACI-rewriting of the configuration by means of rewrite rules specific to each particular system, whose lefthand and righthand sides may in general involve patterns for several objects and messages. By specializing to patterns involving only one object and one message, we can obtain an abstract, declarative, and truly concurrent version of the Actor model [Agha, 1986] (see [Meseguer, 1993, Section 4.7]).

Maude's syntax for object-oriented modules is illustrated by the object-oriented module `ACCNT` below which specifies the concurrent behavior of objects in a very simple class `Accnt` of bank accounts, each having a `bal`(ance) attribute, which may receive messages for crediting or debiting the account, or for transferring funds between two accounts. We assume an already defined functional module `INT` for integers with a subsort relation `Nat < Int` and an ordering predicate `_>=_`.

After the keyword `class`, the name of the class (`Accnt` in this case) is given, followed by a "`|`" and by a list of pairs of the form `a : S` separated by commas, where `a` is an attribute identifier and `S` is the sort inside which the values of such an attribute identifier must range in the given class. In this example, the only attribute of an account is its `bal`(ance), which is declared to be a value in `Nat`. The three kinds of messages involving accounts are `credit`, `debit`, and `transfer` messages, whose user-definable syntax is introduced by the keyword `msg`. The rewrite rules specify in a declarative way the behavior associated to the `credit`, `debit`, and `transfer` messages.

```
omod ACCNT is
  protecting INT .
  class Accnt | bal : Nat .
  msgs credit debit : OId Nat -> Msg .
  msg transfer_from_to_ : Nat OId OId -> Msg .

  vars A B : OId .
```

```
  vars M N N' : Nat .
  rl credit(A,M) < A : Accnt | bal: N >
     => < A : Accnt | bal: N + M > .
  crl debit(A,M) < A : Accnt | bal: N >
      => < A : Accnt | bal: N - M >  if  N >= M .
  crl transfer M from A to B
       < A : Accnt | bal: N > < B : Accnt | bal: N' >
     => < A : Accnt | bal: N - M > < B : Accnt | bal: N' + M >
     if  N >= M .
endom
```

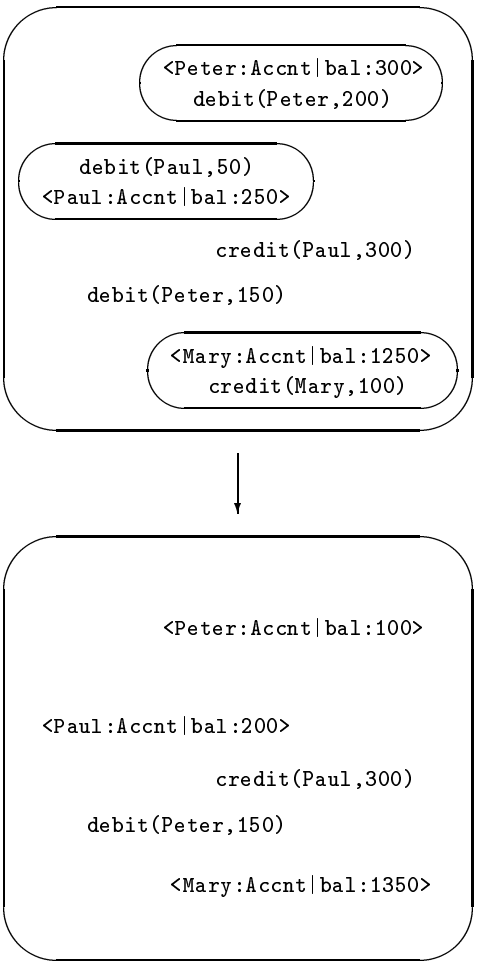


Figure 2. Concurrent rewriting of bank accounts.

The multiset structure of the configuration provides the top level distributed structure of the system and allows concurrent application of the rules. For example, Figure 2 provides a snapshot in the evolution by concurrent rewriting of a simple configuration of bank accounts. To simplify the picture, the arithmetic operations required to update balances have already been performed. However, the reader should bear in mind that the values in the attributes of an object can also be computed by means of rewrite rules, and this adds yet another important level of concurrency to a concurrent object-oriented system, which might be called *intra-object concurrency*.

Intuitively, we can think of messages as "traveling" to come into contact with the objects to which they are sent and then causing "communication events" by application of rewrite rules. In rewriting logic, this traveling is accounted for in a very abstract way by the ACI structural axioms. This abstract level supports both synchronous and asynchronous communication and provides great freedom and flexibility to consider a variety of alternative implementations at lower levels.

Although Maude provides convenient syntax for object-oriented modules, the syntax and semantics of such modules can be reduced to those of system modules, i.e., we can systematically translate an object-oriented module `omod` $\mathcal{O}$ `endom` into a corresponding system module `mod` $\mathcal{O}\#$ `endm`, where $\mathcal{O}\#$ is a theory in rewriting logic. A detailed account of this translation process can be found in [Meseguer, 1993].

5.3 CCS

Milner's *Calculus of Communicating Systems* (CCS) [Milner, 1980; Milner, 1989; Milner, 1990] is among the best well-known and studied concurrency models, and has become the paradigmatic example of an entire approach to "process algebras." We just give a very brief introduction to CCS, referring the reader to Milner's book [1989] for motivation and a comprehensive treatment, before giving two alternative formulations of CCS in rewriting logic and showing the conservativity of these formulations.

We assume a set A of *names*; the elements of the set $\overline{A} = \{\overline{a} \mid a \in A\}$ are called *co-names*, and the members of the (disjoint) union $\mathcal{L} = A \cup \overline{A}$ are *labels* naming ordinary actions. The function $a \mapsto \overline{a}$ is extended to $\mathcal{L}$ by defining $\overline{\overline{a}} = a$. There is a special action called *silent action* and denoted τ, intended to represent internal behaviour of a system, and in particular the synchronization of two processes by means of actions a and $\overline{a}$. Then the set of *actions* is $\mathcal{L} \cup \{\tau\}$. The set of processes is intuitively defined as follows:

- 0 is an inactive process that does nothing.
- If α is an action and P is a process, $\alpha.P$ is the process that performs α and subsequently behaves as P.

- If P and Q are processes, $P + Q$ is the process that may behave as either P or Q.
- If P and Q are processes, $P|Q$ represents P and Q running concurrently with possible communication via synchronization of the pair of ordinary actions a and $\overline{a}$.
- If P is a process and $f : \mathcal{L} \to \mathcal{L}$ is a relabelling function such that $f(\overline{a}) = \overline{f(a)}$, $P[f]$ is the process that behaves as P but with the actions relabelled according to f, assuming $f(\tau) = \tau$.
- If P is a process and $L \subseteq \mathcal{L}$ is a set of ordinary actions, $P\backslash L$ is the process that behaves as P but with the actions in $L \cup \overline{L}$ prohibited.
- If P is a process, I is a process identifier, and $I =_{def} P$ is a defining equation where P may recursively involve I, then I is a process that behaves as P.

This intuitive explanation can be made precise in terms of the following structural operational semantics that defines a labelled transition system for CCS processes.

Action:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

Summation:

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

Composition:

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

Relabelling:

$$\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

Restriction:

$$\frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L} \quad \alpha \notin L \cup \overline{L}$$

Definition:

$$\frac{P \xrightarrow{\alpha} P'}{I \xrightarrow{\alpha} P'} \quad I =_{def} P$$

We now show how CCS can be described and given semantics in rewriting logic. The following modules have been motivated by, but are considerably different from, the corresponding examples in [Meseguer *et al.*, 1992].

```
fth LABEL is
  sort Label .    *** ordinary actions
  op ~_ : Label -> Label .
  var N : Label .
  eq ~~N = N .
endft

--- an action is the silent action or a label
fmod ACTION[X :: LABEL] is
  sort Act .
  subsort Label < Act .
  op tau : -> Act .    *** silent action
endfm

fth PROCESSID is
  sort ProcessId .    *** process identifiers
endft

--- CCS syntax
fmod PROCESS[X :: LABEL, Y :: PROCESSID] is
  protecting ACTION[X] .
  sort Process .
  subsort ProcessId < Process .
  op 0 : -> Process .                        *** inaction
  op _._ : Act Process -> Process .          *** prefix
  op _+_ : Process Process -> Process [assoc comm idem id: 0] .
                                             *** summation
  op _|_ : Process Process -> Process [assoc comm id: 0] .
                                             *** composition
  op _[_/_] : Process Label Label -> Process .
                          *** relabelling: [b/a] relabels "a" to "b"
  op _\_ : Process Label -> Process .    *** restriction
endfm
```

Before defining the operational semantics of CCS processes, we need an auxiliary module in order to build contexts in which process identifiers can be associated with processes, providing in this way recursive definitions of processes. A sort constraint [Meseguer and Goguen, 1993], which is introduced with the keyword `sct` and defines a subsort `Context` by means

of a condition, is used to enforce the requirement that the same process identifier cannot be associated with two different processes in a context.

```
--- defining equations and contexts
fmod CCS-CONTEXT[X :: LABEL, Y :: PROCESSID] is
  protecting PROCESS[X,Y] .
  sorts Def Context Context? .
  subsorts Def < Context < Context? .
  op (_ =def _) : ProcessId Process -> Def .
  protecting LIST[ProcessId]*(op _;_ to __) .
  protecting LIST[Def]*(sort List to Context?) .
  op nil : -> Context .
  op pid : Context? -> List .

  var X : ProcessId .    var P : Process .
  var C : Context .      vars D D' : Context? .

  eq pid(nil) = nil .
  eq pid((X =def P)) = X .
  eq pid(D;D') = pid(D) pid(D') .
  sct (X =def P);C : Context  if  not(X in pid(C)) .
endfm
```

The semantics of CCS processes is usually defined relative to a given context that provides defining equations for all the necessary process identifiers [Milner, 1989, Section 2.4]. The previous module defines the data type of all contexts. We now need to parameterize the module defining the CCS semantics by the choice of a context. This is accomplished by means of the following theory that picks up a context in the sort `Context`.

```
fth CCS-CONTEXT*[X :: LABEL, Y :: PROCESSID] is
  protecting CCS-CONTEXT[X,Y] .
  op context : -> Context .
endft
```

As in the case of linear logic, we have two possibilities in order to write the operational semantics for CCS by means of rewrite rules. On the one hand, we can interpret a transition $P \xrightarrow{\alpha} P'$ as a rewrite, so that the above operational semantics rules become conditional rewrite rules. On the other hand, the transition $P \xrightarrow{\alpha} P'$ can be seen as a term, forming part of a configuration, in such a way that the semantics rules correspond to rewrite rules, as a particular case of the general mapping of sequent systems into rewriting logic that we have presented in Section 4.5.

```
--- CCS transitions
mod CCS1[X :: LABEL, Y :: PROCESSID, C :: CCS-CONTEXT*[X,Y]] is
  sort ActProcess .
```

```
  subsort Process < ActProcess .
  op {_}_ : Act ActProcess -> ActProcess .
  *** {A}P means action A has been performed thus becoming process P

  vars P P' Q Q' : Process .    vars L M : Label .
  var X : ProcessId .           var A : Act .

  *** Prefix
  rl A . P => {A}P .

  *** Summation
  crl P + Q => {A}P'  if  P => {A}P' .

  *** Composition
  crl P | Q => {A}(P' | Q)  if  P => {A}P' .
  crl P | Q => {tau}(P' | Q')  if  P => {L}P'  and  Q => {~L}Q' .

  *** Restriction
  crl P \ L => {A}(P' \ L)
      if  P => {A}P'  and  not(A == L)  and  not(A == ~L) .

  *** Relabelling
  crl P[M / L] => {M}(P'[M / L])  if  P => {L}P' .
  crl P[M / L] => {~M}(P'[M / L])  if  P => {~L}P' .
  crl P[M / L] => {A}(P'[M / L])
      if  P => {A}P'  and  not(A == L)  and  not(A == ~L) .

  *** Definition
  crl X => {A}P'  if  (X =def P) in context  and  P => {A}P' .
endm
```

In the above module, the rewrite rules have the property of being sort-increasing, i.e., in a rule $[t] \longrightarrow [t']$ the least sort of $[t']$ is bigger than the least sort of $[t]$. Thus, one rule cannot be applied unless the resulting term is well-formed. This prevents, for example, rewrites of the following form:

```
{A}(P | Q) ⟶  {A}({B}P' | {C}Q')
```

because the term on the righthand side is not well formed according to the order-sorted signature of the module `CCS1[X,Y,C[X,Y]]`. More precisely, the *Congruence* rule of order-sorted rewriting logic, like the corresponding rule of order-sorted algebra [Goguen and Meseguer, 1992], cannot be applied unless the resulting term $f(t_1, \ldots, t_n)$ is well formed according to the given order-sorted signature. To illustrate this point further, although `A.P` $\longrightarrow$ `{A}P` is a correct instance of the `Prefix` rewrite rule, we cannot use the *Congruence* rule to derive

```
(A.P) | Q ⟶ ({A}P) | Q
```

because the second term `({A}P) | Q` is not well formed.

The net effect of this restriction is that an `ActProcess` term of the form `{A1}...{Ak}P` can only be rewritten into another term of the same form `{A1}...{Ak}{B}P'`, assuming that `P` $\longrightarrow$ `{B}P'` is a `CCS1[X,Y,C[X,Y]]`-rewrite. As another example, a process of the form `A.B.P` can be rewritten first into `{A}B.P` and then into `{A}{B}P`, but cannot be rewritten into `A.{B}P`, because this last term is not well formed. After this discussion, it is easy to see that we have the following conservativity result.

THEOREM 16. Given a CCS process P, there are processes $P_1, \ldots, P_{k-1}$ such that

$$P \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} P_{k-1} \xrightarrow{a_k} P'$$

if and only if `P` can be rewritten into `{a1}...{ak}P'` using the rules in the module `CCS1[X,Y,C[X,Y]]`.

Note also that, since the operators `_+_` and `_|_` are declared commutative, one rule is enough for each one, instead of the two rules in the original presentation. On the other hand, we need three rules for relabelling, due to the representation of the relabelling function.

Let us consider now the second possibility, using the same idea described in Section 4.5 for the linear logic sequent calculus, that, as we have already mentioned there, is applicable to many more cases, with a very broad understanding of the term "sequent."

```
--- CCS operational semantics
mod CCS2[X :: LABEL, Y :: PROCESSID, C :: CCS-CONTEXT*[X,Y]] is
  sort Configuration .

  op (_:_-->_) : Act Process Process -> Configuration .
  op empty : -> Configuration .
  op __ : Configuration Configuration -> Configuration
                                      [assoc comm id: empty] .
  *** a configuration is a multiset of transitions

  vars P P' Q Q' : Process .    vars L M : Label .
  var X : ProcessId .           var A : Act .

  *** Prefix
  rl              empty
      => ---------------------
         (A : (A . P) --> P) .

  *** Summation
  rl         (A : P --> P')
      => --------------------
         (A : P + Q --> P') .
```

```
  *** Composition
  rl          (A : P --> P')
      => ------------------------
         (A : P | Q --> P' | Q) .

  rl     (L : P --> P')(~L : Q --> Q')
      => ----------------------------
          (tau : P | Q --> P' | Q') .

  *** Restriction
  crl         (A : P --> P')
      => ---------------------
         (A : P \ L --> P' \ L)
      if  not(A == L)  and  not(A == ~L) .

  *** Relabelling
  rl             (L : P --> P')
      => ----------------------------
         (M : P[M / L] --> P'[M / L]) .

  rl             (~L : P --> P')
      => -----------------------------
         (~M : P[M / L] --> P'[M / L]) .

  crl           (A : P --> P')
      => ---------------------------
         (A : P[M / L] --> P'[M / L])
      if  not(A == L)  and  not(A == ~L) .

  *** Definition
  crl    (A : P --> P')
      => --------------
         (A : X --> P')
      if (X =def P) in context .
endm
```

Except for the difference in the number of rules for some operators, as already pointed out above for the module `CCS1[X,Y,C[X,Y]]`, this presentation is closer to the original one, and therefore the following conservativity result is immediate.

THEOREM 17. For CCS processes P and P', a transition $P \xrightarrow{A} P'$ is possible according to the structural operational semantics of CCS if and only if

```
empty ⟶ (A : P --> P')
```

is provable in rewriting logic from the rewrite theory `CCS2[X,Y,C[X,Y]]`.

5.4 Structural operational semantics

Structural operational semantics is an approach originally introduced by Plotkin [1981] in which the operational semantics of a programming language is specified in a logical way, independent of machine architecture or implementation details, by means of rules that provide an inductive definition based on the structure of the expressions in the language. We refer the reader to Hennessy's book [1990] for a clear introduction to this subject.

Within "structural operational semantics," two main approaches coexist:

- *Big-step semantics* (also called *natural semantics* by Kahn [1987], Gunter [1991], and Nielson and Nielson [1992], and *evaluation semantics* by Hennessy [1990]). In this approach, the main inductive predicate describes the overall result or value of executing a computation until its termination. For this reason, it is not well suited for languages like CCS where most programs are not intended to be terminating.

- *Small-step semantics* (also called *structural operational semantics* by Plotkin [1981], and Nielson and Nielson [1992], *computation semantics* by Hennessy [1990], and *transition semantics* by Gunter [1991]). In this approach, the main inductive predicate describes in more detail the execution of individual steps in a computation, with the overall computation roughly corresponding to the transitive closure of such small steps. The structural operational semantics of CCS presented at the beginning of Section 5.3 is an example.

Both big-step and small-step approaches to structural operational semantics can be naturally expressed in rewriting logic:

- Big-step semantics can be seen as a particular case of the mapping of sequent systems described in Section 4.5, where semantics rules are mapped to rewrite rules over a "configuration" of sequents or predicates, and the rewriting relation means provability of such a predicate.

- Small-step semantics corresponds to the use of conditional rewrite rules, where a rewrite $t \longrightarrow t'$ means a transition or computation step from a state t to a new state t' as in the explanation of rewriting logic given in Section 3.3. This is illustrated by the `CCS1[X,Y,C[X,Y]]` example in Section 5.3. However, as the `CCS2[X,Y,C[X,Y]]` example shows, the technique of sequent systems of Section 4.5 can also be used in this case.

Since the CCS example has already been discussed in detail in Section 5.3, we give here another example, describing the operational semantics of the functional language Mini-ML taken with slight modifications from Kahn's

paper [1987]. The first thing to point out about this example is that the specification of a language's syntax is outside of the structural operational semantics formalism. By contrast, thanks to the order-sorted type structure of rewriting logic, such specification is now given by a functional module in Maude, as follows:

```
fmod NAT-TRUTH-VAL is
  sort Nat .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  sort TruthVal .
  ops true false : -> TruthVal .
endfm

--- syntax: values, patterns and expressions
fmod ML-SYNTAX[X :: VAR] is
  protecting NAT-TRUTH-VAL .
  sorts Exp Value Pat NullPat Lambda .

  subsorts NullPat Var < Pat .
  op () : -> NullPat .
  op (_,_) : Pat Pat -> Pat .
  subsorts TruthVal Nat NullPat < Value .
  op (_,_) : Value Value -> Value .
  subsorts Value Var Lambda < Exp .
  op s : Exp -> Exp .
  op _+_ : Exp Exp -> Exp [comm] .
  op not : Exp -> Exp .
  op _and_ : Exp Exp -> Exp .
  op if_then_else_ : Exp Exp Exp -> Exp .
  op (_,_) : Exp Exp -> Exp .
  op __ : Exp Exp -> Exp .
  op λ_._ : Pat Exp -> Lambda .
  op let_=_in_ : Pat Exp Exp -> Exp .
  op letrec_=_in_ : Pat Exp Exp -> Exp .
endfm

--- environments are lists of pairs pattern-value
fmod AUX[X :: VAR] is
  protecting ML-SYNTAX[X] .
  sort Pair .
  op <_,_> : Pat Value -> Pair .
  protecting LIST[Pair]*(sort List to Env, op _;_ to __) .
  op Clos : Lambda Env -> Value .
endfm
```

The following module constitutes a direct translation of the natural semantics specification for Mini-ML given by Kahn in [1987], using the general

technique for sequent systems introduced in Section 4.5. Note that the natural semantics rules are particularly well suited for Prolog search, and indeed they are so executed in the system described in [Kahn, 1987].

```
--- natural semantics a la Kahn
mod ML-NAT-SEMANT[X :: VAR] is
  including AUX[X] .
  sort Config .
  op (_|-_-->_) : Env Exp Value -> Config .
  op empty : -> Config .
  op __ : Config Config -> Config [assoc comm id: empty] .

  vars V W : Env .        vars E F G : Exp .
  vars X Y : Var .        vars P Q : Pat .
  vars A B C : Value .    vars N M : Nat .
  var T : TruthVal .

  *** Variables
  rl             empty
     => ------------------------
        ((V <X,A>) |- X --> A).

  crl       (V |- X --> A)
     => ----------------------
        ((V <Y,B>) |- X --> A)
     if  not(X == Y) .

  rl      (V <P,A> <Q,B> |- X --> C)
     => ------------------------------
        (V <(P,Q),(A,B)> |- X --> C) .

  *** Arithmetic expressions
  rl         empty
     => ----------------
        (V |- 0 --> 0) .

  rl        (V |- E --> A)
     => ----------------------
        (V |- s(E) --> s(A)) .

  crl   (V |- E --> A)(V |- F --> B)
     => ---------------------------
           (V |- E + F --> C)
     if  A + B => C .

  rl 0 + N => N .
  rl s(N) + s(M) => s(s(N + M)) .
```

```
*** Boolean expressions
rl            empty
   => ---------------------
      (V |- true --> true) .

rl             empty
   => ------------------------
      (V |- false --> false) .

rl        (V |- E --> true)
   => -------------------------
      (V |- not(E) --> false) .

rl       (V |- E --> false)
   => ------------------------
      (V |- not(E) --> true) .

crl   (V |- E --> A)(V |- F --> B)
   => ---------------------------
         (V |- E and F --> C)
   if  (A and B) => C .

rl T and true => T .
rl T and false => false .

*** Conditional expressions
rl     (V |- E --> true)(V |- F --> A)
   => ---------------------------------
      (V |- if E then F else G --> A) .

rl    (V |- E --> false)(V |- G --> A)
   => --------------------------------
      (V |- if E then F else G --> A) .

*** Pair expressions
rl          empty
   => ------------------
      (V |- () --> ()) .

rl    (V |- E --> A)(V |- F --> B)
   => ---------------------------
        (V |- (E,F) --> (A,B)) .

*** Lambda expressions
rl                empty
   => ------------------------------
      (V |- λP.E --> Clos(λP.E,V)) .
```

```
  rl    (V |- E --> Clos(λP.G,W))(V |- F --> A)(W <P,A> |- G --> B)
     => ------------------------------------------------------------
                             (V |- E F --> B) .

  *** Let and letrec expressions
  rl    (V |- F --> A)(V <P,A> |- E --> B)
     => ---------------------------------
           (V |- let P = F in E --> B) .

  rl    (V <P,A> |- F --> A)(V <P,A> |- E --> B)
     => --------------------------------------
            (V |- letrec P = F in E --> B) .
endm
```

The following module gives an alternative description of the semantics of the Mini-ML language in terms of the small-step approach. The rules can be directly used to perform reduction on Mini-ML expressions, and therefore constitute a very natural functional interpreter for the language.

```
--- sos semantics
mod ML-SOS-SEMANT[X :: VAR] is
  including AUX[X] .
  op [[_]]_ : Exp Env -> Value .

  vars V W : Env .    vars E F G : Exp .
  vars X Y : Var .    vars P Q : Pat .
  vars A B : Value .

  *** Variables
  rl [[X]](V <X,A>) => A .
  crl [[X]](V <Y,B>) => [[X]]V  if  not(X == Y) .
  rl [[X]](V <(P,Q),(A,B)>) => [[X]](V <P,A> <Q,B>) .

  *** Arithmetic expressions
  rl 0 + E => E .
  rl s(E) + s(F) => s(s(E + F)) .
  rl [[0]]V => 0 .
  rl [[s(E)]]V => s([[E]]V) .
  rl [[E + F]]V => [[E]]V + [[F]]V .

  *** Boolean expressions
  rl not(false) => true .
  rl not(true) => false .
  rl E and true => E .
  rl E and false => false .
  rl [[true]]V => true .
  rl [[false]]V => false .
```

```
  rl [[not(E)]]V => not([[E]]V) .
  rl [[E and F]]V => [[E]]V and [[F]]V .

  *** Conditional expressions
  rl if true then E else F => E .
  rl if false then E else F => F .
  rl [[if E then F else G]]V => if [[E]]V then [[F]]V else [[G]]V .

  *** Pair expressions
  rl [[()]]V => () .
  rl [[(E,F)]]V => ([[E]]V,[[F]]V) .

  *** Lambda expressions
  rl [[λP.E]]V => Clos(λP.E,V) .
  rl [[E F]]V => [[E]]V [[F]]V .
  rl Clos(λP.E,W) [[F]]V => [[E]](W <P,[[F]]V>) .

  *** Let and letrec expressions
  rl [[let P = E in F]]V => [[F]](V <P,[[E]]V>) .
  crl [[letrec P = E in F]]V => [[F]](V <P,A>)
      if  [[E]]((V <P,A>) => A .
endm
```

This concludes our discussion of structural operational semantics. Compared with rewriting logic, one of its limitations is the lack of support for structural axioms yielding more abstract data representations. Therefore, the rules must follow a purely syntactic structure, and more rules may in some cases be necessary than if an abstract representation had been chosen. In the case of multiset representations (corresponding to associativity, commutativity, and identity axioms), this has led Milner to favor multiset rewriting presentations [Milner, 1992] in the style of the chemical abstract machine of Berry and Boudol [1992] over the traditional syntactic presentation of structural operational semantics.

5.5 Constraint solving

Deduction can in many cases be made much more efficient by making use of *constraints* that can drastically reduce the search space, and for which special purpose constraint solving algorithms can be much faster than the alternative of expressing everything in a unique deduction mechanism such as some form of resolution.

Typically, constraints are symbolic expressions associated with a particular *theory*, and a constraint solving algorithm uses intimate knowledge about the truths of the theory in question to find solutions for those expressions by transforming them into expressions in *solved form*. One of the simplest examples is provided by standard syntactic unification—the

constraint solver for resolution in first-order logic without equality and in particular for Prolog—where the constraints in question are equalities between terms in a free algebra, i.e., in the so-called Herbrand universe. There are however many other constraints and constraint solving algorithms that can be used to advantage in order to make the representation of problems more expressive and logical deduction more efficient. For example,

- *Semantic unification* (see for example the survey by Jouannaud and Kirchner [1991]), which corresponds to solving equations in a given equational theory.
- *Sorted unification*, either many-sorted or order-sorted [Walther, 1985; Walther, 1986; Schmidt-Schauss, 1989; Meseguer *et al.*, 1989; Smolka *et al.*, 1989; Jouannaud and Kirchner, 1991], where type constraints are added to variables in equations.
- *Higher-order unification* [Huet, 1973; Miller, 1991], which corresponds to solving equations between λ-expressions.
- *Disunification* [Comon, 1991], which corresponds to solving not only equalities but also negated equalities.
- *Solution of equalities and inequalities in a theory*, as for example the solution of numerical constraints built into the constraint logic programming language $CLP(\mathcal{R})$ [Jaffar and Lassez, 1987] and in other languages.

A remarkable property shared by most constraint-solving processes, and already implicit in the approach to syntactic unification problems proposed by Martelli and Montanari [1982], is that the process of solving constraints can be naturally understood as one of applying transformations to a set or multiset of constraints. Furthermore, many authors have realized that the most elegant and simple way to specify, prove correct, or even implement many constraint solving problems is by expressing those transformations as rewrite rules (see for example [Goguen and Meseguer, 1988; Jouannaud and Kirchner, 1991; Comon, 1990; Comon, 1991; Nipkow, 1993]). In particular, the survey by Jouannaud and Kirchner [1991] makes this viewpoint the cornerstone of a unified conceptual approach to unification.

For example, the so-called *decomposition* transformation present in syntactic unification and in a number of other unification algorithms can be expressed by a rewrite rule of the form

```
rl f(t1,...,tn) =?= f(t'1,...,t'n)
   => (t1 =?= t'1) ... (tn =?= t'n) .
```

where in the righthand side multiset union has been expressed by juxtaposition.

Although the operational semantics of such rewrite rules is very obvious and intuitive, their logical or mathematical semantics has remained ambiguous. Although appeal is sometimes made to equational logic as the framework in which such rules exist, the fact that many of these rules are nondeterministic, so that, except for a few exceptions such as syntactic unification, there is in general not a unique solution but rather a, sometimes infinite, set of solutions, makes an interpretation of the rewrite rules as equations highly implausible and potentially contradictory.

We would like to suggest that rewriting logic provides a very natural framework in which to interpret rewrite rules of this nature and, more generally, deduction processes that are nondeterministic in nature and involve the exploration of an entire space of solutions. Since in rewriting logic rewrite rules go only in one direction and its models do not assume either the identification of the two sides of a rewrite step, or even the possible reversal of such a step, all the difficulties involved in an equational interpretation disappear.

Such a proposed use of rewriting logic for constraint solving and constraint programming seems very much in the spirit of recent rewrite rule approaches to constrained deduction such as those of C. Kirchner, H. Kirchner, and M. Rusinovitch [1990] (who use a general notion of constraint language proposed by Smolka [1989]), Bachmair, Ganzinger, Lynch, and Snyder [1992], Nieuwenhuis and Rubio [1992], and Giunchiglia, Pecchiari, and Talcott [1996]. In particular, the ELAN language of C. Kirchner, H. Kirchner, and M. Vittek [1995] (see also [Borovanský *et al.*, 1996]) proposes an approach to the prototyping of constraint solving languages similar in some ways to the one that would be natural using a Maude interpreter.

Exploring the use of rewriting logic as a semantic framework for languages and theorem-proving systems using constraints seems a worthwhile research direction not only for systems used in automated deduction, but also for parallel logic programming languages such as those surveyed in [Shapiro, 1989], the Andorra language [Janson and Haridi, 1991], concurrent constraint programming [Saraswat, 1992], and the Oz language [Henz *et al.*, 1995].

5.6 Action and change in rewriting logic

In the previous sections, we have shown the advantages of rewriting logic as a logical framework in which other logics can be represented, and as a semantic framework for the specification of languages and systems. We would like the class of systems that can be represented to be as wide as possible, and their representation to be as natural and direct as possible. In particular, an important point that has to be considered is the representation of action and change in rewriting logic. In our paper [Martí-Oliet and Meseguer, 1999], we show that rewriting logic overcomes the frame problem, and subsumes and

unifies a number of previously proposed logics of change. In this section, we illustrate this claim by means of an example, referring the reader to the cited paper for more examples and discussion.

The frame problem [McCarthy and Hayes, 1969; Hayes, 1987; Janlert, 1987] consists in formalizing the assumption that facts are preserved by an action unless the action explicitly says that a certain fact becomes true or false. In the words of Patrick Hayes [1987],

> "There should be some economical and principled way of succintly saying what changes an action makes, without having to explicitly list all the things it doesn't change as well [...]. *That* is the frame problem."

Recently, some new logics of action and change have been proposed, among which we can point out the approach of Hölldobler and Schneeberger [1990] (see also [Große *et al.*, 1996; Große *et al.*, 1992]), based on Horn logic with equations, and the approach of Masseron, Tollu, and Vauzeilles [1990; 1993], based on linear logic. The main interest of these formalisms is that they need not explicitly state frame axioms, because they treat facts as resources which are produced and consumed. Having proved in Sections 4.2 and 4.3, respectively, that Horn logic with equations and linear logic can be conservatively mapped into rewriting logic, it is not surprising that the advantages of the two previously mentioned approaches are also shared by rewriting logic. In particular, the rewriting logic rules automatically take care of the task of preserving context, making unnecessary the use of any frame axioms stating the properties that do not change when a rule is applied to a certain state.

We illustrate this point by means of a blocksworld example, borrowed from [Hölldobler and Schneeberger, 1990; Masseron *et al.*, 1990].

```
fth BLOCKS is
  sort BlockId .
endft

mod BLOCKWORLD[X :: BLOCKS] is
  sort Prop .
  op table : BlockId -> Prop .        *** block is on the table
  op on : BlockId BlockId -> Prop .   *** block A is on block B
  op clear : BlockId -> Prop .        *** block is clear
  op hold : BlockId -> Prop .         *** robot hand is holding block
  op empty : -> Prop .                *** robot hand is empty

  sort State .
  subsort Prop < State .
  op 1 : -> State .
  op _⊗_ : State State -> State [assoc comm id: 1] .
```

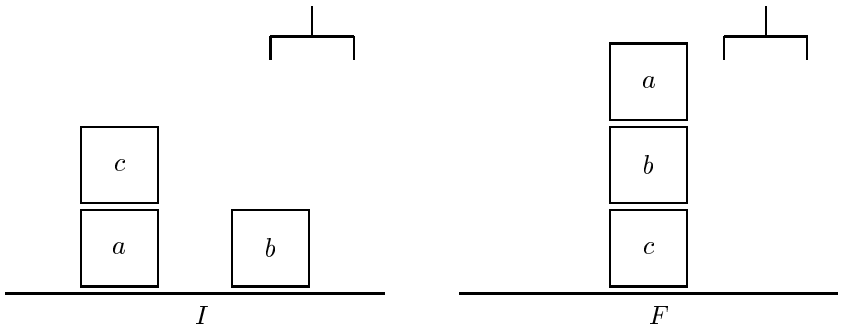


Figure 3. Two states of a blocksworld.

```
  vars X Y : BlockId .
  rl pickup(X) : empty ⊗ clear(X) ⊗ table(X) => hold(X) .
  rl putdown(X) : hold(X) => empty ⊗ clear(X) ⊗ table(X) .
  rl unstack(X,Y) : empty ⊗ clear(X) ⊗ on(X,Y)
     => hold(X) ⊗ clear(Y) .
  rl stack(X,Y) : hold(X) ⊗ clear(Y)
     => empty ⊗ clear(X) ⊗ on(X,Y) .
endm
```

In order to create a world with three blocks $\{a, b, c\}$, we consider the following instantiation of the previous parameterized module.

```
fmod BLOCKS3 is
  sort BlockId .
  ops a b c : -> BlockId .
endfm

  make WORLD is BLOCKWORLD[BLOCKS3] endmk
```

Consider the states described in Figure 3; the state I on the left is the initial one, described by the following term of sort `State` in the rewrite theory (Maude program) `WORLD`

```
  empty ⊗ clear(c) ⊗ clear(b) ⊗ table(a) ⊗ table(b) ⊗ on(c,a) .
```

Analogously, the final state F on the right is described by the term

```
  empty ⊗ clear(a) ⊗ table(c) ⊗ on(a,b) ⊗ on(b,c) .
```

The fact that the plan

```
  unstack(c,a);putdown(c);pickup(b);stack(b,c);pickup(a);stack(a,b)
```

moves the blocks from state I to state F corresponds directly to the following WORLD-rewrite (proof in rewriting logic), where we also show the use of the structural axioms of associativity and commutativity:

```
empty ⊗ clear(c) ⊗ clear(b) ⊗ table(a) ⊗ table(b) ⊗ on(c,a)
 =
empty ⊗ clear(c) ⊗ on(c,a) ⊗ clear(b) ⊗ table(a) ⊗ table(b)
⟶     Cong[Repl[unstack(c,a)],Refl]
hold(c) ⊗ clear(a) ⊗ clear(b) ⊗ table(a) ⊗ table(b)
⟶     Cong[Repl[putdown(c)],Refl]
empty ⊗ clear(c) ⊗ table(c) ⊗ clear(a) ⊗ clear(b) ⊗
                                        table(a) ⊗ table(b)
 =
empty ⊗ clear(b) ⊗ table(b) ⊗ clear(c) ⊗ table(c) ⊗
                                        clear(a) ⊗ table(a)
⟶     Cong[Repl[pickup(b)],Refl]
hold(b) ⊗ clear(c) ⊗ table(c) ⊗ clear(a) ⊗ table(a)
⟶     Cong[Repl[stack(b,c)],Refl]
empty ⊗ clear(b) ⊗ on(b,c) ⊗ table(c) ⊗ clear(a) ⊗ table(a)
 =
empty ⊗ clear(a) ⊗ table(a) ⊗ clear(b) ⊗ on(b,c) ⊗ table(c)
⟶     Cong[Repl[pickup(a)],Refl]
hold(a) ⊗ clear(b) ⊗ on(b,c) ⊗ table(c)
⟶     Cong[Repl[stack(a,b)],Refl]
empty ⊗ clear(a) ⊗ on(a,b) ⊗ on(b,c) ⊗ table(c)
 =
empty ⊗ clear(a) ⊗ table(c) ⊗ on(a,b) ⊗ on(b,c)
```

Hopefully this notation is self-explanatory. For example, the expression *Cong*[*Repl*[pickup(b)],*Refl*] means the application of the *Congruence* rule of rewriting logic to the two WORLD-rewrites obtained by using *Replacement* with the rewrite rule pickup(b) and *Reflexivity*. The *Transitivity* rule is used several times to go from the initial state I to the final state F.

Große, Hölldobler, and Schneeberger prove in [1996] (see also [Große *et al.*, 1992; Hölldobler, 1992]) that, in the framework of conjunctive planning, there is an equivalence between plans generated by linear logic proofs as used by Masseron *et al.* [1990; 1993], and the equational Horn logic approach of Hölldobler and Schneeberger [1990]. In the light of the example above, it is not surprising that we can add to the above equivalence the plans generated by proofs in rewriting logic [Martí-Oliet and Meseguer, 1999]. Moreover, this result extends to the case of disjunctive planning [Brüning *et al.*, 1993; Martí-Oliet and Meseguer, 1999]. In our opinion, rewriting logic compares favorably with these formalisms, not only because it subsumes them, but also because it is intrinsically concurrent, and it is more flexible and general, supporting user-definable logical connectives, which can be chosen to fit the problem at hand. In the words of Reichwein, Fiadeiro, and Maibaum [1992],

> "It is not enough to have a convenient formalism in which to represent action and change: the representation has to reflect the structure of the represented system."

In this respect, we show in [Martí-Oliet and Meseguer, 1999] that the object-oriented point of view supported by rewriting logic becomes very helpful in order to represent action and change.

6 CONCLUDING REMARKS

Rewriting logic has been proposed as a logical framework that seems particularly promising for representing logics, and its use for this purpose has been illustrated in detail by a number of examples. The general way in which such representations are achieved is by:

- Representing formulas or, more generally, proof-theoretic structures such as sequents, as *terms* in an order-sorted equational data type whose equations express structural axioms natural to the logic in question.
- Representing the rules of deduction of a logic as rewrite rules that transform certain patterns of formulas into other patterns modulo the given structural axioms.

Besides, the theory of general logics [Meseguer, 1989] has been used as both a method and a criterion of adequacy for defining these representations as conservative maps of logics or of entailment systems. From this point of view, our tentative conclusion is that, at the level of entailment systems, rewriting logic should in fact be able to represent any finitely presented logic via a conservative map, for any reasonable notion of "finitely presented logic." Making this tentative conclusion definite will require proposing an intuitively reasonable formal version of such a notion in a way similar to previous proposals of this kind by Smullyan [1961] and Feferman [1989].

In some cases, such as for equational logic, Horn logic with equality, and linear logic, we have in fact been able to represent logics in a much stronger sense, namely by conservative maps of logics that also map the models. Of course, such maps are much more informative, and may afford easier proofs, for example for conservativity. However, one should not expect to find representations of this kind for logics whose model theory is very different from that of rewriting logic.

Although this paper has studied the use of rewriting logic as a logical framework, and not as a metalogical one in which metalevel resoning about an object logic is performed, this second use is not excluded and is indeed one of the most interesting research directions that we plan to study. For this purpose, as stressed by Constable [1995], we regard *reflection* as a key

technique to be employed. Some concrete evidence for the usefulness of reflection has been given in Section 4.6.

The uses of rewriting logic as a semantic framework for the specification of languages, systems, and models of computation have also been discussed and illustrated with examples. Such uses include the specification and prototyping of concurrent models of computation and concurrent object-oriented systems, of general programming languages, of automated deduction systems and logic programming languages that use constraints, and of logical representation of action and change in AI.

From a pragmatic point of view, the main goal of this study is to serve as a guide for the design and implementation of a theoretically-based high-level system in which it can be easy to define logics and to perform deductions in them, and in which a very wide variety of systems, languages, and models of computation can similarly be specified and prototyped. Having this goal in mind, the following features seem particularly useful:

- *Executability*, which is not only very useful for prototyping purposes, but is in practice a must for debugging specifications of any realistic size.
- *Abstract user-definable syntax*, which can be specified as an order-sorted equational data type with the desired structural axioms.
- *Modularity and parameterization*[14], which can make specifications very readable and reusable by decomposing them in small understandable pieces that are as general as possible.
- *Simple and general logical semantics*, which can naturally express both logical deductions and concurrent computations.

These features are supported by the Maude interpreter [Clavel *et al.*, 1996]. A very important additional feature that the Maude interpreter has is good support for flexible and expressive strategies of evaluation [Clavel *et al.*, 1996; Clavel and Meseguer, 1996a], so that the user can explore the space of rewritings in intelligent ways.

POSTSCRIPT (2001)

During the five years that have passed since this paper was last revised until its final publication, the ideas put forward here have been greatly developed by several researchers all over the world. The survey paper [Martí-Oliet and Meseguer, 2001] provides a recent snapshot of the state of the art in the theory and applications of rewriting logic with a bibliography including

[14] Parameterization is based on the existence of relatively free algebras in rewriting logic, which generalizes the existence of initial algebras.

more than three hundred papers in this area. Here we provide some pointers to work closely related to the main points developed in this paper, and refer the reader to [Martí-Oliet and Meseguer, 2001] for many more references.

The paper [Clavel *et al.*, 2001] explains and illustrates the main concepts behind Maude's language design. The Maude system, a tutorial and a manual, a collection of examples and case studies, and a list of related papers are available at `http://maude.csl.sri.com`.

The reflective properties of rewriting logic and its applications have been developed in detail in [Clavel, 2000; Clavel and Meseguer, 2001]. A full reflective implementation developed by Clavel and Martí-Oliet of the map from linear logic to rewriting logic described in Section 4.6 appears in [Clavel, 2000]. Reflection has been used to endow Maude with a powerful module algebra of parameterized modules and module composition operations implemented in the Full Maude tool [Durán, 1999]. Moreover, reflection allows Maude to become a powerful *metatool* that has itself been used to build formal tools such as an inductive theorem prover; a tool to check the Church-Rosser property, coherence, and termination, and to perform Knuth-Bendix completion; and a tool to specify, analyze and model check real-time specifications [Clavel *et al.*, 2000; Clavel *et al.*, 1999; Olveczky, 2000].

A good number of examples of representations of logics in rewriting logic have been given by different authors, often in the form of executable specifications, including a map $HOL \to Nuprl$ between the logics of the HOL and Nuprl theorem provers, and a natural representation map $PTS \to RWLogic$ of pure type systems (a parametric family of higher-order logics) in rewriting logic [Stehr, 2002].

Thanks to reflection and to the existence of initial models, rewriting logic can not only be used as a logical framework in which the deduction of a logic $\mathcal{L}$ can be faithfully simulated, but also as a *metalogical framework* in which we can reason about the metalogical properties of a logic $\mathcal{L}$. Basin, Clavel, and Meseguer [2000] have begun studying the use of reflection, induction, and Maude's inductive theorem prover enriched with reflective reasoning principles to prove such metalogical properties.

Similarly, the use of rewriting logic and Maude as semantic framework has been greatly advanced. A number of encouraging case studies giving rewriting logic definitions of programming languages have already been carried out by different authors. Since those specifications usually can be executed in a rewriting logic language, they in fact become *interpreters* for the languages in question. In addition, such formal specifications allow both formal reasoning and a variety of formal analyses for the languages so specified. See [Martí-Oliet and Meseguer, 2001] for a considerable number of references related to these topics.

The close connections between rewriting logic and structural operational semantics have been further developed by Mosses [1998] and Braga [2001] in

the context of Mosses's modular structural operational semantics (MSOS) [Mosses, 1999]. In particular, Braga [2001] proves the correctness of a mapping translating MSOS specifications into rewrite theories. Based on these ideas, an interpreter for MSOS specifications [Braga, 2001] and a Maude Action Tool [Braga *et al.*, 2000; Braga, 2001] to execute Action Semantics specifications have been built using Maude.

The implementation of CCS in Maude has been refined and considerably extended to take into account the Hennessy-Milner modal logic by Verdejo and Martí-Oliet [2000]. The semantic properties of this map from CCS to rewriting logic have been studied in detail in [Carabetta *et al.*, 1998; Degano *et al.*, 2000].

ACKNOWLEDGEMENTS

We would like to thank Manuel G. Clavel, Robert Constable, Harmut Ehrig, Fabio Gadducci, Claude Kirchner, Hélène Kirchner, Patrick Lincoln, Ugo Montanari, Natarajan Shankar, Sam Owre, Miguel Palomino, Gordon Plotkin, Axel Poigné, and Carolyn Talcott for their comments and suggestions that have helped us improve the final version of this paper. We are also grateful to the participants of the May 1993 Dagstuhl Seminar on Specification and Semantics, where this work was first presented, for their encouragement of, and constructive comments on, these ideas.

The work reported in this paper has been supported by Office of Naval Research Contracts N00014-90-C-0086, N00014-92-C-0518, N00014-95-C-0225 and N00014-96-C-0114, National Science Foundation Grant CCR-9224005, and by the Information Technology Promotion Agency, Japan, as a part of the Industrial Science and Technology Frontier Program "New Models for Software Architecture" sponsored by NEDO (New Energy and Industrial Technology Development Organization). The first author was also supported by a Postdoctoral Research Fellowship of the Spanish Ministry for Education and Science and by CICYT, TIC 95-0433-C03-01.

SRI International, Menlo Park, CA 94025, USA and
Center for the Study of Language and Information
Stanford University, Stanford, CA 94305, USA

Current affiliations:

Narciso Martí-Oliet
Depto. de Sistemas Informáticos y Programación
Facultad de Ciencias Matemáticas
Universidad Complutense de Madrid
28040 Madrid, Spain
E-mail: narciso@sip.ucm.es

José Meseguer
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Ave
Urbana IL 61801, USA
E-mail: meseguer@cs.uiuc.edu

BIBLIOGRAPHY

[Abadi *et al.*, 1991] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions, *Journal of Functional Programming*, **1**, 375–416, 1991.

[Agha, 1986] G. Agha. *Actors*, The MIT Press, 1986.

[Aitken *et al.*, 1995] W. E. Aitken, R. L. Constable, and J. L. Underwood. *Metalogical frameworks II: Using reflected decision procedures*, Technical report, Computer Science Department, Cornell University, 1995. Also, lecture by R. L. Constable at the Max Planck Institut für Informatik, Saarbrücken, Germany, July 21, 1993.

[Astesiano and Cerioli, 1993] E. Astesiano and M. Cerioli. Relationships between logical frameworks. In *Recent Trends in Data Type Specification*, M. Bidoit and C. Choppy, eds. pp. 126–143. LNCS 655, Springer-Verlag, 1993.

[Bachmair *et al.*, 1992] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In *Proc. 11th. Int. Conf. on Automated Deduction*, Saratoga Springs, NY, June 1992, D. Kapur, ed. pp. 462–476. LNAI 607, Springer-Verlag, 1992.

[Banâtre and Le Métayer, 1990] J.-P. Banâtre and D. Le Mètayer. The Gamma model and its discipline of programming, *Science of Computer Programming*, **15**, 55–77, 1990.

[Barr, 1979] M. Barr. $*$-*Autonomous Categories*, Lecture Notes in Mathematics 752, Springer-Verlag, 1979.

[Barr and Wells, 1985] M. Barr and C. Wells. *Toposes, Triples and Theories*, Springer-Verlag, 1985.

[Basin *et al.*, 2000] D. Basin, M. Clavel, and J. Meseguer. Rewriting logic as a metalogical framework. In *Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India, December 2000, S. Kapoor and S. Prasad, eds. pp. 55–80. LNCS 1974, Springer-Verlag, 2000.

[Basin and Constable, 1993] D. A. Basin and R. L. Constable. Metalogical frameworks. In *Logical Environments*, G. Huet and G. Plotkin, eds. pp. 1–29. Cambridge University Press, 1993.

[Bergstra and Tucker, 1980] J. Bergstra and J. Tucker. Characterization of computable data types by means of a finite equational specification method. In *Proc. 7th. Int. Colloquium on Automata, Languages and Programming*, J. W. de Bakker and J. van Leeuwen, eds. pp. 76–90. LNCS 81, Springer-Verlag, 1980.

[Berry and Boudol, 1992] G. Berry and G. Boudol. The chemical abstract machine, *Theoretical Computer Science*, **96**, 217–248, 1992.

[Borovanský *et al.*, 1996] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Proc. First Int. Workshop on Rewriting Logic and its Applications*, Asilomar, California, September 1996, J. Meseguer, ed. pp. 35–50. ENTCS 4, Elsevier, 1996. http://www.elsevier.nl/locate/entcs/volume4.html

[Bouhoula *et al.*, 2000] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic, *Theoretical Computer Science*, **236**, 35–132, 2000.

[Braga, 2001] C. Braga. *Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics*, Ph.D. thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brazil, 2001.

[Braga *et al.*, 2000] C. Braga, H. Haeusler, J. Meseguer, and P. Mosses. Maude Action Tool: Using reflection to map action semantics to rewriting logic. In *Algebraic Methodology and Software Technology* Iowa City, Iowa, USA, May 2000, T. Rus, ed. pp. 407–421. LNCS 1816, Springer-Verlag, 2000.

[Brüning *et al.*, 1993] S. Brüning, G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Disjunction in plan generation by equational logic programming. In *Beiträge zum 7. Workshop Plannen und Konfigurieren*, A. Horz, ed. pp. 18–26. Arbeitspapiere der GMD 723, 1993.

[Carabetta *et al.*, 1998] G. Carabetta, P. Degano, and F. Gadducci. CCS semantics via proved transition systems and rewriting logic. In *Proc. Second Int. Workshop on Rewriting Logic and its Applications*, Pont-à-Mousson, France, September 1998, C. Kirchner and H. Kirchner, eds. pp. 253–272. ENTCS 15, Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`

[Cerioli and Meseguer, 1996] M. Cerioli and J. Meseguer. May I borrow your logic? (Transporting logical structure along maps), *Theoretical Computer Science*, **173**, 311–347, 1997.

[Clavel, 2000] M. Clavel. *Reflection in Rewriting Logic: Metalogical Foundations and Metaprogramming Applications*, CSLI Publications, 2000.

[Clavel *et al.*, 2001] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic, *Theoretical Computer Science*, 2001. To appear.

[Clavel *et al.*, 2000] M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *Cafe: An Industrial-Strength Algebraic Formal Method*, K. Futatsugi, A. T. Nakagawa, and T. Tamai, eds. pp. 1–31. Elsevier, 2000.

[Clavel *et al.*, 1999] M. Clavel, F. Durán, S. Eker, J. Meseguer, and M.-O. Stehr. Maude as a formal meta-tool. In *FM'99 — Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Volume II*, Toulouse, France, September 1999, J. M. Wing, J. Woodcock, and J. Davies, eds. pp. 1684–1703. LNCS 1709, Springer-Verlag, 1999.

[Clavel *et al.*, 1996] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In *Proc. First Int. Workshop on Rewriting Logic and its Applications*, Asilomar, California, September 1996, J. Meseguer, ed. pp. 65–89. ENTCS 4, Elsevier, 1996. `http://www.elsevier.nl/locate/entcs/volume4.html`

[Clavel and Meseguer, 1996] M. Clavel and J. Meseguer. Axiomatizing reflective logics and languages. In *Proc. Reflection'96*, San Francisco, USA, G. Kiczales, ed. pp. 263–288. April 1996.

[Clavel and Meseguer, 1996a] M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In *Proc. First Int. Workshop on Rewriting Logic and its Applications*, Asilomar, California, September 1996, J. Meseguer, ed. pp. 125–147. ENTCS 4, Elsevier, 1996. `http://www.elsevier.nl/locate/entcs/volume4.html`

[Clavel and Meseguer, 2001] M. Clavel and J. Meseguer. Reflection in conditional rewriting logic, *Theoretical Computer Science*, 2001. To appear.

[Chandy and Misra, 1988] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.

[Cockett and Seely, 1992] J. R. B. Cockett and R. A. G. Seely. Weakly distributive categories. In *Applications of Categories in Computer Science*, M. P. Fourman, P. T. Johnstone, and A. M. Pitts, eds. pp. 45–65. Cambridge University Press, 1992.

[Comon, 1990] H. Comon. Equational formulas in order-sorted algebras. In *Proc. 17th. Int. Colloquium on Automata, Languages and Programming*, Warwick, England, July 1990, M. S. Paterson, ed. pp. 674–688. LNCS 443, Springer-Verlag, 1990.

[Comon, 1991] H. Comon. Disunification: A survey. In *Computational Logic: Essays in Honor of Alan Robinson*, J.-L. Lassez and G. Plotkin, eds. pp. 322–359. The MIT Press, 1991.

[Degano *et al.*, 2000] Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. *A causal semantics for CCS via rewriting logic*. Manuscript, Dipartimento di Informatica, Università di Pisa, submitted for publication, 2000.

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, J. van Leeuwen *et al.*, eds. pp. 243–320. The MIT Press/Elsevier, 1990.

[Durán, 1999] F. Durán. *A Reflective Module Algebra with Applications to the Maude Language*, Ph.D. thesis, Universidad de Málaga, Spain, 1999.

[Ehrig *et al.*, 1991] H. Ehrig, M. Baldamus, and F. Cornelius. Theory of algebraic module specification including behavioural semantics, constraints and aspects of generalized morphisms. In *Proc. Second Int. Conf. on Algebraic Methodology and Software Technology*, Iowa City, Iowa, pp. 101–125. 1991.

[Feferman, 1989] S. Feferman. Finitary inductively presented logics. In *Logic Colloquium'88*, R. Ferro *et al.*, eds. pp. 191–220. North-Holland, 1989.

[Felty and Miller, 1990] A. Felty and D. Miller. Encoding a dependent-type λ-calculus in a logic programming language. In *Proc. 10th. Int. Conf. on Automated Deduction*, Kaiserslautern, Germany, July 1990, M. E. Stickel, ed. pp. 221–235. LNAI 449, Springer-Verlag, 1990.

[Fiadeiro and Sernadas, 1988] J. Fiadeiro and A. Sernadas. Structuring theories on consequence. In *Recent Trends in Data Type Specification*, D. Sannella and A. Tarlecki, eds. pp. 44–72. LNCS 332, Springer-Verlag, 1988.

[Gardner, 1992] P. Gardner. *Representing Logics in Type Theory*, Ph.D. Thesis, Department of Computer Science, University of Edinburgh, 1992.

[Girard, 1987] J.-Y. Girard. Linear logic, *Theoretical Computer Science*, **50**, 1–102, 1987.

[Giunchiglia *et al.*, 1996] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories - Towards an Architecture for Open Mechanized Reasoning Systems. In *Proc. First Int. Workshop on Frontiers of Combining Systems*, F. Baader and K. U. Schulz, eds. pp. 157–174, Kluwer Academic Publishers, 1996.

[Goguen and Burstall, 1984] J. A. Goguen and R. M. Burstall. Introducing institutions. In *Proc. Logics of Programming Workshop*, E. Clarke and D. Kozen, eds. pp. 221–256. LNCS 164, Springer-Verlag, 1984.

[Goguen and Burstall, 1986] J. A. Goguen and R. M. Burstall. A study in the foundations of programming methodology: Specifications, institutions, charters and parchments. In *Proc. Workshop on Category Theory and Computer Programming*, Guildford, UK, September 1985, D. Pitt *et al.*, eds. pp. 313–333. LNCS 240, Springer-Verlag, 1986.

[Goguen and Burstall, 1992] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming, *Journal of the Association for Computing Machinery*, **39**, 95–146, 1992.

[Goguen and Meseguer, 1988] J. A. Goguen and J. Meseguer. Software for the Rewrite Rule Machine. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, Tokyo, Japan, pp. 628–637. ICOT, 1988.

[Goguen and Meseguer, 1992] J. A. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions, and partial operations, *Theoretical Computer Science*, **105**, 217–273, 1992.

[Goguen *et al.*, 1992] J. A. Goguen, A. Stevens, K. Hobley, and H. Hilberdink. 2OBJ: A meta-logical framework based on equational logic, *Philosophical Transactions of the Royal Society, Series A*, **339**, 69–86, 1992.

[Goguen *et al.*, 2000] J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, J. A. Goguen and G. Malcolm, eds. pp. 3–167. Kluwer Academic Publishers, 2000.

[Große *et al.*, 1996] G. Große, S. Hölldobler, and J. Schneeberger. Linear deductive planning, *Journal of Logic and Computation*, **6**, 233–262, 1996.

[Große *et al.*, 1992] G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Equational logic programming, actions, and change. In *Proc. Int. Joint Conf. and Symp. on Logic Programming*, K. Apt, ed. pp. 177–191. The MIT Press, 1992.

[Gunter, 1991] C. Gunter. Forms of semantic specification, *Bulletin of the EATCS*, **45**, 98–113, 1991.

[Harper *et al.*, 1993] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics, *Journal of the Association for Computing Machinery*, **40**, 143–184, 1993.

[Harper *et al.*, 1989] R. Harper, D. Sannella, and A. Tarlecki. Structure and representation in LF. In *Proc. Fourth Annual IEEE Symp. on Logic in Computer Science*, Asilomar, California, pp. 226–237. 1989.

[Harper *et al.*, 1989a] R. Harper, D. Sannella, and A. Tarlecki. Logic representation in LF. In *Category Theory and Computer Science*, Manchester, UK, September 1989, D. H. Pitt *et al.*, eds. pp. 250–272. LNCS 389, Springer-Verlag, 1989.

[Hayes, 1987] P. J. Hayes. What the frame problem is and isn't. In *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*, Z. W. Pylyshyn, ed. pp. 123–137. Ablex Publishing Corp., 1987.

[Hennessy, 1990] M. Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics*, John Wiley and Sons, 1990.

[Henz *et al.*, 1995] M. Henz, G. Smolka, and J. Würtz. Object-oriented concurrent constraint programming in Oz. In *Principles and Practice of Constraint Systems: The Newport Papers*, V. Saraswat and P. van Hentenryck, eds. pp. 29–48. The MIT Press, 1995.

[Hölldobler, 1992] S. Hölldobler. On deductive planning and the frame problem. In *Logic Programming and Automated Reasoning*, St. Petersburg, Russia, July 1992, A. Voronkov, ed. pp. 13–29. LNAI 624, Springer-Verlag, 1992.

[Hölldobler and Schneeberger, 1990] S. Hölldobler and J. Schneeberger. A new deductive approach to planning, *New Generation Computing*, **8**, 225–244, 1990.

[Huet, 1973] G. Huet. A unification algorithm for typed lambda calculus, *Theoretical Computer Science*, **1**, 27–57, 1973.

[Jaffar and Lassez, 1987] J. Jaffar and J. Lassez. Constraint logic programming. In *Proc. 14th. ACM Symp. on Principles of Programming Languages*, Munich, Germany, pp. 111–119. 1987.

[Janlert, 1987] L.-E. Janlert. Modeling change—The frame problem. In *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*, Z. W. Pylyshyn, ed. pp. 1–40. Ablex Publishing Corp., 1987.

[Janson and Haridi, 1991] S. Janson and S. Haridi. Programming paradigms of the Andorra kernel language. In *Proc. 1991 Int. Symp. on Logic Programming*, V. Saraswat and K. Ueda, eds. pp. 167–186. The MIT Press, 1991.

[Jouannaud and Kirchner, 1991] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In *Computational Logic: Essays in Honor of Alan Robinson*, J.-L. Lassez and G. Plotkin, eds. pp. 257–321. The MIT Press, 1991.

[Kahn, 1987] G. Kahn. *Natural semantics*, Technical report 601, INRIA Sophia Antipolis, February 1987.

[Kirchner *et al.*, 1990] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints, *Revue Francaise d'Intelligence Artificielle*, **4**, 9–52, 1990.

[Kirchner *et al.*, 1995] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In *Principles and Practice of Constraint Systems: The Newport Papers*, V. Saraswat and P. van Hentenryck, eds. pp. 133–160. The MIT Press, 1995.

[Laneve and Montanari, 1992] C. Laneve and U. Montanari. Axiomatizing permutation equivalence in the λ-calculus. In *Proc. Third Int. Conf. on Algebraic and Logic Programming*, Volterra, Italy, September 1992, H. Kirchner and G. Levi, eds. pp. 350–363. LNCS 632, Springer-Verlag, 1992.

[Laneve and Montanari, 1996] C. Laneve and U. Montanari. Axiomatizing permutation equivalence, *Mathematical Structures in Computer Science*, **6**, 219–249, 1996.

[Mac Lane, 1971] S. Mac Lane. *Categories for the Working Mathematician*, Springer-Verlag, 1971.

[Martelli and Montanari, 1982] A. Martelli and U. Montanari. An efficient unification algorithm, *ACM Transactions on Programming Languages and Systems*, **4**, 258–282, 1982.

[Martini and Masini, 1993] S. Martini and A. Masini. *A computational interpretation of modal proofs*, Technical report TR-27/93, Dipartimento di Informatica, Università di Pisa, November 1993.

[Martí-Oliet and Meseguer, 1991] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey, *International Journal of Foundations of Computer Science*, **2**, 297–399, 1991.

[Martí-Oliet and Meseguer, 1999] N. Martí-Oliet and J. Meseguer. Action and change in rewriting logic. In *Dynamic Worlds: From the Frame Problem to Knowledge Management*, R. Pareschi and B. Fronhöfer, eds. pp. 1–53. Kluwer Academic Publishers, 1999.

[Martí-Oliet and Meseguer, 2001] N. Martí-Oliet and J. Meseguer. Rewriting logic: Roadmap and bibliography, *Theoretical Computer Science*, 2001. To appear.

[Masini, 1993] A. Masini. *A Proof Theory of Modalities for Computer Science*, Ph.D. Thesis, Dipartimento di Informatica, Università di Pisa, 1993.

[Masseron *et al.*, 1990] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic. In *Foundations of Software Technology and Theoretical Computer Science*, Bangalore, India, December 1990, K. V. Nori and C. E. Veni Madhavan, eds. pp. 63–75. LNCS 472, Springer-Verlag, 1990.

[Masseron *et al.*, 1993] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic I: Actions as proofs, *Theoretical Computer Science*, **113**, 349–370, 1993.

[Matthews *et al.*, 1993] S. Matthews, A. Smaill, and D. Basin. Experience with FS_0 as a framework theory. In *Logical Environments*, G. Huet and G. Plotkin, eds. pp. 61–82. Cambridge University Press, 1993.

[Mayoh, 1985] B. Mayoh. *Galleries and institutions*, Technical report DAIMI PB-191, Computer Science Department, Aarhus University, 1985.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, B. Meltzer and D. Michie, eds. pp. 463–502. Edinburgh University Press, 1969.

[Meseguer, 1989] J. Meseguer. General logics. In *Logic Colloquium'87*, H.-D. Ebbinghaus *et al.*, eds. pp. 275–329. North-Holland, 1989.

[Meseguer, 1990] J. Meseguer. A logical theory of concurrent objects. In *Proc. OOPSLA-ECOOP'90*, N. Meyrowitz, ed. pp. 101–115. ACM Press, 1990.

[Meseguer, 1992] J. Meseguer. Conditional rewriting logic as a unified model of concurrency, *Theoretical Computer Science*, **96**, 73–155, 1992.

[Meseguer, 1992b] J. Meseguer. Multiparadigm logic programming. In *Proc. Third Int. Conf. on Algebraic and Logic Programming*, Volterra, Italy, September 1992, H. Kirchner and G. Levi, eds. pp. 158–200. LNCS 632, Springer-Verlag, 1992.

[Meseguer, 1993] J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In *Research Directions in Object-Based Concurrency*, G. Agha, P. Wegner, and A. Yonezawa, eds. pp. 314–390. The MIT Press, 1993.

[Meseguer, 1993b] J. Meseguer. Solving the inheritance anomaly in concurrent object-oriented programming. In *Proc. ECOOP'93, 7th. European Conf.*, Kaiserslautern, Germany, July 1993, O. M. Nierstrasz, ed. pp. 220–246. LNCS 707, Springer-Verlag, 1993.

[Meseguer, 1996] J. Meseguer. Rewriting logic as a semantic framework for concurrency: A progress report. In *CONCUR'96: Concurrency Theory*, Pisa, Italy, August 1996, U. Montanari and V. Sassone, eds. pp. 331–372. LNCS 1119, Springer-Verlag, 1996.

[Meseguer, 1998] J. Meseguer. Membership algebra as a logical framework for equational specification. In *Recent Trends in Algebraic Development Techniques*, Tarquinia, Italy, June 1997, F. Parisi-Presicce, ed. pp. 18–61. LNCS 1376, Springer-Verlag, 1998.

[Meseguer *et al.*, 1992] J. Meseguer, K. Futatsugi, and T. Winkler. Using rewriting logic to specify, program, integrate, and reuse open concurrent systems of cooperating agents. In *Proc. IMSA'92, Int. Symp. on New Models for Software Architecture*, Tokyo, 1992.

[Meseguer and Goguen, 1993] J. Meseguer and J. A. Goguen. Order-sorted algebra solves the constructor-selector, multiple representation and coercion problems, *Information and Computation*, **104**, 114–158, 1993.

[Meseguer *et al.*, 1989] J. Meseguer, J. A. Goguen, and G. Smolka. Order-sorted unification, *Journal of Symbolic Computation*, **8**, 383–413, 1989.

[Meseguer and Winkler, 1992] J. Meseguer and T. Winkler. Parallel programming in Maude. In *Research Directions in High-Level Parallel Programming Languages*, J. P. Banâtre and D. Le Métayer, eds. pp. 253–293. LNCS 574, Springer-Verlag, 1992.

[Miller, 1991] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification, *Journal of Logic and Computation*, **1**, 497–536, 1991.

[Milner, 1980] R. Milner. *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag, 1980.

[Milner, 1989] R. Milner. *Communication and Concurrency*, Prentice Hall, 1989.

[Milner, 1990] R. Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, J. van Leeuwen *et al.*, eds. pp. 1201–1242. The MIT Press/Elsevier, 1990.

[Milner, 1992] R. Milner. Functions as processes, *Mathematical Structures in Computer Science*, **2**, 119–141, 1992.

[Mosses, 1998] P. D. Mosses. Semantics, modularity, and rewriting logic. In *Proc. Second Int. Workshop on Rewriting Logic and its Applications*, Pont-à-Mousson, France, September 1998, C. Kirchner and H. Kirchner, eds. ENTCS 15, Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`

[Mosses, 1999] P. D. Mosses. *Foundations of modular SOS*, Technical report, Research Series RS-99-54, BRICS, Department of Computer Science, University of Aarhus, 1999.

[Nadathur and Miller, 1988] G. Nadathur and D. Miller. An overview of λProlog. In *Fifth Int. Joint Conf. and Symp. on Logic Programming*, K. Bowen and R. Kowalski, eds. pp. 810–827. The MIT Press, 1988.

[Nielson and Nielson, 1992] H. R. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*, John Wiley and Sons, 1992.

[Nieuwenhuis and Rubio, 1992] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In *Proc. 11th. Int. Conf. on Automated Deduction*, Saratoga Springs, NY, June 1992, D. Kapur, ed. pp. 477–491. LNAI 607, Springer-Verlag, 1992.

[Nipkow, 1993] T. Nipkow. Functional unification of higher-order patterns. In *Proc. Eighth Annual IEEE Symp. on Logic in Computer Science*, Montreal, Canada, pp. 64–74. 1993.

[Olveczky, 2000] P. C. Ölveczky. *Specification and Analysis of Real-Time and Hybrid Systems in Rewriting Logic*, Ph.D. thesis, University of Bergen, Norway, 2000.

[Paulson, 1989] L. Paulson. The foundation of a generic theorem prover, *Journal of Automated Reasoning*, **5**, 363–397, 1989.

[Pfenning, 1989] F. Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Proc. Fourth Annual IEEE Symp. on Logic in Computer Science*, Asilomar, California, pp. 313–322. 1989.

[Plotkin, 1981] G. D. Plotkin. *A structural approach to operational semantics*, Technical report DAIMI FN-19, Computer Science Department, Aarhus University, September 1981.

[Poigné, 1989] A. Poigné. Foundations are rich institutions, but institutions are poor foundations. In *Categorical Methods in Computer Science with Aspects from Topology*, H. Ehrig *et al.* eds. pp. 82–101. LNCS 393, Springer-Verlag, 1989.

[Reichwein *et al.*, 1992] G. Reichwein, J. L. Fiadeiro, and T. Maibaum. Modular reasoning about change in an object-oriented framework, Abstract presented at the *Workshop Logic & Change at GWAI'92*, September 1992.

[Reisig, 1995] W. Reisig. *Petri Nets: An Introduction*, Springer-Verlag, 1985.

[Salibra and Scollo, 1993] A. Salibra and G. Scollo. A soft stairway to institutions. In *Recent Trends in Data Type Specification*, M. Bidoit and C. Choppy, eds. pp. 310–329. LNCS 655, Springer-Verlag, 1993.

[Saraswat, 1992] V. J. Saraswat. *Concurrent Constraint Programming*, The MIT Press, 1992.

[Schmidt-Schauss, 1989] M. Schmidt-Schauss. *Computational Aspects of Order-Sorted Logic with Term Declarations*, LNCS 395, Springer-Verlag, 1989.

[Scott, 1974] D. Scott. Completeness and axiomatizability in many-valued logic. In *Proceedings of the Tarski Symposium*, L. Henkin *et al.* eds. pp. 411–135. American Mathematical Society, 1974.

[Seely, 1989] R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In *Categories in Computer Science and Logic*, Boulder, Colorado, June 1987, J. W. Gray and A. Scedrov, eds. pp. 371–382. Contemporary Mathematics 92, American Mathematical Society, 1989.

[Shapiro, 1989] E. Shapiro. The family of concurrent logic programming languages, *ACM Computing Surveys*, **21**, 412–510, 1989.

[Smolka, 1989] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types*, Ph.D. Thesis, Fachbereich Informatik, Universität Kaiserslautern, 1989.

[Smolka *et al.*, 1989] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-sorted equational computation. In *Resolution of Equations in Algebraic Structures, Volume 2*, M. Nivat and H. Aït-Kaci, eds. pp. 297–367. Academic Press, 1989.

[Smullyan, 1961] R. M. Smullyan. *Theory of Formal Systems*, Annals of Mathematics Studies 47, Princeton University Press, 1961.

[Stehr, 2002] M.-O. Stehr. *Rewriting Logic and Type Theory — From Applications to Unification*, Ph.D. thesis, Computer Science Department, University of Hamburg, Germany, 2002. In preparation.

[Talcott, 1993] C. Talcott. A theory of binding structures and applications to rewriting, *Theoretical Computer Science*, **112**, 99–143, 1993.

[Tarlecki, 1984] A. Tarlecki. Free constructions in algebraic institutions. In *Proc. Mathematical Foundations of Computer Science '84*, M. P. Chytil and V. Koubek, eds. pp. 526–534. LNCS 176, Springer-Verlag, 1984.

[Tarlecki, 1985] A. Tarlecki. On the existence of free models in abstract algebraic institutions, *Theoretical Computer Science*, **37**, 269–304, 1985.

[Troelstra, 1992] A. S. Troelstra. *Lectures on Linear Logic*, CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford University, 1992.

[Verdejo and N. Martí-Oliet, 2000] A. Verdejo and N. Martí-Oliet. Implementing CCS in Maude. In *Formal Methods for Distributed System Development*, Pisa, Italy, October 2000, T. Bolognesi and D. Latella, eds. pp. 351–366. Kluwer Academic Publishers, 2000.

[Walther, 1985] C. Walther. A mechanical solution to Schubert's steamroller by many-sorted resolution, *Artificial Intelligence*, **26**, 217–224, 1985.

[Walther, 1986] C. Walther. A classification of many-sorted unification theories. In *Proc. 8th. Int. Conf. on Automated Deduction*, Oxford, England, 1986, J. H. Siekmann, ed. pp. 525–537. LNCS 230, Springer-Verlag, 1986,

DAVID BASIN, SEÁN MATTHEWS

LOGICAL FRAMEWORKS

1 INTRODUCTION

One way to define a logic is to specify a language and a deductive system. For example, the language of first-order logic consists of the syntactic categories of terms and formulae, and its deductive system establishes which formulae are theorems. Typically we have a specific language in mind for a logic, but some flexibility about the kind of deductive system we use; we are able to select from, e.g., a Hilbert calculus, a sequent calculus, or a natural deduction calculus. A *logical framework* is an abstract characterization of one of these kinds of deductive system that we can use to formalize particular examples. Thus a logical framework for natural deduction should allow us to formalize natural deduction for a wide range of logics from, e.g., propositional logic to intuitionistic type-theories or classical higher-order logic.

Exactly how a logical framework abstractly characterizes a kind of deductive system is difficult to pin-down formally. From a high enough level of abstraction, we can see a deductive system as defining sets; i.e. we have a recursive set corresponding to well-formed syntax, a recursive set corresponding to proofs, and a recursively enumerable set of provable formulae.[1] But this view is really too coarse: we expect a logical framework to be able to capture more than just the sets of well-formed and provable formulae associated with a logic. If this were all that we wanted, then any Turing complete programming language would constitute a logical framework, in so far as it can implement a proof-checker for any logic.

In this chapter we present and examine two different kinds of frameworks, each representing a different view of what a deductive system is: $\rightarrow$-frameworks (deduction interpreted as reasoning in a weak logic of implication) and ID-frameworks (deduction interpreted as showing that a formula is a member of an inductively defined set). Either of these can be used to formalize any recursively enumerable relation. However, before calling a system a logical framework we will demand that it preserves additional structure. Thus we first consider what are the important and distinguishing characteristics of the different kinds of deductive systems, then we examine frameworks based on different sorts of possible *metalogic* (or *metatheory*) and we show that these are well-suited to representing the deductive systems for certain classes of *object logics* (or *object theories*). By providing procedures by which the deductive system of any object logic in a class can be naturally encoded in some metalogic, we show how effective frameworks are for formalizing particular logics.

[1] We make the assumption in this chapter that the property of being a well-formed syntactic entity or a proof is recursive; i.e. we know one when we see it.

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, 89–163.

When we say that an encoding is *natural* we shall mean not only that it is high-level and declarative but also that there is an appropriate bijection between derivations in the object logic and derivations manipulating the encoding in the metalogic. For example, a Hilbert system can be naturally encoded using an inductive definition where each rule in the object logic corresponds to a rule in the metalogic. Similarly, there are natural encodings of many natural deduction systems in a fragment of the language of higher-order logic, which make use of a simple, uniform, method for writing down rules as formulae of the metalogic, so that it is possible to translate between natural deduction proofs of the object logic and derivations in the metalogic.

The term 'logical framework' came into use in the 1980s; however the study of metalogics and methods of representing one logic in another has a longer history in the work of logicians interested in languages for the metatheoretic analysis of logics, and computer scientists seeking conceptual and practical foundations for implementing logic-based systems. Although these motivations differ and are, at least in part, application oriented, behind them we find something common and more general: logical frameworks clarify our ideas of what we mean by a 'deductive system' by reducing it to its abstract principles. Thus work on logical frameworks contributes to the work of Dummett, Gentzen, Hacking, Prawitz, and others on the larger question of what is a logic.

1.1 Some historical background

Historically, the different kinds of deductive systems have resulted from different views of logics and their applications. Thus the idea of a deductive system as an inductive definition developed out of the work of Frege, Russell, Hilbert, Post, Gödel and Gentzen attempting to place mathematics on firm foundational ground. In particular, Hilbert's program (which Gödel famously showed to be impossible) was to use the theory of proofs to establish the consistency of all of classical mathematics, using only finitary methods. From this perspective, of *metatheory as proof theory*, a deductive system defines a set of objects in terms of an initial set and a set of rules that generate new objects from old, and a proof is the tree of rule applications used to show that an object is in the set. For Frege and Hilbert these objects were simply theorems, but later Gentzen took them to be sequents, i.e. pairs of collections of formulae. But, either way, a deductive system defines a recursively enumerable set, which is suitable for analysis using inductive arguments.

How should the metatheory used to define these inductive definitions be characterized? Hilbert required it to be finitary, so it has traditionally been taken to be the primitive recursive fragment of arithmetic (which is essentially, e.g., what Gödel [1931] used for his incompleteness theorems). However, despite the endorsement by Hilbert and Gödel, arithmetic is remarkably unsuitable, in the primitives it offers, as a general theory of inductive definitions. Thus, more recent investigations, such as those of Smullyan [1961] and Feferman [1990], have

proposed theories of inductive definitions based on sets of strings or S-expressions, structures more tractable in actual use.

A different view of deductive systems is found in work in computer science and artificial intelligence, where logics have been formalized for concrete applications like machine checked proof. The work of, e.g., de Bruijn [Nederpelt *et al.*, 1994] does not attempt to analyze the meta-theoretic properties of deductive systems, but concentrates rather on common operations, such as binding and substitution. The goal is to provide an abstract characterization of such operations so that deductive systems can be easily implemented and used to prove theorems; i.e. metatheory is seen as providing a *unifying language* for implementing, and ultimately using, deductive systems. A result of this concern with ease in use (i.e. building proofs) rather than ease of metatheoretic analysis (i.e. reasoning about proofs) is that work has emphasized technically more complex, but more usable, notations such as natural deduction, and resulted in frameworks based on higher-order logics and intuitionistic type-theories instead of the inductive definitions of the older proof theory tradition.

1.2 Focus and organization

This chapter presents the concepts underlying the various logical frameworks that have been proposed, examining the relationship between different kinds of deductive systems and metatheory. Many issues thus fall outside our scope. For example, we do not investigate semantically based approaches to formalizing and reasoning about logics, such as the *institutions* of Goguen and Burstall [1992] or the *general logics* of Meseguer [1989]. Even within our focus, we do not attempt a comprehensive survey of all the formalisms that have been proposed. Neither do we directly consider implementation questions, even though computer implementations now play an important role in the field. Further references are given in the bibliography; the reader is referred in particular to the work of Avron *et al.*, Paulson, Pfenning, Pollack, and ourselves, where descriptions of first-hand experience with logical frameworks can be found.

The remainder of this chapter is organized as follows. In §2 we briefly survey three kinds of deductive systems, highlighting details relevant for formalizing abstractions of them. In §3 we consider a logic based on minimal implication as an abstraction of natural deduction. It turns out that this abstraction is closely related to a generalization of natural deduction due to Schroeder-Heister. In §4 we consider in detail a particular metatheory that formalizes this abstraction. We also consider quantification and so-called 'higher-order syntax'. In §5 we present a case study: the problem of formalizing modal logics. In §6 we examine sequent calculi and their more abstract formalization as consequence relations. In §7 we investigate the relationship between sequent systems and inductive definitions, and present a particular logic for inductive definitions. Finally, we draw conclusions and point to some current and possible future research directions.

2 KINDS OF DEDUCTIVE SYSTEMS

Many different kinds of formalization of deduction have been proposed, for a range of purposes. However in this chapter we are interested in deductive systems designed for the traditional purposes of logicians and philosophers, of investigating the foundations of language and reasoning; in fact we restrict our interest even further, to three kinds of system, which are commonly called *Hilbert calculi*, *sequent calculi* and *natural deduction*. But this restriction is more apparent than real since, between them, these three cover the vast majority of deductive systems that have been proposed. We describe only the details that are important here, i.e. the basic mechanics; for deeper and more general discussion, the reader is referred to Sundholm's articles on systems of deduction elsewhere in this handbook [1983; 1986].

For the purposes of comparison, we shall present, as an example, the same simple logics in each style: propositional logics of minimal and classical implication. The language we will work with then is as follows.

DEFINITION 1. Given a set of atomic propositions P, the language of propositions, L, is defined as the smallest set containing P, where if A and B are in L, then $(A \supset B)$ is in L.

For the sake of readability we assume that $\supset$ associates to the right and omit unnecessary parentheses; for example, we abbreviate $(A \supset (B \supset C))$ as $A \supset B \supset C$. We now proceed to the different presentations.

2.1 Hilbert calculi

Historically, Hilbert calculi are the oldest kind of presentation we consider. They are also, in some technical sense, the simplest. A Hilbert calculus defines a set of *theorems* in terms of a set of *axioms* Ax and a set of *rules of proof* R. A *rule of proof* is a relation between formulae $A_1, \ldots, A_n$ and a formula A. A rule is usually written in a two-dimensional format and sometimes decorated with its name. For example

$$\frac{A_1 \ldots A_n}{A}\ name$$

says that by rule $name$, given $A_1, \ldots, A_n$, it follows that A. The set of formulae defined by a Hilbert calculus is the smallest set containing Ax and closed under R; we call this set a *theory*, and the formulae in it *theorems*.

The set of theorems in a logic of minimal implication can be defined as a Hilbert calculus where the set of axioms contains all instances of the axiom schemata K

$$A \supset B \supset A$$

and S

$$(A \supset B) \supset (A \supset B \supset C) \supset A \supset C\,.$$

We call these schemata because A, B and C stand for arbitrary formulae in L. In addition, we have one rule of proof, *detachment*, which takes the form

$$\frac{A \supset B \quad A}{B}\ Det.$$

The Hilbert calculus defined by K, S and Det is the set of theorems of the minimal (or intuitionistic) logic of implication. We call this theory $\mathrm{HJ}^{\supset}$.[2]

The set of theorems in a logic of classical implication, $\mathrm{HK}^{\supset}$, is slightly larger than $\mathrm{HJ}^{\supset}$. We can formalize $\mathrm{HK}^{\supset}$ by adding to $\mathrm{HJ}^{\supset}$ a third axiom schema, for *Peirce's law*

$$((A \supset B) \supset A) \supset A\,. \tag{1}$$

A proof in a Hilbert calculus consists of a demonstration that a formula is in the (inductively defined) set of theorems. We can think of a proof as either a tree, where the leaves are axioms, and the branches represent rule applications, or as a list of formulae, ending in the theorem, where each entry is either an axiom, or follows from previous entries by a rule. Following common practice, we use the list notation here.

The following is an example of a proof of $A \supset A$ in $\mathrm{HJ}^{\supset}$ and thus also in $\mathrm{HK}^{\supset}$:

1. $A \supset A \supset A$	K
2. $(A \supset A \supset A) \supset (A \supset (A \supset A) \supset A) \supset A \supset A$	S
3. $A \supset (A \supset A) \supset A$	K
4. $(A \supset (A \supset A) \supset A) \supset A \supset A$	Det 2,1
5. $A \supset A$	Det 4,3

It turns out that proving theorems in a Hilbert calculus by hand, or even on a machine, is not practical: proofs can quickly grow to be enormous in size, and it is often necessary (e.g. in the proof we have just presented) to invent instances of axiom schemata that have no intuitive relationship to the formula being proven. However Hilbert calculi were never really intended to be used to build proofs, but rather as a tool for the metatheoretic analysis of logical concepts such as deduction. And from this point of view, the most important fact about Hilbert calculi is that they are essentially inductive definitions;[3] i.e. well-suited for arguments (about provability) by induction.

[2] For the proof systems presented in this section, we follow the tradition where the first letter indicates the kind of deduction system (H for Hilbert, N for natural deduction, and L for sequent calculus), the second letter indicates the logic (J for minimal, or intuitionistic logic, and K for classical logic), and superscripts name the connectives.

[3] The two are so closely related that, e.g., Aczel [1977] identifies them.

2.2 Sequent calculi

Useful though Hilbert calculi are, Gentzen [1934] found them unsatisfactory for his particular purposes, and so developed a very different style that has since become the standard notation for much of proof theory, and which is known as *sequent calculus*.

With a sequent calculus we do not define directly the set of theorems; instead we define a binary 'sequent' relation between collections of formulae, Γ and Δ, and identify a subset of instances of this relation with theorems. We shall write this relation as $\Gamma \vdash \Delta$, where Γ is called the *antecedent* and Δ the *succedent*. This is often read as 'if all the formulae in Γ are true, then at least one of the formulae in Δ is true'.

The rules of a sequent calculus are traditionally divided into two subsets consisting of the *logical* rules, which define logical connectives, and the *structural* rules, which define the abstract properties of the sequent relation itself. The basic properties of any sequent system are given by the following rules which state that $\vdash$ is reflexive for singleton collections ($Basic$) and satisfies a form of transitivity (Cut):

$$\frac{}{A \vdash A}\, Basic \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\, Cut \tag{2}$$

A typical set of structural rules is then

$$\begin{array}{cc} \dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\, WL & \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\, WR \\[2ex] \dfrac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\, CL & \dfrac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\, CR \\[2ex] \dfrac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\, PL & \dfrac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}\, PR \end{array} \tag{3}$$

which define $\vdash$ to be a relation on finite sets that is also monotonic in both arguments (what we will later call *ordinary*). The names WL, CL and PL stand for *Weakening*, *Contraction* and *Permutation Left* of the sequent, while WR, CR and PR name the same operations on the right.

We can give a deductive system for classical implication, which we call $\mathrm{LK}^{\supset}$, in terms of this sequent calculus by adding logical rules for $\supset$:

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \supset B \vdash \Delta, \Delta'}\, \supset\text{-}L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \supset B, \Delta}\, \supset\text{-}R$$

The names $\supset$-L and $\supset$-R indicate that these are rules for introducing the implication connective on the left and the right of $\vdash$. We get a sequent calculus for

minimal implication, $\mathrm{LJ}^{\supset}$, by adding the restriction on $\vdash$ that its succedent is a sequence containing exactly one formula.[4]

Like Hilbert proofs, sequent calculus proofs can be equivalently documented as lists or trees. Since proofs are less unwieldy than for Hilbert calculi, trees are a practical notation. A sequent calculus proof of $A \supset A$ is trivial, so instead we take as an example a proof of Peirce's law (1):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,Basic}{A \vdash B, A}\,WR}{\vdash (A \supset B), A}\,\supset\text{-}R \quad \dfrac{}{A \vdash A}\,Basic}{((A \supset B) \supset A) \vdash A, A}\,\supset\text{-}L}{((A \supset B) \supset A) \vdash A}\,CR}{\vdash ((A \supset B) \supset A) \supset A}\,\supset\text{-}R$$

Notice that it is critical in this proof that the succedent can consist of more than one formula, and that CR can be applied. Neither this proof (since CR is not available), nor any other, of Peirce's law is possible in $\mathrm{LJ}^{\supset}$.

Technically we can regard a sequent calculus as a Hilbert calculus for a binary connective $\vdash$; however the theorems of this system are in the language of sequents over L, not formulae in the language of L itself. The set of theorems a sequent calculus defines is taken to be the set of formulae A in L such that $\vdash A$ is provable, i.e., there is a proof of the sequent $\Gamma \vdash \Delta$, where Γ is empty, and Δ is the singleton A.

2.3 Natural deduction

A second important kind of deductive system that Gentzen [1934] (and subsequently Prawitz [1965]) developed was *natural deduction*. In contrast to the sequent calculus, which is intended as a formalism that supports metatheoretic analysis, natural deduction, as its name implies, is intended to reflect the way people 'naturally' work out logical arguments. Thus Gentzen suggested that, e.g., in order to convince ourselves that the S axiom schema

$$(A \supset B) \supset (A \supset B \supset C) \supset A \supset C$$

is true, we would, reading $\supset$ as 'implies', informally reason as follows: 'Assuming $A \supset B$ and then $A \supset B \supset C$ we have to show that $A \supset C$, and to do this, it is enough to show that C is true assuming A. But if A is true then, from the first assumption, B is true, and, given that A and B are true, by the second assumption C is true. Thus the S schema is true (under the intended interpretation)'.

[4] As a result, in $\mathrm{LJ}^{\supset}$ the structural rules (WR, CR and PR) become inapplicable, and $\Delta = \emptyset$ in the logical rule $\supset$-L.

To represent this style of argument *under assumption* we need a new kind of rule, called a *rule of inference*, that allows temporary hypotheses to be made and discharged. This is best explained by example. Consider implication; we can informally describe some of its properties as: (i) if, assuming A, it follows that B, then it follows that $A \supset B$, and (ii) if $A \supset B$ and A, then it follows that B. We might represent this reasoning diagrammatically as follows:

$$\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \supset B}\ \supset\text{-}I \qquad \dfrac{A \supset B \quad A}{B}\ \supset\text{-}E \tag{4}$$

where $\supset$-I and $\supset$-E are to be pronounced as 'Implication Introduction' and 'Implication Elimination', since they explain how to introduce and to eliminate (i.e. to create and to use) a formula with $\supset$ as the main connective.

We call $A \supset B$ in $\supset$-E the *major premise*, and A the *minor premise*. In general, the major premise of an elimination rule is the premise in which the eliminated connective is exhibited and all other premises are minor premises. Square brackets around assumptions indicate that they are considered to be temporary, and made only for the course of the derivation; when applying the rule we can *discharge* these occurrences. Thus, when applying $\supset$-I, we can discharge (zero or more) occurrences of the assumption A which has been made for the purposes of building a derivation of B, which shows that $A \supset B$. Of course, when applying the $\supset$-I rule, there may be other assumptions (so called open assumptions) that are not discharged by the rule. Similarly, the two premises of $\supset$-E may each have open assumptions, and the conclusion follows under the union of these.

To finish our account of $\supset$, we must explain how these rules can be used to build formal proofs. Gentzen formalized natural deduction derivations just like in sequent and Hilbert calculi, as trees, explaining how formulae are derived from formulae. With natural deduction though, there is an added complication: we also have to track the temporary assumptions that are made and discharged. Thus along with the tree of formulae, a natural deduction derivation has a *discharge function*, which associates with each node N in the tree leaf nodes above N that the rule application at node N discharges. Moreover, there is the proviso that each leaf node can be discharged by at most one node. A proof, then, is a derivation where all the assumptions are discharged (i.e. all assumptions are temporary); the formula proven is a theorem of the logic.

A natural deduction proof, with its discharge function, is a complex object in comparison with a Hilbert or sequent proof. However, it has a simple two-dimensional representation: we just decorate each node with the name of the rule to which it corresponds, and a unique number, and decorate with the same number each leaf node of the tree that it discharges. In this form we can document the previously given informal argument of the truth of the S axiom schema as a formal

proof (we only number the nodes that discharge some formula): [5]

$$\dfrac{\dfrac{\dfrac{\dfrac{[A \supset B \supset C]_2 \quad [A]_3}{B \supset C} \supset\text{-}E \quad \dfrac{[A \supset B]_1 \quad [A]_3}{B} \supset\text{-}E}{C} \supset\text{-}I_3}{A \supset C} \supset\text{-}I_2}{(A \supset B) \supset (A \supset B \supset C) \supset A \supset C} \supset\text{-}I_1$$

Once we have committed ourselves to this formalization of natural deduction, we find that the two rules (4) together define exactly minimal implication, in a deductive system which we call $\mathrm{NJ}^{\supset}$.

While the claims of natural deduction to be the most intuitive way to reason are plausible, the style does have some problems. First, there is the question of how to encode classical implication. We can equally easily give a direct presentation of either classical or minimal implication, using a Hilbert or sequent calculus, but not using natural deduction. While $\mathrm{NJ}^{\supset}$ is standard for minimal implication, there is, unfortunately, no simple equivalent for classical implication (what we might imagine calling $\mathrm{NK}^{\supset}$): the standard presentation of classical implication in natural deduction is as part of a larger, functionally complete, set of connectives, including, e.g., negation and disjunction, through which we can appeal to the law of excluded middle. Alternatively we can simply accept Peirce's law as an axiom. We cannot, however, using the language of natural deduction, define classical implication simply in terms of introduction and elimination rules for the $\supset$ connective.

A second problem concerns proofs themselves, which are complex in comparison to the formal representations of proofs in Hilbert or sequent calculi. It is worth noting that a Hilbert calculus can be seen as a special simpler case of natural deduction, since axioms of a Hilbert calculus can be treated as rules with no premises, and the rules of a Hilbert calculus correspond to natural deduction rules where no assumptions are discharged.

3 NATURAL DEDUCTION AND THE LOGIC OF IMPLICATION

Given the previous remarks about the complexity of formalizations of natural deduction, it might seem unlikely that a satisfactory logical framework for it is possible. In fact, this is not the case. In this section we show that there is an abstraction of natural deduction that is the basis of an effective logical framework. Preliminary to this, however, we consider another way of formalizing natural deduction using sequents.

[5] It may help the reader trying to compare this formal proof diagram with the previous informal argument, to read it 'bottom up'; i.e. upwards from the conclusion at the root.

3.1 Natural deduction and sequents

We can describe natural deduction informally as 'proof under assumption', so the basic facility needed to formalize such a calculus is a mechanism for managing assumptions. Sequents provide this: assumptions can be stored in the antecedent of the sequent and deleted when they are discharged. Thus, if we postulate a relation $\vdash$ satisfying $Basic$ and the structural rules (3), where the succedent is a singleton set, then we can encode $\mathrm{NJ}^{\supset}$ using the rules:[6]

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\,\supset\text{-}I \qquad \frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B}\,\supset\text{-}E \tag{5}$$

This view of natural deduction is often technically convenient (we shall use it ourselves later) but it is unsatisfactory in some respects. Gentzen, and later Prawitz, had a particular idea in mind of how natural deduction proofs should look, and they made a distinction between it and the sequent calculus, using proof trees with discharge functions for natural deduction, and sequent notation for sequent calculus. We also later consider 'natural' generalizations of natural deduction that cannot easily be described using a sequent notation.

3.2 Encoding rules using implication

The standard notation for natural deduction, based on assumptions and their discharge, while intuitive, is formally complex. Reducing it to sequents allows us to formalize presentations in a simpler 'Hilbert' style; however we have also said that this is not altogether satisfactory. We now consider another notation that can encode not only natural deduction but also generalizations that have been independently proposed.

A natural 'horizontal' notation for rules can be based on lists $\langle A_1, \ldots, A_n \rangle$ and an arrow $\rightarrow$. Consider the rules (4): we can write the elimination rule in horizontal form simply as

$$\frac{A \supset B \quad A}{B} \quad \equiv \quad \langle A \supset B, A \rangle \rightarrow B$$

and the introduction rule as

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \supset B} \quad \equiv \quad \langle A^{\dagger} \rightarrow B \rangle \rightarrow A \supset B\,.$$

In the introduction rule we have marked the assumption A with the symbol $\dagger$ to indicate that it is discharged. But this is actually unnecessary; using this linear

[6] Importantly, we can prove that the relation defined by this encoding satisfies the Cut rule above; it thus defines a consequence relation (see §6).

notation, precisely the formulae (here there is only one) occurring on the left-hand side of some $\rightarrow$-connective in the premise list are discharged.

This new notation is suggestive: Gentzen explicitly motivated natural deduction as a formal notation for intuitive reasoning, and we have now mapped natural language connectives such as 'follows from' onto the symbol $\rightarrow$. Why not make this explicit and read $\rightarrow$ as formal implication, and '$\langle \ldots \rangle$' as a conjunction of its components? The result, if we translate into the conventional language of (quantifier free) predicate logic, is just:

$$\begin{array}{c} (T(A \supset B) \mathbin{\&} T(A)) \rightarrow T(B) \\ (T(A) \rightarrow T(B)) \rightarrow T(A \supset B) \end{array} \tag{6}$$

which, reading the unary predicate symbol T as 'True', is practically identical with Gentzen's natural language formulation.

What is the significance of this logical reading of rules? The casual relationship observed above is not sufficient justification for an identification, and we will see that we must be careful. However, after working out the details, this interpretation provides exactly what we need for an effective logical framework, allowing us to trade the complex machinery of trees and discharge functions for a pure 'logical' abstraction.

We separate the investigation of such interpretations into several parts. First, we present a metalogic based on (minimal) implication and conjunction and give a uniform way of translating a set of natural deduction rules $\mathcal{R}$ into a set of formulae $\mathcal{R}^*$ in the metalogic. For an object logic $\mathcal{L}$ presented by a set of natural deduction rules $\mathcal{R}$, this yields a metatheory $\mathcal{L}^*$ given by the metalogic extended with $\mathcal{R}^*$. Second, we demonstrate that for any such $\mathcal{L}$, the translation is *adequate*. This means that $\mathcal{L}^*$ is 'strong enough' to derive (the representative of) any formula derivable in $\mathcal{L}$. Third, we demonstrate that the translation is *faithful*. That is that $\mathcal{L}^*$ is not too strong; we can only derive in $\mathcal{L}^*$ representatives of formulae that are derivable in $\mathcal{L}$, and further, given such a derivation, we can recover a proof in $\mathcal{L}$.[7]

3.3 The metatheory and translation

We have previously defined natural deduction in terms of formula-trees and discharge functions. We now describe the logic with which we propose to replace it.

Thus we assume that $\rightarrow$ and $\&$ have at least the properties they have in minimal logic. Above we have provided three separate formalizations of (the implicational fragment of) minimal logic: as a Hilbert calculus, as a sequent calculus, and as natural deduction. Since these all formalize the same set of theorems, we could base our analysis on any of them. However, it will be convenient for our development

[7] Notice that this is a stronger requirement than the model-theoretic requirement of soundness, which requires that we should not be able to prove anything false in the model, but not that we can recover a proof of anything that we have shown to be true.

to use natural deduction.[8] Even though this may seem circular, it isn't: we simply need some calculus to fix the meaning of these connectives and later to formalize the correctness of our embeddings.

For the rest of this section, we will formalize deductive systems associated with *propositional* logics. We leave the treatment of quantifiers in the object logic, for which we need quantification in the metalogic (here we need only, as we will see, quantifier-free schemata), and a general theory of syntax, to §4. For now, we simply assume a suitable term algebra formalizing the language of the object logic. We will build formulae in the metalogic from the unary predicate T and the connectives $\rightarrow$ and $\&$. To aid readability we assume that $\&$ binds tighter than $\rightarrow$, which (as usual) associates to the right. Now, let $\mathrm{NJ}^{\rightarrow,\&}$ be the natural deduction calculus based on the following rules:

$$\frac{A \quad B}{A \,\&\, B}\,\&\text{-}I \qquad \frac{A \,\&\, B \qquad \begin{array}{c}[A, B]\\ \vdots\\ C\end{array}}{C}\,\&\text{-}E$$

$$\frac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \rightarrow B}\,\rightarrow\text{-}I \qquad \frac{A \rightarrow B \quad A}{B}\,\rightarrow\text{-}E$$

We also formally define the translation, given by the mapping '$*$', from rules of natural deduction to the above language. Here A_i varies over formulae in the logic, and Φ_i over the premises (along with the discharged hypotheses) of a rule Θ.

$$\begin{array}{lcl} \left(\dfrac{\Phi_1 \ldots \Phi_n}{A}\right)^* & \rightsquigarrow & \Phi_1^* \,\&\, \cdots \,\&\, \Phi_n^* \rightarrow T(A) \\ \left(\begin{array}{c}[A_1, \ldots, A_n]\\ \vdots\\ A\end{array}\right)^* & \rightsquigarrow & T(A_1) \,\&\, \cdots \,\&\, T(A_n) \rightarrow T(A) \end{array} \tag{7}$$

Note that axioms and rules that discharge no hypotheses constitute degenerate cases, i.e. $A^* \rightsquigarrow T(A)$. We extend this mapping to sets of rules and formulae, i.e. $\mathcal{R}^* \equiv \{\Theta^* \mid \Theta \in \mathcal{R}\}$.

[8] One reason is that we can then directly relate derivability in the metalogic with derivability in the object logic. As is standard [Prawitz, 1965], derivability refers to natural deduction derivations with possibly open assumptions. Provability is a special case where there are no open assumptions. This generalization is also relevant to showing that we can correctly represent particular consequence relations (see §6).

3.4 Adequacy

If A is a theorem of an object logic $\mathcal{L}$ defined by rules $\mathcal{R}$, then we would like $T(A)$ to be provable in the metatheory $\mathcal{L}^* = \mathrm{NJ}^{\rightarrow,\&} + \mathcal{R}^*$ (i.e. $\mathrm{NJ}^{\rightarrow,\&}$ extended by the rules $\mathcal{R}^*$). More generally, if A is derivable in $\mathcal{L}$ under the assumptions $\Delta = \{A_1, \ldots, A_n\}$, then we would like for $T(A)$ to be derivable in $\mathcal{L}^*$ under the assumptions Δ^*.

Consider, for example, the object logic $\mathcal{L}$ over the language $\{\oplus, \otimes, \times, +\}$ defined by the following rules $\mathcal{R}$:

$$\frac{+}{\otimes}\,\alpha \qquad \frac{+}{\times}\,\beta \qquad \frac{\otimes \quad \times}{\oplus}\,\gamma \qquad \frac{\begin{array}{c}[+]\\ \vdots \\ \oplus\end{array}}{\oplus}\,\delta \tag{8}$$

We can prove, for example, $\oplus$ by:

$$\frac{\dfrac{[+]_1}{\otimes}\,\alpha \qquad \dfrac{[+]_1}{\times}\,\beta}{\dfrac{\oplus}{\oplus}\,\gamma}\,\delta_1 \tag{9}$$

Under our proposed encoding, the rules are translated to the following set $\mathcal{R}^*$:

$$T(+) \rightarrow T(\otimes) \qquad (\alpha^*)$$
$$T(+) \rightarrow T(\times) \qquad (\beta^*)$$
$$T(\otimes)\ \&\ T(\times) \rightarrow T(\oplus) \qquad (\gamma^*)$$
$$(T(+) \rightarrow T(\oplus)) \rightarrow T(\oplus) \qquad (\delta^*)$$

and we can prove $T(\oplus)$ in the metatheory $\mathcal{L}^* = \mathrm{NJ}^{\rightarrow,\&} + \mathcal{R}^*$ by:

$$\frac{\dfrac{}{(T(+) \rightarrow T(\oplus)) \rightarrow T(\oplus)}\,\delta^* \qquad \dfrac{\begin{array}{c}[T(+)]_1\\ \vdots\ \Sigma \\ T(\oplus)\end{array}}{T(+) \rightarrow T(\oplus)}\,{\rightarrow}\text{-}I_1}{T(\oplus)}\,{\rightarrow}\text{-}E$$

where Σ is:

$$\frac{\dfrac{\dfrac{}{T(+) \rightarrow T(\otimes)}\,\alpha^* \quad T(+)}{T(\otimes)}\,{\rightarrow}\text{-}E \qquad \dfrac{\dfrac{}{T(+) \rightarrow T(\times)}\,\beta^* \quad T(+)}{T(\times)}\,{\rightarrow}\text{-}E}{T(\otimes)\ \&\ T(\times)}\,\&\text{-}I$$
$$\frac{\dfrac{}{T(\otimes)\ \&\ T(\times) \rightarrow T(\oplus)}\,\gamma^* \qquad T(\otimes)\ \&\ T(\times)}{T(\oplus)}\,{\rightarrow}\text{-}E$$

Notice how assumptions in $\mathcal{L}$ are modeled directly by assumptions in $\mathcal{L}^*$, and how, in general, a rule application in $\mathcal{L}$ corresponds to a fragment of the derivation in $\mathcal{L}^*$. We have, for instance, the equivalence

$$\begin{array}{c}[+]\\ \vdots\\ \oplus\\ \hline \oplus\end{array}\delta \quad \equiv \quad \dfrac{\dfrac{}{(T(+)\to T(\oplus))\to T(\oplus)}\,\delta^* \qquad \dfrac{\begin{array}{c}[T(+)]_1\\ \vdots\\ T(\oplus)\end{array}}{T(+)\to T(\oplus)}\,{\to}\text{-}I_1}{T(\oplus)}\,{\to}\text{-}E$$

Intuitively, then, we use $\to$-E to unpack the rule, and $\to$-I to gather together the subproofs and discharged hypotheses.

We call the right-hand side of this the corresponding *characteristic fragment* for the rule δ. In the same way there are characteristic fragments for each of the other rules, and out of these we can build a meta-derivation corresponding to any derivation in our original logic. Moreover, these characteristic fragments can be restricted to have a special form. Given a natural deduction rule Θ then Θ^* (in the general case) has the form

$$\begin{aligned}(T(A_{1_1}) \,\&\, \cdots \,\&\, T(A_{1_{m^1}}) \to T(A_1)) \,\&\, \cdots \\ \&\; (T(A_{n_1}) \,\&\, \cdots \,\&\, T(A_{n_{m^n}}) \to T(A_n)) \to T(A)\,,\end{aligned}$$

and we can show:

LEMMA 2. *Given a rule Θ, there is a characteristic fragment Σ where for $1 \leq i \leq n$, atomic assumptions $T(A_{i_1}), \ldots T(A_{i_{m^i}})$ are discharged in subderivations of $T(A_i)$, and then these subderivations are combined together with Θ^* to prove $T(A)$. Further, if we insist also that the major premise of an elimination rule is never the result of the application of an introduction rule (i.e. $\to$-I or $\&$-I), while the minor premise is always either atomic or the result of the application of an introduction rule, then Σ is unique.*

Proof. By an analysis of the possible structure of Θ. ■

For a rule Θ that does not discharge assumptions, i.e., of the form

$$\frac{A_1 \ldots A_n}{A},$$

the characteristic fragment is just a demonstration that, given Θ^*, the rule

$$\frac{T(A_1) \ldots T(A_n)}{T(A)}$$

is derivable in $\mathcal{L}^*$. That is, there is a derivation of $T(A)$ where the only open assumptions are $T(A_1), \ldots, T(A_n)$. Note that this standard notion of derivability (see, e.g., Troelstra [1982] or Hindley and Seldin [1986]), can be extended

(see Schroeder-Heister [1984b] and §3.6 below) to account for rules that discharge assumptions; in this extended sense, each characteristic fragment justifies a derived rule that allows us to simulate $\mathcal{L}$-derivations in $\mathcal{L}^*$.

Given the existence of characteristic fragments, it is a simple matter to model $\mathcal{L}$-derivations in $\mathcal{L}^*$. To begin with, since our metalogic is a natural deduction calculus, we can model assumptions in the encoded logic as assumptions in the metalogic. Each such assumption B_i is modeled by an assumption $T(B_i)$. Now, given a derivation in the object logic, we can inductively replace rules by corresponding characteristic fragments to produce a derivation in $\mathcal{L}^*$. For example, in (9) we proved $\oplus$ using four rule applications; after we gave a proof in $\mathcal{L}^*$ of $T(\oplus)$ built from the four characteristic fragments that correspond to these applications.

In this form, however, this observation assumes not just a metalogic ($\mathrm{NJ}^{\to,\&}$) but also a particular proof calculus for the metalogic (natural deduction), which, since we want a purely logical characterization, we want to avoid. It is easy to remove this assumption by observing that A follows from the assumptions A_1 to A_n in $\mathrm{NJ}^{\to,\&}$ iff $A_1 \,\&\, \cdots \,\&\, A_n \to A$ follows without the assumptions. Thus we have a theorem that states the adequacy of encodings of natural deduction calculi in $\mathrm{NJ}^{\to,\&}$.[9]

THEOREM 3 (Adequacy). *For any natural deduction calculus $\mathcal{L}$ defined by a set of rules $\mathcal{R}$, if the formula A is derivable under assumptions $A_1, \ldots, A_n$, then $T(A_1) \,\&\, \cdots \,\&\, T(A_n) \to T(A)$ is provable in $\mathcal{L}^* = \mathrm{NJ}^{\to,\&} + \mathcal{R}^*$.*

Notice that the proof is based on translation, and hence is constructive: given a derivation in the object logic we can construct one in the metalogic.

3.5 Faithfulness

In proving adequacy, we only used the metatheory $\mathcal{L}^*$ in a limited way where derivations had a simple structure built by pasting characteristic fragments together. Of course, there are other ways to build proofs in $\mathcal{L}^*$ and it could be that this freedom allows us to derive representations of formulae that are not derivable in $\mathcal{L}$. We now show that our translation is faithful, i.e. this is not the case.

To see why we have to be careful about faithfulness, consider the following: in the deductive system $\mathrm{NJ}^{\to,\&}$, the $\to$ connective is minimal implication (the logic is a conservative extension of $\mathrm{NJ}^{\to}$). But what happens if we strengthen the metalogic to be classical, e.g. by adding classical negation or assuming Peirce's law? We can still show adequacy of our encodings, since derivations in $\mathrm{NJ}^{\to,\&}$ remain valid when additional rules are present, but we can also use the encoding to derive formulae that are not derivable in the original system. Consider what happens if we try to prove something that is classically but not minimally valid,

[9] However notice that the translation is only defined on rules without side conditions, i.e. for 'pure' natural deduction; for more idiosyncratic logics, see the discussion in §6.

like, for instance, Peirce's law itself:

$$T(((A \supset B) \supset A) \supset A)\,. \tag{10}$$

Using the axioms in (6), we can reduce this to the problem of proving

$$((T(A) \to T(B)) \to T(A)) \to T(A)\,. \tag{11}$$

But this is itself an instance of Peirce's law, and is provable if $\to$ is classical implication. Thus $T(\cdot)$ does not here define the set of minimal logic theorems.

If $\to$ is read as minimal implication, then the same trick does not work. We can still reduce (10) to (11), but we are not able to take the final step of appealing to Peirce's law in the metalogic.

We now show that assuming $\to$ to be minimal really does lead to a faithful presentation of object logics, by demonstrating a direct relationship between derivations of formulae $T(A)$ in the $\mathcal{L}^*$ and derivations of A in $\mathcal{L}$.

The desired relationship is not a simple isomorphism: we pointed out in discussing adequacy above, that for each natural deduction rule translated into our logical language, it is possible to find a characteristic fragment, and using these fragments we can translate derivations in $\mathcal{L}$ into derivations in $\mathcal{L}^*$. However an arbitrary derivation in $\mathcal{L}^*$ may not have a corresponding derivation in $\mathcal{L}$ (it is a simple exercise to construct a proof of $T(\oplus)$ that does not use characteristic fragments). But if we look more carefully at the derivation of $T(\oplus)$ we have given, and the fragments from which it is constructed, we can see that it has a particularly simple structure, and this structure has a technical characterization, similar to that we used for the characteristic fragments themselves. The derivations we build to show adequacy are in what is called *expanded normal form* (ENF): the major premise of an elimination rule (i.e. $\to$-E or $\&$-E) is never the result of the application of an introduction rule (i.e. $\to$-I or $\&$-I), and all minor premises are either atomic or the result of introduction rules. Not all derivations are in ENF, but any derivation can be transformed into one that is. We have the following:

FACT 4 (Prawitz [1971]). There is an algorithm transforming any derivation in $\mathrm{NJ}^{\to,\&}$ into an equivalent derivation in ENF, and this equivalent derivation is unique.

From this we get the theorem we need; again, like for the statement of adequacy, we abstract away from the deductive system to get a pure logical characterization:

THEOREM 5 (Faithfulness). *For any natural deduction system $\mathcal{L}$ defined by a set of rules $\mathcal{R}$, if we can prove $T(A_1)\ \&\ \cdots\ \&\ T(A_n) \to T(A)$ in $\mathcal{L}^*$, then A is derivable in $\mathcal{L}$ from the assumptions $A_1, \ldots, A_n$.*

Proof. The theorem follows immediately from the existence of ENF derivations and Lemma 6 below. ■

LEMMA 6. *There is an effective transformation from ENF derivations in $\mathcal{L}^*$ of $T(A)$ from atomic assumptions Δ, to natural deduction derivations in $\mathcal{L}$ of A from assumptions $\{B \mid T(B) \in \Delta\}$.*

Proof. We prove this by induction on the structure of ENF derivations. Intuitively, we show that an ENF derivation is built out of characteristic fragments, and thus can easily be translated back into the original deductive system. Consider a derivation Σ of $T(A)$.

Base case: $T(A)$ corresponds to either an assumption in Δ or a (premisless) rule of the encoded theory. The translation then consists either of the assumption A or of the corresponding premisless rule with conclusion A.

Step case: Since the derivation is in ENF and the conclusion is atomic, the last rule applied is an elimination rule; more specifically, the derivation must have the form

$$\dfrac{\dfrac{}{\Phi_1^* \,\&\, \cdots \,\&\, \Phi_n^* \to T(A)}\,\Theta^* \qquad \dfrac{\begin{array}{ccc}\vdots\,\Sigma_1 & & \vdots\,\Sigma_n\\ \Phi_1^* & \cdots & \Phi_n^*\end{array}}{\Phi_1^* \,\&\, \cdots \,\&\, \Phi_n^*}\,\sharp}{T(A)}\,{\to}\text{-}E \tag{12}$$

where $\sharp$ consists only of applications of $\&$-I and Θ^* is

$$\Phi_1^* \,\&\, \cdots \,\&\, \Phi_n^* \to T(A)$$

for some rule Θ of the encoded system.

By the definition of the encoding, each Φ_i^*, derived by a proof Σ_i, is then of the form

$$T(A_{i_1}) \,\&\, \cdots \,\&\, T(A_{i_{m^i}}) \to T(A_i).$$

Since the conclusion of Σ_i proves an implication, and is in ENF, the last rule applied must be $\to$-I; thus Σ_i must have the form

$$\dfrac{\dfrac{[T(A_{i_1}) \,\&\, \cdots \,\&\, T(A_{i_{m^i}})]_1 \qquad \begin{array}{c}[T(A_{i_1})]\ldots[T(A_{i_{m^i}})]\\ \vdots\,\Sigma_i'\\ T(A_i)\end{array}}{T(A_i)}\,\sharp}{T(A_{i_1}) \,\&\, \cdots \,\&\, T(A_{i_{m^i}}) \to T(A_i)}\,{\to}\text{-}I_1$$

with open assumptions Δ, where $\sharp$ consists only of applications of $\&$-E, which 'unpack' the assumption $T(A_{i_1}) \,\&\, \cdots \,\&\, T(A_{i_{m^i}})$ into its component propositions. In other words, (12) corresponds to the characteristic fragment for Θ.

By induction we can apply the same analysis to each Σ_i', taking account not only of the undischarged assumptions Δ, but also the new atomic assumptions $\Delta + \{T(A_{i_1}), \ldots, T(A_{i_{m^i}})\}$ which have been introduced, and we are finished. ■

3.6 Generalizations of natural deduction

The distinction we mentioned above (in §3.1) that Gentzen and Prawitz make between natural deduction and sequent calculi has recently been emphasized by Schroeder-Heister [1984a; 1984b], who has proposed a 'generalized natural deduction'. This extension, which follows directly from Gentzen's appeal to intuition to justify natural deduction, is not formalizable using sequents in the way suggested in §3.[10]

Consider again Gentzen's proposed 'natural language' analysis of logical connectives. In these terms we can characterize Implication Elimination as

> If $A \supset B$ and A, then it follows that B

which we formalize as:

$$\frac{A \supset B \quad A}{B}$$

Now consider the slightly more convoluted

> If $A \supset B$ and assuming that if, given that from A we could derive B, we could derive C, then we can derive C.

which is also true. There is a natural generalization of rules that allows us to express this in the spirit of natural deduction; namely,

$$\frac{A \supset B \qquad \begin{array}{c}\left[\dfrac{A}{B}\right] \\ \vdots \\ C\end{array}}{C} \tag{13}$$

where we can assume rules as hypotheses (and by generalization, have rules as hypotheses of already hypothetical rules, and so on). Schroeder-Heister shows that it is quite a simple matter to extend the formula-tree/discharge-function formalization to cope with this extension: we simply allow discharge functions to point to subtrees rather than just leaves.

Why is such a generalization interesting? Schroeder-Heister uses it to analyze systematically the intuitionistic logical connectives. However it has more general use, and at least one immediate application: we can use it to encode Peirce's law without introducing new connectives, as

$$\begin{array}{c}\left[\dfrac{A}{B}\right] \\ \vdots \\ \dfrac{A}{A}\end{array}$$

[10] Though see Avron [1990] for relevant discussion.

which, in English, is equivalent to

> if, by assuming that some B follows from A, it follows that A, then it follows that A.

This provides, finally, a self-contained generalized natural deduction formulation of classical implication. (At least in the sense that the meaning of implication can be defined by rules involving no auxiliary connectives.)

The process of generalization can obviously be continued beyond allowing rules as hypotheses, though. There is no reason why we cannot also have them as conclusions. Thus, for instance, it is clearly true that

$$\frac{A \supset B}{\left(\dfrac{A}{B} \right)} \to\text{-}E_C$$

(where round brackets do not, in the manner of square brackets, represent the discharge of an assumption, but simply grouping), which might reasonably be read as:

> If $A \supset B$ then from A it follows that B.

By now, however, while our intuition still seems to be functioning soundly, the formula-tree/discharge-function formalization is beginning to break down. It can cope with rules as assumptions, but the more general treatment of both rules and formulae in derivations that we are now proposing is more difficult. With our proposed alternative of the metalogic $\mathrm{NJ}^{\to,\&}$, on the other hand, the same problems do not arise. In fact one way of looking at the faithfulness argument developed above is as essentially a demonstration that this sort of 'confusion' can be unwound for the purpose of recovering a proof in a system that does not in the end allow it. The question then is, how closely does our logic match the traditional formulation?

In the most general case, allowing rules as both hypotheses and conclusions, such a question is not quite meaningful, since we do not have a traditional formulation against which we can compare. However if we limit ourselves to the restricted case where we have rules as assumptions, it is still possible to be properly formal, since we can still compare our encoding to Schroeder-Heister's more traditional formalization in terms of formula-trees and discharge-functions.

Such a formulation is enough to allow us, by a 'natural' generalization of the encoding in (7), to formalize, for instance, (13) as

$$(T(A \supset B) \mathbin{\&} ((T(A) \to T(B)) \to T(C))) \to T(C)$$

and $\to$-E_C as

$$T(A \supset B) \to T(A) \to T(B)\,.$$

It is now possible, by a corresponding generalization of the notion of deduction (see Schroeder-Heister [1984b]) to show adequacy and faithfulness for this larger class of deductive systems.

Generalized natural deduction and curryed rules

Until now we have been using $\mathrm{NJ}^{\rightarrow,\&}$ to encode and reason with deductive systems. If we want to show a direct relationship between the traditional notation of natural deduction and our encoding, then we seem to need both the $\rightarrow$ and $\&$ connectives in the metalogic. However, compare $\rightarrow$-E and $\rightarrow$-E_C; while there seems to be large differences between the two as rules, there is very little either in natural language or in logical framework terms: one is simply the 'curryed' form of the other; in this case the conjunction seems almost redundant.

On the other hand, one generalization that we have not made so far, and one that is suggested by $\mathrm{NJ}^{\rightarrow,\&}$, is to allow a natural deduction rule with multiple conclusions; e.g. assuming an ordinary conjunction $\wedge$, have a (tableau-like) rule

$$\frac{A \wedge B}{A, B}\ \&\text{-}E_{MC}$$

corresponding to the natural language

> If $A \wedge B$ then it follows that A and B.

Both of which correspond to the framework formalization

$$T(A \wedge B) \rightarrow T(A)\ \&\ T(B)\,.$$

However, there seems to be less of a need for rules of this form: a single encoded rule $C \rightarrow A\ \&\ B$ can be replaced by a pair $C \rightarrow A$ and $C \rightarrow B$.

If we are willing to accept this restriction to the language of $\rightarrow$ alone, then we can simplify the metalogic we are using: if we have conjunctions only in the antecedents of implications $\rightarrow$, they can always be eliminated by 'currying' the formulae conjoined in the antecedent, allowing us to dispose of conjunction completely. For example, the curryed form of the axiom for $\rightarrow$-E in (6) is then

$$T(A \supset B) \rightarrow T(A) \rightarrow T(B)\,.$$

Notice that this form of $\rightarrow$-E is indistinguishable from the encoding we give above for $\rightarrow$-E_C: we are no longer able to distinguish some different rules in generalized natural deduction, and thus we lose the faithfulness/adequacy bijection that we have previously demonstrated. However this problem is not serious: we are only identifying proofs that differ in simple ways, e.g., by the application of curryed versus uncurryed rules. Furthermore, this possible confusion arises in the case of *generalized* natural deduction, but not in the traditional form.

In the next section we describe a full logical framework based on the ideas we have developed here. In fact it turns out that we can adopt the same notation to formalize not only deductive systems, but also languages, in a uniform manner.

4 FRAMEWORKS BASED ON TYPE-THEORIES

In the last section we established a relationship between natural deduction and the logic of implication. However we considered only reasoning about fragments of propositional logic, and when we turn to predicate logics, we find that the mechanisms of binding and substitution introduce some entirely new problems for us to solve.

The first problem is simply how to encode languages where operators bind variables. Such variable binding operators include standard logical quantifiers, the λ of the λ-calculus, fixedpoint operators like μ in fixedpoint logics, etc. Until now we have been using a simple term algebra to represent syntax, where, e.g. a binary connective like implication is represented by a binary function. However, with the introduction of binding and substitution this approach is less satisfactory. For instance $\forall x.\, \phi(x)$ and $\forall y.\, \phi(y)$ are distinct syntactically, but not in terms of the deductive system (any proof of one proves the other). Binding operators also complicate operations performed on syntax; e.g. substitution. The second problem is that proof-rules become more complex: the rules for quantifiers place conditions on the contexts (e.g. insisting that certain variables do not appear free) in which they can be applied.

Now we extend our investigation to deal with these problems, and complete the development of a practical $\rightarrow$-framework. We tackle the two problems of language encoding and rule encoding together, by introducing the λ-calculus as a representation language into our system. This provides us with a way to encode quantifiers using *higher-order* syntax and then to encode rules for these quantifiers.

There are different ways that we can combine the λ-calculus with a metalogic. One possibility is simply to add it to the term language, extending, e.g., the theory $\mathrm{NJ}^{\rightarrow}$ to a fragment of higher-order logic.[11] Another, similar, possibility, and one which offers some theoretical advantages, is to use a *type-theory*. We investigate the type-theoretic approach in this section. Type-theories based on the λ-calculus are well-known to be closely related to intuitionistic logics like $\mathrm{NJ}^{\rightarrow}$ via 'Curry-Howard' (see Howard [1980]), or *propositions-as-types*, isomorphisms, a fact which allows us to carry across much of what we already know about encoding deductive systems from $\mathrm{NJ}^{\rightarrow}$. Moreover, an expressive enough type-theory provides a unified language for representing not just syntax, but also proof-rules and proofs. Thus a type-theoretic logical framework can provide a single solution to the apparently distinct problems of encoding languages and deductive systems: The encoding problems are reduced to declaration of appropriate (higher-order) signatures and the checking problems (e.g. well-formedness of syntax and the correctness of derivations) to the problem of type checking against these signatures.

[11] See, e.g., Felty [1989], Paulson [1994] or Simpson [1992], for examples of this approach.

4.1 Some initial observations

We begin by considering the problem of formalizing a binding operator, taking $\forall$ as our example. Consider what happens if we follow through the analysis given above for natural deduction in propositional logic. First we state, in Gentzen style natural language, some of the properties of $\forall$; e.g.

> if, for arbitrary t, it follows that $\phi(t)$, then it follows that $\forall x.\, \phi(x)$

and

> if it follows that $\forall x.\, \phi(x)$, then for any t it follows that $\phi(t)$.

These are traditionally represented, in the notation of natural deduction, as

$$\frac{\phi}{\forall x.\, \phi}\,\forall\text{-}I \qquad \text{and} \qquad \frac{\forall x.\, \phi}{\phi[x \leftarrow t]}\,\forall\text{-}E$$

where, in $\forall$-I, the variable x does not appear free in any undischarged assumption, and the notation $\phi[x \leftarrow t]$ denotes the formula ϕ where the term t has been substituted through for x (care being taken to avoid capturing variables). The relationship between these rules and their informal characterizations is less direct here than in the propositional case; for example, we model the statement 'If, for arbitrary t, it follows that $\phi(t)$' indirectly, by assuming that an arbitrary variable x can stand for an arbitrary term, then ensuring that x really is arbitrary by requiring that it does not occur free in the current assumptions.

Consider how we might use a 'logical' language instead of rules by extending our language based on minimal implication. To start with, we need a way of saying that a term is arbitrary, which we can accomplish with universal quantification. Furthermore, unlike in the propositional case, we have two syntactic categories. As well as formulae (fm), we now have terms (tm), and we will use types to formally distinguish between them. If we combine quantification with typing, by writing $(x{:}y)$ to mean 'for all x of type y', then a first attempt at translating natural language into (semi)formal language results in the following:

$$\begin{aligned} &T(\forall x.\, \phi(x)) \rightarrow (t{:}tm)T(\phi(t)) \\ &(t{:}tm)T(\phi(t)) \rightarrow T(\forall x.\, \phi(x)) \end{aligned} \tag{14}$$

However while this appears to capture our intuitions, it is not clear that it is formally meaningful. If nothing else, one might question the cavalier way we have treated the distinction between object-level and metalevel languages. In the following sections we show that this translation does in fact properly correspond to the intuitive reading we have suggested.

4.2 Syntax as typed terms

We begin by considering how languages that include variable-binding operators can be represented. We want a method of representing syntax where terms in the

object logic are represented by terms in the metalogic. This representation should be computable and we want too that it is compositional; i.e. that the representative of a formula is built directly from the representatives of its subformulae.

Types provide a starting point for solving these problems. A type system can be used to classify terms into types where well-typed terms correspond to well-formed syntax. Consider the example of first-order logic; this has two syntactic categories, terms and formulae, and, independent of whether we view syntax as strings, trees, or something more abstract, we must keep track of the categories to which subexpressions belong. One way to do this is to tag expressions with their categories and have rules for propagating these tags to ensure that entire expressions are 'well-tagged'. Even if there is only one syntactic category we still need some notion of well-formed syntax; in minimal logic, for instance, some strings built from implication and variables, such as $\phi \supset \psi$, are well-formed formulae, while others, like $\phi \supset \supset \psi$, are not.

In a typed setting, we can reduce the problem of syntactic well-formedness to the problem of well-typedness by viewing syntax as a typed term algebra: we associate a type of data with each syntactic category and regard operators over syntax as typed functions. For instance, $\supset$ corresponds to a function that builds a formula given two formulae, i.e., a function of type $\mathit{fm} \times \mathit{fm} \to \mathit{fm}$. Under this reading, and using infix notation, $\phi \supset \psi$ is a well-typed formula provided that ϕ and ψ are both well-typed formulae of type fm, whereas $\phi \supset \supset \psi$ is ill-typed.

This view of syntax as typed terms provides a formalization of the treatment of propositional languages as term algebras that we have informally adopted up to now. We will see that it also provides a basis for formalizing languages with quantification. In fact the mechanism by which this is done is so flexible that it is possible to claim:

THESIS 7. The typed λ-calculus provides a logical basis for representing many common kinds of logical syntax.

Note that we do not claim that the typed λ-calculus, as we shall use it, is a universal solution applicable to formalizing any language. We cannot, for instance, formalize a syntax based on a non-context-free grammar. Neither can we give a finite signature for languages that require infinitely many (or parameterized) sets of productions. The notation is remarkably flexible nevertheless, and, especially if we refine the type system a little further, can deal with a great many subtle problems — a good example of this can be found in the analysis of the syntax of higher-order logic developed by Harper *et al.* [1993].

The simply typed λ-calculus

We assume that the reader is familiar with the basics of the λ-calculus, e.g. reduction (we denote one-step reduction by $\to_\beta$ and its reflexive-transitive closure by $\overset{*}{\to}_\beta$) and conversion ($=_\beta$), and review only syntax and typing here; further details can be found in Barendregt [1984; 1991] or Hindley and Seldin [1986]. We now

define a type theory based on this, called the *simply typed* λ-calculus, or more succinctly $\lambda^{\to}$. Our development is based loosely on that of Barendregt and anticipates extensions we develop later.

We start by defining the syntax we use:

DEFINITION 8. Fix a denumerable set of variables $\mathcal{V}$. The terms and types of $\lambda^{\to}$ are defined relative to a signature consisting of a non-empty set of *base types* $\mathcal{B}$, a set of *constants* $\mathcal{K}$, and a *constant signature* Σ, which is a set $\{\ c_i{:}A_i \mid 1 \leq i \leq n, c_i \in \mathcal{K}, A_i \in \mathrm{Ty}\ \}$, where the c_i are distinct, and:

- The set of *types* Ty is given by

$$\mathrm{Ty} ::= \mathcal{B} \mid \mathrm{Ty} \to \mathrm{Ty}\,.$$

- The set of *terms* $\mathcal{T}$ is given by

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}^{\mathrm{Ty}}.\,\mathcal{T}\,.$$

  A variable x occurs *bound* in a term if it is in the scope of a $\lambda\, x$ and x occurs *free* otherwise. We implicitly identify terms that are identical under a renaming of bound variables.

- A *typing context* is a sequence $x_1{:}A_1, \ldots, x_n{:}A_n$ of *bindings* where the x_i are distinct variables and $A_i \in \mathrm{Ty}$ for $(1 \leq i \leq n)$.

  For convenience, we shall overload the $\in$ relation, extending it from sets to sequences in the obvious manner (i.e. so that $x_i{:}A_i \in x_1{:}A_1, \ldots, x_n{:}A_n$ iff $1 \leq i \leq n$).

- A *type assignment relation* $\vdash_\Sigma$ is a binary relation, indexed by Σ, defined between typing contexts Γ and typing assertions $M{:}A$ where $M \in \mathcal{T}$ and $A \in \mathrm{Ty}$, by the inference rules:

$$\frac{c{:}A \in \Sigma}{\Gamma \vdash_\Sigma c{:}A}\ assum \qquad \frac{\Gamma, x{:}A \vdash_\Sigma M{:}B}{\Gamma \vdash_\Sigma (\lambda x^A.\, M){:}(A \to B)}\ abst$$

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash_\Sigma x{:}A}\ hyp \qquad \frac{\Gamma \vdash_\Sigma M{:}A \to B \quad \Gamma \vdash_\Sigma N{:}A}{\Gamma \vdash_\Sigma (MN){:}B}\ appl$$

This definition states that types consist of the closure of a set of base types under the connective $\to$ and that terms are either variables, constants, applications or (typed) λ-abstractions. The rules constitute a deductive system that defines when a term is well-typed relative to a context and a signature, which in turn assign types to free variables and constants. Notice that we do not have to make explicit the standard side condition for $abst$ that the variable x does not appear free in Γ, since this is already implicitly enforced by the requirement that variables in a type context must be distinct.

As an example, if $\{A, B\} \subseteq \mathrm{Ty}$ then

$$\dfrac{\dfrac{\dfrac{x{:}A \in x{:}A, y{:}B}{x{:}A, y{:}B \vdash_\Sigma x{:}A}\, assum}{x{:}A \vdash_\Sigma \lambda y^B.\, x{:}B \to A}\, abst}{\vdash_\Sigma \lambda x^A.\, \lambda y^B.\, x{:}A \to (B \to A)}\, abst \tag{15}$$

is a proof (for any Σ). Further, it is easy to see that:

FACT 9. Provability for this system (and thus well-typing) is decidable.

A Curry-Howard isomorphism for $\lambda^\to$

The rules for $\lambda^\to$ suggest the natural deduction presentation for $\mathrm{NJ}^\to$ (see §3.1): if we rewrite all instances of $\Gamma \vdash_\Sigma M{:}A$ in a proof in $\lambda^\to$ as $\Gamma, \Sigma \vdash M{:}A$ and uniformly replace each typing assertion $c{:}A$, $x{:}A$, and $M{:}A$ by the type A, then $abst$ and $appl$ correspond to $\to$-I and $\to$-E, while hyp and $assum$ together correspond to $Basic$. The following makes this correspondence more precise.

FACT 10.

1. There is a bijection between types in $\lambda^\to$ and propositions in $\mathrm{NJ}^\to$ where a proposition in $\mathrm{NJ}^\to$ is provable precisely when the corresponding $\lambda^\to$ type is inhabited (has a member).

2. There is a bijection between members of types in $\lambda^\to$ and proofs of the corresponding propositions in $\mathrm{NJ}^\to$.

Part 1 of this characterizes the direct syntactic bijection between types and propositions where base types correspond to atomic propositions [12] and the function space constructor corresponds to the implication connective. The correspondence between provability and inhabitation then follows from the correspondence between the proof-rules of the two systems. Then part 2 refines the bijection to the level of inhabiting terms on the one hand, and proofs on the other.

Fact 10 states an isomorphism between types and propositions that we can characterize as 'truth is inhabitation': if $M{:}A$ then the proposition corresponding to A is provable and, moreover, there is a corresponding notion of reduction in the two settings where β-reduction of M in the λ-calculus corresponds to reduction of the proof A (in the sense of Prawitz).

We exploit this isomorphism in reasoning about encodings in this chapter. In this section we show the correctness of encodings by reasoning about normal forms of terms in type-theory and in §5 we will reason about normal forms of proofs.

[12] This fact holds for the pure version of $\lambda^\to$ without constants. We also implicitly assume a bijection between base types and propositional variables.

REMARK 11. For reasoning about the correctness of encodings it is sometimes necessary to use a more complex isomorphism based on both β and η-conversion. It is possible to establish a correspondence between so called long $\beta\eta$-normal forms[13] and corresponding classes of derivations in $\mathrm{NJ}^{\to}$. More details can be found in [Harper *et al.*, 1993].

Representation

In order to represent syntax in $\lambda^{\to}$ we define a suitable signature Σ and establish a correspondence between terms in the metalogic and syntax in the object logic. Our signature will consist of a base type for each syntactic category of the object logic and a constant for each constructor in the object logic. We will see that we do not need a type *variable* to formalize variables in the object logic because we can use instead variables of the metalogic. Syntax belonging to a given category in the object logic then corresponds to terms in the metalogic belonging to the type of that category.

This is best illustrated with a simple example: we represent the syntax of minimal logic itself. We follow Church's [1940] convention, where o is the type of propositions, so the signature is the (singleton) set of base types $\{o\}$, and the constant signature is

$$\Sigma \equiv \{imp\colon o \to o \to o\}\,. \tag{16}$$

The constructor imp builds a proposition from two propositions. With respect to this signature, $imp\ x\ (imp\ y\ z)$ is a term that belongs to o when x, y, and z belong to o; i.e. $x{:}o, y{:}o, z{:}o \vdash_{\Sigma} imp\ x\ (imp\ y\ z){:}o$. This corresponds to the fact that $x \supset (y \supset z)$ is a proposition of minimal logic provided that x, y, and z are propositions.

Thus formulae of minimal logic correspond to well-typed terms of type o. Formulating this correspondence requires some care though: we have to check adequacy and faithfulness for the represented syntax; this amounts to showing that (i) every formula in minimal logic is represented by a term of type o, and (ii) that every term of type o represents a formula in minimal logic.

As a first attempt to establish such a correspondence, consider the following mapping $\ulcorner\cdot\urcorner$, from terms in minimal logic to λ-calculus terms:

$$\ulcorner x \urcorner = x$$
$$\ulcorner t_1 \supset t_2 \urcorner = imp\ \ulcorner t_1 \urcorner\ \ulcorner t_2 \urcorner$$

As an example, under this representation the formula $x \supset (y \supset z)$ corresponds to the term $imp\ x\ (imp\ y\ z)$. The representation function is an injection from propositions to terms of type o, provided that variables are declared to be of type

[13] A term $M = \lambda x_1 \ldots x_n.\, x M_1 \ldots M_m$ is in long $\beta\eta$-normal form when 1) x is an x_i, a constant, or a free variable of M, 2) $x M_1 \ldots M_n$ is of base type (i.e. x is applied fully to all possible arguments) and 3) each M_i is in long $\beta\eta$-normal form.

o in the context. However, it is not surjective: there are too many terms of type o. For example in a context where x is of type $o \to o$ and y is of type o, then $x\ y$ is of type o, and so is $imp\ ((\lambda z^o.\ z)\ y)\ y$; but neither of these is in the image of $\ulcorner \cdot \urcorner$. The problem with the first example is that the variable x is of higher type (i.e. the function type $o \to o$) and not a base type in $\mathcal{B}$. The problem with the second example is that it is not in normal form. Any such term, however, has a unique equivalent β-normal form, which we can compute. In the above example the term has the normal form $imp\ y\ y$, which is $\ulcorner y \supset y \urcorner$. Our correspondence is thus a bijection when we exclude the cases above: we consider only β-normal form terms where free variables are of type o.

THEOREM 12. *$\ulcorner \cdot \urcorner$ is a bijection between propositions in minimal logic with propositional variables $x_1, \ldots, x_n$, and β-normal form terms of type o containing only free variables $x_1, \ldots, x_n$, all of type o.*

First-order syntax

The syntax of minimal logic is very simple, and we do not need all of $\lambda^{\to}$ to encode it. We can be precise about what we mean by 'not all' if we associate types with orders: observe that any type γ has a unique representation as $\alpha_1 \to \ldots \to \alpha_n \to \beta$, where $\to$ associates to the right and β is a base type. Now define the *order* of γ to be 0 if $n = 0$, and $1 + max(Ord(\alpha_1), \ldots, Ord(\alpha_n))$ otherwise.[14] In our encoding of minimal propositional logic we have used only the first-order fragment of $\lambda^{\to}$; i.e. variables are restricted to the base type o and the only function constant imp is first-order (since its two arguments are of base type); this is another way of saying that an encoding using a simple term algebra is enough.

This raises an obvious question: why adopt a full higher-order notation if a simple first-order notation is enough? Indeed, in a first-order setting, results like Theorem 12 are much easier to prove because there are no complications introduced by reduction and normal forms. The answer is that the situation changes when we introduce quantifiers and other variable binding operators. A 'naive' encoding of syntax is still possible but is much more complicated.

Consider, as an example, the syntax of first-order arithmetic. This is defined in terms of two syntactic categories, terms and formulae, which are usually specified as:

$$\begin{aligned} \text{terms } T &::= x \mid 0 \mid sT \mid T + T \mid T \times T \\ \text{formulae } F &::= T = T \mid \neg F \mid F \wedge F \mid F \vee F \mid F \supset F \mid \forall x.\ F \mid \exists x.\ F \end{aligned}$$

How should we represent this? A possible first-order notation is as follows: we define a base type v of variables, in addition to types i for terms (i.e. 'individuals') and o for formulae. Since variables are also terms, our signature requires a 'coercion' function mapping elements of type v to elements of type i. The

[14] Note that there is not complete agreement in the literature about the order of base types, which is sometimes defined to be 1, not 0.

rest of the signature is formalized as we would expect; e.g. $plus$ is a constant of type $i \to i \to i$, atomic formulae are built from the equality relation eq of type $i \to i \to o$, connectives are defined as propositional functions over o, and a quantifier like $\forall$ is formalized by declaring a function constant *all* of type $v \to o \to o$ taking a variable and a formula to a formula.

This part of the encoding presents no difficulties. However the problems with first-order representations of a language with binding are not directly in the representation, but rather in the way we will use that representation. It is in the formalization of proof-rules where we encounter the problems, in particular with substitution. Consider, for instance, $\forall$-E in (4.1), where we use the notation $\phi[x \leftarrow t]$; we need to formalize an analogue for our encoding, which we can do by introducing a ternary function, sub of type $o \to i \to v \to o$, where $sub(\phi, t, x) = \psi$ is provable precisely when $\phi[x \leftarrow t] = \psi$. With this addition $\forall$-E is axiomatizable as

$$\forall \phi, \psi{:}o.\forall t{:}i.\forall x{:}v.\ (sub(\phi, t, x) = \psi) \to T(all\ x\ \phi) \to T(\psi)\,. \qquad (17)$$

There are several ways we might axiomatize the details of sub. The most direct approach is simply to formalize the standard textbook account of basic concepts such as free and bound variables, capture, and equivalence under bound variable renaming, which we can easily do by structural recursion on terms and formulae. The definitions are well-known, although complex enough that some care is required, e.g. bound variables must sometimes be renamed to avoid capture. Note too that in (17) we have used the equality predicate over formulae (not to be confused with equality over terms in the object logic, i.e. eq), which must either be provided by the metalogic or additionally formalized. Examples of such equational encodings of logics are given by Martí-Oliet and Meseguer [2002].

Other encodings have also been explored: for instance we can use a representation of terms that finesses problems involving bound variable names by eliminating them entirely. Such an approach was originally suggested by de Bruijn [1972] who represents terms with bound variables by replacing occurrences of such variables with 'pointers' to the operators that bind them. A related approach has been proposed by Talcott [1993], who has axiomatized a general theory of binding structure and substitution, which can be used as the basis for encoding logics that require such facilities. Another recent development, which is gaining some popularity, is to provide extensions of the λ-calculus that formalize operators for 'explicit substitutions' [Abadi *et al.*, 1991].

In all of these approaches, direct or indirect, there is considerable overhead, and not only at the formalization stage itself: when we are building proofs, we have to construct subproofs using the axiomatization of sub at every application of $\forall$-E. Being forced to take such micro-steps in the metatheory simply to apply a rule in the object logic is both awkward and tedious. And there is another serious problem that appears when we consider a rule like $\forall$-I in (4.1), where we have a side condition that *the variable x does not appear free in any undischarged*

assumption. There is no direct way, in the language we have been developing, to complete the formalization of this: we cannot reify the semiformal

$$\forall\phi{:}o.\forall x{:}v.\ x \text{ not free in context} \rightarrow T(\phi) \rightarrow T(all\ x\ \phi) \tag{18}$$

into a formal statement (we cannot, in general, refer to contexts in this way).

Again there are ways to get around the problem by complicating the formalization. For instance while we cannot tell which variables are free in the context, we can, under certain circumstances, keep track of those that might be. We can define a new type sv, of sets of variables, with associated algebra and predicates, where, e.g., $notin(x, c)$ is a predicate on types v and sv that is true iff x does not occur in the set c, and $union$ is a function of type $sv \rightarrow sv \rightarrow sv$ that returns the union of its arguments. In this setting we can then expand $T(\cdot)$ so that rather than being a predicate on o, it is a predicate on o and sv; this yields the formalization

$$\forall\phi{:}o.\ \forall x{:}v.\ \forall c{:}sv.\ notin(x, c) \rightarrow T(\phi, c) \rightarrow T(all\ x\ \phi, c)\,.$$

In addition, we have, of course, to modify all the other rules so they keep track of the variables that might be added to the context; for instance $\rightarrow$-I now has the formalization

$$\forall\phi, \psi{:}o.\forall c{:}sv.\ (T(\phi, c) \rightarrow T(\psi, union(c, fv(\phi)))) \rightarrow T(imp\,\phi\,\psi, c)\,,$$

where *fv* returns the set of variables free in a formula (i.e. for ψ we have added all the free variables of ϕ to the free variables in the context).

Clearly, first-order syntax in combination with $\rightarrow$ is becoming unacceptably complicated at this point, and is far removed from the 'sketched' characterization in (14). If part of the motivation of a logical framework is to provide a high-level abstraction of a deductive system, which we can then use to implement particular systems, then, since substitution and binding are standard requirements, we might expect them to be built into the framework, rather than encoded from scratch each time we need them. We thus now examine a very different sort of encoding in terms of $\lambda^{\rightarrow}$ that does precisely this.

Higher-order syntax

When using a first-order encoding of syntax, each time we apply a proof-rule like $\forall$-E, we must construct a subproof about substitution. But in $\lambda^{\rightarrow}$ we already have a substitution mechanism available that we can exploit if we formalize variable binding operators a bit differently. The formalization is based on what is now commonly called *higher-order* syntax.

We start by observing that in $\lambda^{\rightarrow}$ the λ operator binds variables, β-reduction provides the sort of substitution we want, and we have a built-in equivalence that accounts for renaming of bound variables. Higher-order syntax is a way of exploiting this, using higher-order functions to formalize the variable binding operators of

an encoded logic directly, avoiding the complications associated with a first-order encoding.

The idea is best illustrated with an example. If we return to the problem of how to encode the language of arithmetic, then, using higher-order syntax our signature need contain only the two sorts, for terms and formulae; i.e. $\mathcal{B} = \{i, o\}$.

We do not need a sort corresponding to a syntactic category of variables, because now we will represent them directly using variables of the metalogic itself, which are either declared in the context or bound by λ in the metalogic. The signature Σ is then

$$\begin{aligned}\{&0{:}i, s{:}i \to i, \mathit{plus}{:}i \to i \to i, \mathit{times}{:}i \to i \to i, \mathit{eq}{:}i \to i \to o,\\ &\mathit{falsum}{:}o, \mathit{neg}{:}o \to o, \mathit{or}{:}o \to o \to o, \mathit{and}{:}o \to o \to o,\\ &\mathit{imp}{:}o \to o \to o, \mathit{all}{:}(i \to o) \to o, \mathit{exists}{:}(i \to o) \to o\}\,. \end{aligned} \quad (19)$$

In this signature all and exists no longer have the (first-order) type $v \to o \to o$; instead they are second order, taking as their arguments predicate valued functions (which have first-order types).

Using higher-order syntax, an operator that binds a variable can be conceptually decomposed into two parts. First, if ϕ is an encoded formula, i.e. of type o, possibly including a free metavariable x of type i, then $\lambda x^i.\, \phi$ is the abstraction of ϕ over x, where x is now bound. The result is, however, of type $i \to o$, not of type o. Second, we convert $\lambda x^i.\, \phi$ back into an object of type o, and indicate the variable binding operator we want, by applying that operator to it. For example, applying all to $\lambda x^i.\, \phi$ yields the term $\mathit{all}\ (\lambda x^i.\, \phi)$ of type o. Similarly, for a substitution we reverse the procedure. Given $\mathit{all}\ (\lambda x^i.\, \phi)$, for which we want to generate the substitution instance $\phi[x \leftarrow t]$, we first strip off the operator all and apply (in $\lambda^{\to}$) the result to t, to get $(\lambda x^i.\, \phi)t$. But in $\lambda^{\to}$ this reduces to $\phi[x \leftarrow t]$. Hence, we needn't formalize explicitly any substitution mechanism for the object logic since we can exploit the substitution that is (already) formalized for the metalogic.

Of course we must check that all of this works. But it is easy to extend adequacy (Theorem 12) for this signature to show that the terms and formulae of first-order arithmetic are correctly represented by normal form members of types i and o respectively.

Now we can formalize $\forall$-E and $\forall$-I in a way that directly reflects the sketch in (14):

$$\begin{aligned}&\forall \phi^{i \to o}.\, (T(\mathit{all}\ \phi) \to \forall x^i.\, T(\phi\ x))\\ &\forall \phi^{i \to o}.\, ((\forall x^i.\, T(\phi\ x)) \to T(\mathit{all}\ \phi))\end{aligned} \quad (20)$$

If we compare this with (18) above, we can see how, by using metavariables as object variables, we are able to formalize the side condition 'x not free in context' in (20) by having x bound directly by a universal quantifier at the metalevel.

In conclusion, higher-order syntax provides strong supporting evidence for Thesis 7 by providing a mechanism for using the λ of the metatheory to provide directly the machinery needed for variable binding, substitution, variable renaming,

and the like, which are typically needed for representing and using object logics that contain variable binding operators.

4.3 Rules of proof and dependent types

We have shown how we can use a type-theory to represent syntax, reducing the problem of syntactic well-formedness to the problem of decidable well-typing. We now extend the language we are developing so that we can do the same with proof-rules.

We can use $\lambda^{\rightarrow}$ as a representation language for proofs too, but it is too weak to reduce proof checking to type checking alone. To see why, consider the two function symbols $times$ and $plus$ in the signature defined in (19). Both are of type $i \rightarrow i \rightarrow i$, which means that as far as the typing enforced by $\lambda^{\rightarrow}$ is concerned, they are interchangeable; i.e., if t is a well-typed term in $\lambda^{\rightarrow}$, and we replace every occurrence of the constant $times$ with the constant $plus$, we get a well-typed term t'. If t is supposed to represent a piece of syntax, this is what we want; for instance if we have used the type-checking of $\lambda^{\rightarrow}$ to show that $t \equiv eq\ (times\ 0\ (s0))\ 0$ is a well-formed formula, i.e. that $t{:}o$, then we immediately know that t' is a well-formed formula too. Unfortunately, what is useful for encoding syntax makes it impossible to define a type of proofs: in arithmetic we want t, but not t', to be provable, but we cannot make this distinction in $\lambda^{\rightarrow}$: we cannot define a type pr such that $a{:}pr$ iff a is the encoding of a proof, since we would not be able to tell whether a 'proof' is of t or of t'.

This observation may seem at odds with the relationship between $\mathrm{NJ}^{\rightarrow}$ and $\lambda^{\rightarrow}$ established in §3, since we have already used $\mathrm{NJ}^{\rightarrow}$ to encode (propositional) proofs. But in our discussion of the Curry-Howard isomorphism, we were careful to talk about the *propositional fragment* of $\mathrm{NJ}^{\rightarrow}$, but in order to encode even propositional proofs in $\mathrm{NJ}^{\rightarrow}$ we have used the language of *quantifier-free predicate* logic, not propositional logic, and in order to encode the rules for quantifiers, we needed explicit quantification.

To represent proofs we proceed by extending $\lambda^{\rightarrow}$ with *dependent* types, that is, with types that can depend on terms. Specifically, we introduce a new operator, Π, where, if A is a type, and for every $t \in A$, $B[x \leftarrow t]$ is a type, then so is $\Pi x^{A}.\, B$. In other words, we use Π to build families of types, $B[x \leftarrow t]$, indexed by A. Π is sometimes called a dependent function space constructor because its members are functions f where, for every $t \in A$, $f(t)$ belongs to the type $B[x \leftarrow t]$. The addition of dependent types generalizes $\lambda^{\rightarrow}$ since when x does not occur free in B, the type $\Pi x^{A}.\, B$ is simply $A \rightarrow B$ because its members are just the functions from A to B that we have in $\lambda^{\rightarrow}$.

Given dependent function types, we can define the provability relation for a logic as a type-valued function: instead of having pr be a single type, we index it over the formulae it might prove, i.e. we define it to be a function from objects ϕ of type o (i.e. formulae) to the type of proofs of ϕ. Using this, we define typed function constants that correspond to rules of proof. For example, we can now

formalize implication elimination as

$$impe{:}\Pi x^o.\,\Pi y^o.\,pr(imp\ x\ y) \to pr(x) \to pr(y)$$

i.e., $impe$ is a function which, given formulae x and y (objects of type o), and terms proving $x \supset y$ and x, returns a term proving y.

We now provide the formal details of an extension of $\lambda^{\to}$ in which we can build dependent types, and show that the approach to representing deductive systems using type systems actually works the way we want.

The metalogic λ^{p}

The particular theory we present, which we call λ^{p}, is closely related to the Edinburgh LF type-theory [Harper *et al.*, 1993] and the Π fragment of the AUTOMATH language AUT-PI [de Bruijn, 1980]. Our presentation is based on a similar presentation by Barendregt [1991; 1992], which we have chosen for its relative simplicity.

We define the expressions and types of λ^{p} together as follows:

DEFINITION 13 (Pseudo-Terms). Let $\mathcal{V}$ be an infinite set of variables and $\mathcal{K}$ be a set of constants that contains at least two elements, $*$ and $\Box$, which are called *sorts*. A set of *pseudo-terms* $\mathcal{T}$ is described by the following grammar

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}^{\mathcal{T}}.\,\mathcal{T} \mid \Pi\mathcal{V}^{\mathcal{T}}.\,\mathcal{T}$$

Π binds variables exactly like λ. Substitution (respecting bound variables) and bound variable renaming are defined in the standard manner.

DEFINITION 14 (A deductive system for λ^{p}). We define, together, judgments for a *valid signature*, a *valid context* and a *valid typing*. In the following, s ranges over $\{*, \Box\}$:

- A *signature* is a sequence given by the grammar

  $$\Sigma ::= \langle\,\rangle \mid \Sigma, c{:}A$$

  where c ranges over $\mathcal{K}$. A *signature* Σ *is valid* when it satisfies the relation $\vdash_s$ defined by:

  $$\frac{}{\vdash_s \langle\,\rangle} \qquad \frac{\vdash_s \Sigma \quad \vdash_\Sigma A{:}s \quad c \notin dom(\Sigma)}{\vdash_s \Sigma, c{:}A}$$

- A *context* is a sequence given by the grammar

  $$\Gamma ::= \langle\,\rangle \mid \Gamma, x{:}A$$

  where x ranges over $\mathcal{V}$. A *context is valid* with respect to a valid signature Σ when it satisfies the relation $\vdash_c$ defined by:

  $$\frac{}{\vdash_c \langle\,\rangle} \qquad \frac{\vdash_c \Gamma \quad \Gamma \vdash_\Sigma A{:}s \quad x \notin dom(\Gamma)}{\vdash_c \Gamma, x{:}A}$$

- A *type assignment relation* $\vdash_\Sigma$, indexed by a valid signature Σ, is defined between valid typing contexts Γ and typing assertions $a{:}B$ where $a, B \in \mathcal{T}$ is given by the rules:

$$\frac{}{\Gamma \vdash_\Sigma *{:}\Box}\ axiom \qquad \frac{\Gamma \vdash_\Sigma A{:}* \quad \Gamma, x{:}A \vdash_\Sigma B{:}s}{\Gamma \vdash_\Sigma \Pi x^A.\, B{:}s}\ form$$

$$\frac{c{:}A \in \Sigma}{\Gamma \vdash_\Sigma c{:}A}\ assum \qquad \frac{\Gamma, x{:}A \vdash_\Sigma b{:}B \quad \Gamma \vdash_\Sigma \Pi x^A.\, B{:}s}{\Gamma \vdash_\Sigma \lambda x^A.\, b{:}\Pi x^A.\, B}\ abst$$

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash_\Sigma x{:}A}\ hyp \qquad \frac{\Gamma \vdash_\Sigma f{:}\Pi x^A.\, B \quad \Gamma \vdash_\Sigma a{:}A}{\Gamma \vdash_\Sigma f(a){:}B[x \leftarrow a]}\ appl$$

$$\frac{\Gamma \vdash_\Sigma a{:}B \quad \Gamma \vdash_\Sigma B'{:}s \quad B =_\beta B'}{\Gamma \vdash_\Sigma a{:}B'}\ conv$$

We use the two sorts $*$ and $\Box$ to classify entities in $\mathcal{T}$ into levels. We say that $*$ is the set of types and $\Box$ is the set of kinds. As in $\lambda^\to$, terms, which are here a subset of the pseudo-terms, belong to types; unlike in $\lambda^\to$, types, which are here also pseudo-terms, belong to $*$. For example, if o is a type (i.e. $o{:}*$), then $\lambda x^o.\, x$ is a term of type $\Pi x^o.\, o$, which we can abbreviate as $o \to o$, since x does not occur free in o. It is possible to build kinds, in limited ways, using the constant $*$; in particular, the rules we give allow the formation of kinds with range $*$, e.g. $o \to *$ but exclude kinds with domain $*$, e.g. $* \to o$. Hence we can form kinds like $o \to *$ that have type-valued functions as members, but we cannot form kinds by quantifying over the set of types.

We state without proof a number of facts about this system. They have been proven in the more general setting of the so-called λ-cube (a family of eight related type systems) and generalized type systems examined by Barendregt [1991; 1992], but see also Harper *et al.* [1993] who show that the closely related LF type-theory has similar properties.

FACT 15. Term reduction in λ^p is Church-Rosser: given $A, B, B' \in \mathcal{T}$, then if $A \xrightarrow{*}_\beta B$ and $A \xrightarrow{*}_\beta B'$ there exists $C \in \mathcal{T}$ where both $B \xrightarrow{*}_\beta C$ and $B' \xrightarrow{*}_\beta C$.

FACT 16. Terms in λ^p are strongly normalizing: if $\Gamma \vdash_\Sigma A{:}B$, then A and B are strongly normalizing (all β-reductions starting with A or B terminate).

FACT 17. λ^p satisfies unicity of types: if $\Gamma \vdash_\Sigma A{:}B$ and $\Gamma \vdash_\Sigma A{:}B'$, then $B =_\beta B'$.

From the decidability of these operations it follows that:

FACT 18. All judgments of λ^p are decidable.

Relationship to other metalogics

The proof-rules for λ^p extend the rules given for $\lambda^\to$ in Definition 8. The rules for $\lambda^\to$ essentially correspond to the identically named rules for λ^p restricted so

that in every typing assertion $a{:}B$, a is a term of $\lambda^{\to}$ and B is a simple type. The correspondence is not quite exact since in λ^{p} we have to prove that a signature, context, and type are well-formed (i.e., the first three parts of Definition 8), whereas this is assumed to hold in $\lambda^{\to}$. The need for this explicit demonstration of well-formedness is also reflected in the second premise of the λ^{p} rule for abstraction.

An example should clarify the connection between the two systems. In §4.2 we gave a signature Σ for minimal logic

$$\{\, imp{:}\ o \to o \to o \,\}\,.$$

In λ^{p} we have

$$\Sigma = o{:}*, imp{:}o \to o \to o\,.$$

According to this, if x is of type o, then we can show in λ^{p} that $imp\ x\ x$ is a well-formed proposition, i.e., $\Gamma \vdash_{\Sigma} imp\ x\ x{:}o$ where $\Gamma = x{:}o$, as follows:

$$\dfrac{\dfrac{\dfrac{imp{:}o \to o \to o \in \Sigma}{\Gamma \vdash_{\Sigma} imp{:}o \to o \to o}\ assum \quad \dfrac{x{:}o \in \Gamma}{\Gamma \vdash_{\Sigma} x{:}o}\ hyp}{\Gamma \vdash_{\Sigma} imp\ x{:}o \to o}\ appl \quad \dfrac{x{:}o \in \Gamma}{\Gamma \vdash_{\Sigma} x{:}o}\ hyp}{\Gamma \vdash_{\Sigma} imp\ x\ x{:}o}\ appl$$

However, the rules of λ^{p} formalize a strictly more expressive type-theory than $\lambda^{\to}$, and correspond, via a Curry-Howard isomorphism, to a more expressive logic. Terms are built, as we have already seen, by declaring function constants that form typed objects from other typed objects, e.g., $imp\ x\ x$ above corresponds to a term of type o. An n-ary predicate symbol P, which takes arguments of types $s_1, \ldots, s_n$, has the kind $s_1 \to \ldots \to s_n \to *$. The Π-type constructor corresponds either to universal quantification or (in its non-dependent form) implication. For example, given the signature

$$\Sigma = s_1{:}*, s_2{:}*, p{:}s_1 \to s_2 \to *$$

we can show that there is a t such that

$$\vdash_{\Sigma} t{:}(\Pi x^{s_1}.\, \Pi y^{s_2}.\, p(x,y)) \to \Pi y^{s_2}.\, \Pi x^{s_1}.\, p(x,y)\,,$$

which corresponds to demonstrating the provability of the formula

$$\vdash_{\Sigma} (\forall x.\, \forall y.\, p(x,y)) \to \forall y.\, \forall x.\, p(x,y)$$

in a traditional sorted first-order setting.

In λ^{p} we generalize $\lambda^{\to}$ so that types can depend on terms. We have not carried through this generalization to allow, e.g., types depending on types, which would allow impredicative higher-order quantification. As a result, and given the above

discussion, logics like λ^{p} and the LF are often described as first-order. Alternatively, since we can also quantify over functions (as opposed to predicates) at all types, some authors prefer to talk about minimal implicational predicate logic with quantification over all higher types [Simpson, 1992], or ω-order logic (hh^{ω}) [Felty, 1991], to emphasize that these logics are more than first-order, but are not fully higher-order.

4.4 Representation in λ^{p}

We have reached the point where the intuitions we have formulated about the relationship between natural deduction calculi and the logic of implication are reduced to a single formal system, the type-theory λ^{p}. In this system, the problems of encoding the syntax and proof rules of a deductive system are reduced to the single problem of providing a signature Σ and the problems of checking well-formedness of syntax and proof checking are reduced to (decidable) type-checking. We will expand on these points with two examples.

A simple example: minimal logic

A deductive system is encoded in λ^{p} by a signature that encodes

1. The language of the object logic and
2. The deductive system.

In §4.3 we gave a signature suitable for encoding the language of minimal logic. As we have seen, this consists first of an extension of the signature with types corresponding to syntactic categories and then with function constants over these types. The encoding of the deductive system also proceeds in two stages. First, we represent the basic judgments of the object logic.[15] To do this, for each judgment we augment the signature with a function from the relevant syntactic categories to a type. For minimal logic we have one judgment, that a formula is provable, so we add to the signature a function pr, of kind $o \to *$, where for any proposition $p \in o$, $pr(p)$ should be read as saying that the formula represented by p is provable. Second, we add constants to the signature that build (representatives of) proofs. Each constant is associated with a type that encodes (under the propositions-as-types correspondence) a proof rule of the object logic. For minimal logic we add constants with types that encode the formulae given in (6) from §3.2, which axiomatize the rules for minimal logic.

[15] Recall that *judgments* are assertions such as, e.g., that a proposition is provable. Typically, a logic only has a single judgment, but not always; for instance λ^{p} itself, in our presentation, has three judgments: a signature is well-formed, a context is well-formed, and a typing assertion is provable relative to a well-formed signature and context. The reader should be aware of the following possible source of confusion. By using a metalogic we have judgments at two levels: we use the judgment in λ^{p} that a typing assertion is provable relative to a signature and a context to demonstrate the truth of judgments in some object logic.

Putting the above pieces together, minimal logic is formalized by the following signature.

$$\begin{aligned}\Sigma \equiv\ & o{:}*, pr{:}o \to *, imp{:}o \to o \to o,\\ & impi{:}\Pi A^o B^o.\ (pr(A) \to pr(B)) \to pr(imp\ A\ B),\\ & impe{:}\Pi A^o B^o.\ pr(imp\ A\ B) \to pr(A) \to pr(B)\end{aligned}$$

It is an easy exercise to prove in λ^{p} that this is a well-formed signature.

Now consider how we use this signature to prove the proposition $A \supset A$. We encode this as $imp\ A\ A$ and prove it by showing that the judgment $pr(imp\ A\ A)$ has a member. We require, of course, that A is a proposition and we formalize this by the context $A{:}o$, which is well-formed relative to Σ. (In the following proof we have omitted rule names, but these can be easily reconstructed.)
Part I:

$$\dfrac{\dfrac{\dfrac{impi{:}\Pi A^o B^o.\ (pr(A) \to pr(B)) \to pr(imp\,A\,B) \in \Sigma \quad \dfrac{A{:}o \in A{:}o}{A{:}o \vdash_\Sigma A{:}o}}{A{:}o \vdash_\Sigma impi{:}\Pi A^o B^o.\ (pr(A) \to pr(B)) \to pr(imp\,A\,B)} \quad \dfrac{A{:}o \in A{:}o}{A{:}o \vdash_\Sigma A{:}o}}{A{:}o \vdash_\Sigma impi\,A{:}\Pi B^o.\ (pr(A) \to pr(B)) \to pr(imp\,A\,B)} \quad \dfrac{A{:}o \in A{:}o}{A{:}o \vdash_\Sigma A{:}o}}{A{:}o \vdash_\Sigma impi\,A\,A{:}(pr(A) \to pr(A)) \to pr(imp\,A\,A)}$$

Part II:

$$\dfrac{\dfrac{y{:}pr(A) \in A{:}o, y{:}pr(A)}{A{:}o, y{:}pr(A) \vdash_\Sigma y{:}pr(A)} \quad \dfrac{\cdots}{A{:}o \vdash_\Sigma pr(A) \to pr(A){:}*}}{A{:}o \vdash_\Sigma \lambda y^{pr(A)}.\ y{:}pr(A) \to pr(A)}$$

(We have elided the subproof showing that $pr(A) \to pr(A)$ is well-formed, which is straightforward using the formation rule $form$.) Putting the two parts together gives:

$$\dfrac{\text{Part I} \qquad \text{Part II}}{A{:}o \vdash_\Sigma impi\,A\,A\,(\lambda y^{pr(A)}.\ y){:}pr(imp\,A\,A)}$$

Note that the reader interested in actually using a metalogic for machine supported proof construction should not be frightened away by the substantial 'meta-level overhead' that is associated with carrying out a proof of even very simple propositions like $A \supset A$. Real implementations of logical frameworks can hide much of this detail by partially automating the work of proof construction. Because all the judgments of λ^{p} are decidable, the well-formedness of signatures and contexts can be checked automatically, as can the typing of the terms that encode proofs.[16]

[16] This second point is not so important: Although the decidability of syntactic well-formedness is important, in practice, a framework is not used to decide if a given proof is valid, but as an interactive tool for building proofs.

This example shows how both well-formedness and proof checking are uniformly reduced to type checking. With respect to well-formedness of syntax, a proof that the type $pr(imp\,A\,A)$ is inhabited is only possible if $imp\,A\,A$ is of type o, i.e. it represents a well-formed proposition. That members of type o really represent well-formed propositions follows from adequacy and faithfulness of the representation of syntax, which for this example was argued (for $\lambda^{\rightarrow}$) in §4.2. With respect to proof checking, we have proven that the term $impi\,A\,A\,(\lambda y^{\,pr(A)}.\,y)$ inhabits the type $pr(imp\,A\,A)$. In the same way that a term of type o represents a proposition, a term of type $pr(p)$ represents a proof of p. In this example, the term represents the following natural deduction proof of $A \supset A$.

$$\frac{[A]_y}{A \supset A}\,\supset\text{-}I_y$$

The exact correspondence (adequacy and faithfulness) between terms and the proofs that they encode can be formalized (see Harper *et al.* [1993] for details), though we do not do this here since it requires first formalizing natural deduction proof trees and the representation of discharge functions. The idea is simple enough though: the proof rule $\supset$-I is encoded using a constant $impi$, and a well-typed application of this constructs a proof-term formalizing the operation of discharging an assumption. Specifically, $impi$ builds an object (proof representative) of the type (proposition) $pr(imp\ A\ B)$ given an object of type $pr(A) \rightarrow pr(B)$, i.e. a proof that can take any object of proof $pr(A)$ (the hypothesis), and from it produce an object of type $pr(B)$.

In the example above the function must construct a proof of $pr(A)$ from $pr(A)$, and $\lambda y^{pr(A)}.\,y$ does this. In general, the question of which occurrences of $pr(A)$ are discharged and how the proof of B is built is considerably more complex. Consider for example

$$impi\,A\,(imp\,B\,A)\,(\lambda x^{pr(A)}.\,impi\,B\,A\,(\lambda y^{pr(B)}.\,x))\,, \tag{21}$$

which is a member of the type $pr(imp\,A\,(imp\,B\,A))$ in a context where $A{:}o$ and $B{:}o$. This term represents a proof where Implication Introduction has been applied twice and the first (reading left to right) application discharges an assumption x and the second discharge (of y) is vacuous. This proof-term corresponds to the following natural deduction proof.

$$\frac{\dfrac{[A]_x}{B \supset A}\,\supset\text{-}I_y}{A \supset (B \supset A)}\,\supset\text{-}I_x \tag{22}$$

A larger example: first-order arithmetic

A more complex example of a theory that we can easily formalize in λ^{p} is first-order arithmetic, and in fact we can define this as a direct extension of the system

$$
\begin{aligned}
\mathit{oril}&{:}\Pi A^o B^o.\, pr(A) \to pr(\mathit{or}\ A\ B)\\
\mathit{orir}&{:}\Pi A^o B^o.\, pr(B) \to pr(\mathit{or}\ A\ B)\\
\mathit{ore}&{:}\Pi A^o B^o C^o.\, pr(\mathit{or}\ A\ B) \to (pr(A) \to pr(C))\\
&\qquad\qquad \to (pr(B) \to pr(C)) \to pr(C)\\
\mathit{raa}&{:}\Pi A^o.\, (pr(\mathit{imp}\ A\ \mathit{falsum}) \to pr(\mathit{falsum})) \to pr(A)\\
\mathit{alli}&{:}\Pi A^{i\to o}.\, (\Pi x^i.\, pr(A(x))) \to pr(\mathit{all}(A))\\
\mathit{alle}&{:}\Pi A^{i\to o} x^i.\, pr(\mathit{all}(A)) \to pr(A(x))\\
\mathit{existsi}&{:}\Pi A^{i\to o}.\, \Pi x^i.\, pr(A(x)) \to pr(\mathit{exists}(A))\\
\mathit{existse}&{:}\Pi A^{i\to o}.\, \Pi C^o.\, pr(\mathit{exists}(A)) \to (\Pi x^i.\, pr(A(x)) \to pr(C)) \to pr(C)\\
\mathit{ind}&{:}\Pi A^{i\to o}.\, pr(A(0)) \to (\Pi x^i.\, pr(A(x)) \to pr(A(sx))) \to pr(\mathit{all}(A))
\end{aligned}
$$

Figure 1. Some proof-rules for arithmetic

we have already formalized. We extend the signature with the formalization of the syntax of arithmetic that we developed in §4.2 then we formalize the new rules, axioms and axiom-schemas that we need.

We have formalized some of the proof-rules in Figure 1 and most are self-explanatory. The first five extend minimal logic to propositional logic by adding rules for disjunction and falsum. We use the constant falsum to encode $\bot$ (from which we can define negation as $\mathit{not}\ A \equiv \mathit{imp}\ A\ \mathit{falsum}$).

In our rules we assume a 'classical' falsum, i.e. the rule:

$$\dfrac{\begin{array}{c}[A \supset \bot]\\ \vdots\\ \bot\end{array}}{A}\ \bot_c$$

encoded as raa. If we wanted an 'intuitionistic' falsum, we would replace this with the simpler rule encoded by

$$\Pi A^o.\, pr(\mathit{falsum}) \to pr(A).$$

For the quantifier rules we have not only given the rules for universal quantification (alli and alle) but also the rules for existential quantification, given by $\mathit{existsi}$ and $\mathit{existse}$.

$$\dfrac{A[x \leftarrow t]}{\exists x.\, A}\ \exists\text{-}I \qquad \dfrac{\exists x.\, A \qquad \begin{array}{c}[A]\\ \vdots\\ C\end{array}}{C}\ \exists\text{-}E$$

These come with the usual side conditions: in $\forall$-I, x cannot be free in any undischarged assumptions on which A depends and, for $\exists$-E, x cannot be free in C or any assumptions other than A upon which (in the subderivation) C depends.

Object Logic		**Metalogic**
Syntactic Categories terms, individuals	$\leadsto$	Base Types $\{i{:}*, o{:}*\}$
Connectives & Constructors $\vee$	$\leadsto$	First-Order Constants $or{:}o \to o \to o$
Variable Binding Operators $\forall$	$\leadsto$	Higher-Order Constants $all{:}(i \to o) \to o$
Judgment $\vdash p$	$\leadsto$	Type Valued Functions $pr{:}o \to *$
Inference Rule $\dfrac{A}{A \vee B}\ \vee\text{-}IL$	$\leadsto$	Constant Declaration $oril{:}\Pi A^o B^o.\, pr(A) \to pr(or\ A\ B)$
Deductive System		Signature Declaration
Deduction		Typing Proof

Figure 2. Correspondence between object logics and their encodings

If we stop with the quantifier rules, the result is an encoding of first-order logic over the language of arithmetic. We have to add more rules to formalize the theory of equality and arithmetic. Thus, for example, ind formalizes the induction rule

$$\frac{A[x \leftarrow 0] \qquad \begin{array}{c}[A]\\ \vdots\\ A[x \leftarrow sx]\end{array}}{\forall x.\, A}$$

and enforces the side condition that x does not occur free in any assumptions other than those discharged by the application of the rule. The other rules of arithmetic are formalized in a similar fashion.

4.5 Summary

Figure 2 contains a summary. It is worth emphasizing that there is a relationship between the metalogic and the way that it is used, and an $\to$-framework like λ^{p} is well-suited to particular kinds of encodings. The idea behind higher-order syntax and the formalization of judgments using types is to internalize within the metalogic as much of the structure of terms and proofs as possible. By this we mean that syntactic notions and operations are subsumed by operations provided by the framework logic. In the case of syntax, we have seen how variable binding in the object logic is implemented by λ-abstraction in the framework logic and how substitution is implemented by β-reduction. Similarly, when representing proof-

rules and proof-terms, rather than our having to formalize, and then reason explicitly about, assumptions and their discharging, this is also captured directly in the metalogic. Support for this sort of internalization is one of the principles behind the design of these framework logics. The alternative (also possible in λ^{P}) is to *externalize*, i.e., explicitly represent, such entities. The external approach is taken when using frameworks based on inductive definitions, which we will consider in §7.

5 ENCODING LESS WELL BEHAVED LOGICS

So far, we have restricted our attention to fairly standard, e.g. intuitionistic or classical, logics. We now consider how an $\rightarrow$-framework can treat the more 'unconventional' logics that we encounter in, for example, philosophy or artificial intelligence, for which such simple calculi are not available. As previously observed, most metalogics (and all the examples examined in this chapter) are 'universal' in the sense that they can represent any recursively enumerable relation, and thus any logic expressible in terms of such relations. However, there is still the question of how effective and natural the resulting encodings are.

We take as our example one of the more common kinds of philosophical logics: modal logic, i.e. propositional logic extended with the unary $\Box$ connective and the *necessitation* rule (see Bull and Segerberg [1984]). Modal logics, as a group, have common features; for example, 'canonical' presentations use Hilbert calculi and, when natural deduction presentations are known, the proof-rules typically are not encodable in terms of the straightforward translation presented in §3.2. In §7 we will see how Hilbert presentations of these logics can be directly encoded as inductive definitions. Here we consider the problem of developing natural deduction presentations in an $\rightarrow$-framework. We explore two different possibilities, *labelled deductive systems* and *multiple judgment systems*, consider how practical they are, and how they compare.

5.1 Modal logic

We consider two modal logics in this section: K and an important extension, $\mathrm{S4}$. A standard presentation of K is as a Hilbert system given by the axiom schemata

$$(A \supset B) \supset (A \supset B \supset C) \supset A \supset C$$
$$A \supset B \supset A$$
$$((A \supset \bot) \supset \bot) \supset A$$
$$\Box(A \supset B) \supset \Box A \supset \Box B$$

and the rules

$$\frac{A \supset B \quad A}{B}\,Det \qquad \text{and} \qquad \frac{A}{\Box A}\,Nec$$

The first of these rules is just the rule of detachment from §2.1, and the second is called necessitation. We get S4 from K by adding the additional axiom schemata:

$$\Box A \supset \Box\Box A$$
$$\Box A \supset A$$

As noted above, we can see a Hilbert calculus as a special case of a natural deduction calculus, where the rules discharge no assumptions and axioms are premisless rules.

There is an important difference between Hilbert and natural deduction calculi however, which is in the nature of what they reason about: Hilbert calculi manipulate formulae that are true in all contexts, i.e. valid (theorems), in contrast to natural deduction calculi, which typically manipulate formulae that are true under assumption. This difference causes problems when we try to give natural deduction-like presentations of modal logics, i.e. presentations that allow reasoning under temporary assumptions. The problem can be easily summarized:

PROPOSITION 19. *The deduction theorem (see §7.3) fails for* K *and* S4.

Proof. First, observe (e.g., semantically using Kripke structures; see §5.2) that $A \supset \Box A$ is not provable in K or S4. However, if the deduction theorem held, we could derive this formula as follows: assume A, then, by necessitation, we have $\Box A$, and by the deduction theorem we would have that $A \supset \Box A$ is a theorem. This is a contradiction. ■

The deduction theorem is a justification for the natural deduction rule $\supset$-I, but this in turn is precisely the rule that distinguishes natural deduction-like from Hilbert calculi: without it, one collapses into the other.

The problem of natural deduction encodings of modal logics is well known, and various fixes have been proposed. In some of these, the rules $\supset$-I and $\supset$-E are kept intact by extending the language of natural deduction itself. For instance if we allow global side conditions on rules then (following Prawitz [1965]) for S4 we have the rules

$$\frac{\begin{array}{c}\vdots\ *\\ A\end{array}}{\Box A}\ \Box\text{-}I \qquad \text{and} \qquad \frac{\Box A}{A}\ \Box\text{-}E$$

where $*$ means that *all undischarged assumptions are boxed; i.e. of the form* $\Box B$. Notice that given this side condition, the argument we have used to illustrate the failure of the deduction theorem no longer works. But the language of $\rightarrow$ does not provide the vocabulary to express this side condition on $\Box$-I, so we cannot encode such a proof rule in the same fashion as proof-rules were encoded in §3.2.

5.2 A Kripke semantics for modal logics

A common way of understanding the meaning of formulae in a modal logic is in terms of the *Kripke*, or *possible worlds* semantics (see Kripke [1963] or van Benthem [1984] for details). We shall use this style of interpretation in developing our encodings.

A Kripke model (W, R, V) for a modal logic consists of a nonempty set of *worlds* W, a binary *accessibility* relation R defined over W, and a *valuation* predicate V over W and the propositional variables. We then define a *forcing* relation $\Vdash$ between worlds and formulae as follows: $a \Vdash A$ iff $V(a, A)$ for A atomic; $a \Vdash A \supset B$ iff $a \Vdash A$ implies $a \Vdash B$; and $a \Vdash \Box A$ iff for all $b \in W$ if $a\ R\ b$ then $b \Vdash A$.

Using the Kripke semantics, we can classify modal logics by the behavior of R alone. For instance we have

FACT 20. Let R be the accessibility relation of a Kripke model.

- A formula A is a theorem of K iff A is forced at all worlds of all Kripke models.
- A formula A is a theorem of $\mathrm{S4}$ iff A is forced at all worlds of all Kripke models where R is reflexive ($x\ R\ x$) and transitive (if $x\ R\ y$ and $y\ R\ z$, then $x\ R\ z$).

It is now possible to see why the deduction theorem fails. Consider a Kripke model (W, R, V) and a formula $A \supset B$. In the deduction theorem we assume, for the sake of argument, A as a new axiom, and show that B is then a theorem; i.e. assuming $\forall a \in W.\, a \Vdash A$, we show that $\forall a \in W.\, a \Vdash B$. But it does not follow from this that $A \supset B$ is a theorem; i.e. that $\forall a \in W.\, a \Vdash A \supset B$.

It is however easy to find a correct 'semantic' analogue of the deduction theorem:

FACT 21. For any Kripke model (W, R, V)

$$\forall a \in W.\, (a \Vdash A \to a \Vdash B) \to a \Vdash A \supset B\,.$$

The problem of providing a natural deduction encoding of a modal logic can be reduced to the problem of capturing this semantic property of $\supset$ in rules that can be directly encoded in the language of implication. We will consider two ways of doing this, which differ in the extent to which they make the Kripke semantics explicit.

5.3 Labelled deductive systems[17]

The above analysis suggests one possible solution to our problem: we can internalize the semantics into the deductive calculus. Hence, instead of reasoning with

[17] The work described in this section was done in collaboration with Luca Viganò.

formulae, we reason about formulae in worlds; i.e. we work with pairs $a{:}A$ where a is a world and A is a formula.

Taking this approach, the rules

$$\frac{\begin{array}{c}[x{:}A]\\ \vdots\\ x{:}B\end{array}}{x{:}A \supset B}\supset\text{-}I \qquad \frac{\begin{array}{c}[x\ R\ y]\\ \vdots\\ y{:}A\end{array}}{x{:}\Box A}\Box\text{-}I$$

$$\frac{\begin{array}{c}[x{:}A \supset \bot]\\ \vdots\\ y{:}\bot\end{array}}{x{:}A}\bot\text{-}E \qquad \frac{x{:}A \supset B \quad x{:}A}{x{:}B}\supset\text{-}E \qquad \frac{x{:}\Box A \quad x\ R\ y}{y{:}A}\Box\text{-}E$$

define a natural deduction calculus, which we call K_L. (We also require the side conditions that in $\Box$-I y is different from x and does not occur in the assumptions on which $y{:}A$ depends, except those of the form $x\ R\ y$ that are discharged by the inference.) These rules formalize the meaning of both $\supset$ and $\Box$ in terms of the Kripke semantics; i.e., we locate applications of $\supset$-I in some particular world, and take account of the other worlds in defining the behavior of $\Box$ and $\bot$ (where it suffices to derive a contradiction in any world). We can show:

FACT 22 (Basin *et al.* [1997a]). $a{:}A$ is provable in K_L iff A is true in all Kripke models, and therefore, by the completeness of K with respect to the set of all Kripke models, iff A is a theorem of K.

As an example of a proof in K_L of a K theorem, we show that $\Box$ distributes over $\supset$.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[a{:}\Box(A \supset B)]_1 \quad [a\ R\ b]_3}{b{:}A \supset B}\Box\text{-}E \quad \cfrac{[a{:}\Box A]_2 \quad [a\ R\ b]_3}{b{:}A}\Box\text{-}E}{b{:}B}\supset\text{-}E}{a{:}\Box B}\Box\text{-}I_3}{a{:}\Box A \supset \Box B}\supset\text{-}I_2}{a{:}\Box(A \supset B) \supset \Box A \supset \Box B}\supset\text{-}I_1$$

Further, and essential for our purpose, there are no new kinds of side conditions on the rules of K_L, so we have no difficulty in formalizing these in an $\rightarrow$-framework.[18] The following is a signature for K_L in λ^{p} (note that for the sake of

[18] There is of course the side condition on $\Box$-I. But this can be formalized in the same way that eigenvariable conditions are formalized in logics with quantifiers, by using universal quantification in the metalogic.

readability we write '$:$' and R in infix form):

$$\begin{aligned}\Sigma_{\mathrm{K}_L} \equiv\ & w{:}*,\ o{:}*,\ \text{`}:\text{'}{:}w \to o \to *,\ R{:}w \to w \to *,\\ & \mathit{falsum}{:}o,\ \mathit{imp}{:}o \to o \to o,\ \mathit{box}{:}o \to o,\\ & \mathit{FalseE}{:}\Pi A^o x^w y^w.\ (x{:}\mathit{imp}\ A\ \mathit{falsum} \to y{:}\mathit{falsum}) \to x{:}A,\\ & \mathit{impI}{:}\Pi A^o B^o x^w.\ (x{:}A \to x{:}B) \to x{:}\mathit{imp}\ A\ B,\\ & \mathit{impE}{:}\Pi A^o B^o x^w.\ x{:}A \to x{:}\mathit{imp}\ A\ B \to x{:}B,\\ & \mathit{boxI}{:}\Pi A^o x^w.\ (\Pi y^w.\ x\ R\ y \to y{:}A) \to x{:}\mathit{box}\ A,\\ & \mathit{boxE}{:}\Pi A^o x^w y^w.\ x{:}\mathit{box}\ A \to x\ R\ y \to y{:}A\end{aligned}$$

The signature reflects that there are two types, a type of worlds w and type of formulae o, and two judgments; one about the relationship between worlds and formulae, asserting that a formula is true at that world, and a second, between two worlds, asserting that the first accesses the second. Adequacy and faithfulness follow by the style of analysis given in §3.

K_L *as a base for other modal logics*

We can now take K_L as a base upon which to formalize other modal logics. Since modal logics are characterized, in terms of Kripke models, purely in terms of their accessibility relations, to get other modal logics we must simply modify the behavior of R in our encoding. Thus, since $\mathrm{S4}$ corresponds to the class of Kripke models with transitive and reflexive accessibility relations, we can enrich our signature with:

$$\begin{aligned}&\mathit{Ref}{:}\Pi x^w.\ x\ R\ x\\ &\mathit{Trans}{:}\Pi x^w y^w z^w.\ x\ R\ y \to y\ R\ z \to x\ R\ z\end{aligned}$$

Again, we can show [Basin *et al.*, 1997a] that this really formalizes $\mathrm{S4}$.

The limits of K_L

It might appear from the discussion above that we can implement any modal logic we want, simply by adding the axioms for the appropriate accessibility relation to K_L. That is, we represent a logic by embedding its semantics in the metalogic, a formalization technique that is sometimes called *semantic embedding* (see van Benthem [1984] or Ohlbach [1993] for details on this approach). We must be careful though; not every embedding based on labelling accurately captures the semantics and different kinds of embeddings capture more structure than others.

Consider K_L again: the rules for $\supset$ and $\Box$ reflect the meaning that the Kripke semantics gives the connectives. On the other hand, the semantics does not explicitly state how the rules for $\bot$ should function using labels. Following from the

rules for $\supset$, a plausible formalization is

$$\dfrac{\begin{array}{c}[x{:}A \supset \bot] \\ \vdots \\ x{:}\bot\end{array}}{x{:}A}\ \bot\text{-}E^{-}$$

which, like the rule for implication, stays in one world and ignores the existence of different possible worlds. But if we investigate the logic that results from using this rule (instead of $\bot$-E), we find that it is not complete with respect to the Kripke semantics.

An examination of the Gentzen-style natural language characterization of $\bot$ shows where the problem lies:

> If the assumption that $A \supset \bot$ in world x is inconsistent with the interpretation, then A is true in world x.

This says nothing about where the inconsistency might be and specifically does not say that it should be in the world x itself. The role of negation in encoding the semantics of logics is subtle and we lack space to develop this topic here. We therefore restrict ourselves to a few comments; much more detail can be found in [Basin *et al.*, 1997a]. In K_L we assumed that it is enough to be able to show that the inconsistency is in some world. It turns out that this is sufficient for a large class of logics; but again this does not reflect the complete behavior of $\bot$. Some accessibility relations require a richer metalogic than one based on minimal implication and this may in turn require formalizing all of first or even higher-order logic. In such formalizations, we must take account of the possibility that the inconsistency of an assumption about a world might manifest itself as a contradiction in the theory of the accessibility relation, or *vice versa*. It is possible then to use classical first (or higher-order) logic as a metatheory to formalize the Kripke semantics in a complete way, however the result also has drawbacks. In particular, we lose structure in the proofs available in K_L. In K_L we reason in two separate systems. We can reason in just the theory of the accessibility relation and then use the results of this in the theory of the labelled propositions; however, we cannot go in the other direction, i.e. we cannot use reasoning in the theory of labelled propositions as part of an argument about the relations. This enforced separation provides extra structure that we can exploit, e.g., to bound proof search (see, e.g., [Basin *et al.*, 1997b]). And in spite of enforcing this separation, K_L is a sufficient foundation for a very large class of standard logics.[19]

[19] In [Basin *et al.*, 1997a] we show that it is sufficient to define almost all the modal logics of the so-called Geach hierarchy, which includes most of those usually of interest, i.e. K, T, $\mathrm{S4}$, $\mathrm{S5}$, etc., though not, e.g., the modal logic of provability G [Boolos, 1993].

5.4 *Alternative multiple judgment systems*

Using a labelled deductive system, we built a deductive calculus for a modal logic based on two judgments: a formula is true in a world and one world accesses another. But this is only one of many possible presentations. We now consider another possibility, using multiple judgments that distinguish between truth (in a world) and validity that is due originally to Avron *et al.* [1992] (and further developed in [Avron *et al.*, 1998]).

Validity

Starting with K, we can proceed in a Gentzen-like manner, by writing down the behavior of the logical connectives as given by the Kripke semantics. If we abbreviate 'A is true in all worlds' to $V(A)$ (A is valid; i.e. A is a theorem), then we have

$$\frac{V(A \supset B) \quad V(A)}{V(B)} \supset\text{-}E_V \qquad \text{and} \quad \frac{V(A)}{V(\Box A)} \Box\text{-}I_V\,, \tag{23}$$

which can be easily verified against the Kripke semantics (they directly reflect the two rules Det and Nec). Since the deduction theorem fails, we do not have an introduction rule for $\supset$ in terms of V, neither do we have a rule for $\bot$.

Thus the first part of the signature for K simply records the rules (23):

$$\begin{aligned}\Sigma_1 \equiv\ & o{:}*,\ V{:}o \to *,\\ & False{:}o,\ imp{:}o \to o \to o,\ box{:}o \to o,\\ & impE_V{:}\Pi A^o B^o.\, V(A) \to V(imp\ A\ B) \to V(B),\\ & boxI_V{:}\Pi A^o.\, V(A) \to V(box\ A)\end{aligned}$$

And, as observed above, we have

LEMMA 23. *The rules encoded in Σ_1 are sound with respect to the standard Kripke semantics of* K*, if we interpret $V(A)$ as $\forall a.\, a \Vdash A$.*

The rest of V. The rest of the details about $V(\cdot)$ could be summarized simply by declaring all the axioms of K to be valid. In this case $V(\cdot)$ would simply encode a Hilbert presentation of K. However there is a more interesting possibility, which supports proof under assumption.

Truth in a world

As previously observed we do have a kind of semantic version of the deduction theorem relativized to any given world. We can use this to formalize more about

the meaning of implication and $\perp$. If we abbreviate 'A is true in some arbitrary but fixed world c' to $T(A)$, then we have:

$$\frac{T(A \supset B) \quad T(A)}{T(B)} \supset\text{-}E_T \qquad \frac{\begin{array}{c}[T(A)] \\ \vdots \\ T(B)\end{array}}{T(A \supset B)} \supset\text{-}I_T \qquad \frac{\begin{array}{c}[T(A \supset \perp)] \\ \vdots \\ T(\perp)\end{array}}{T(A)} \perp\text{-}E_T$$

When we talked above about validity we did not have a rule for introducing implication, or reflecting the behavior of $\perp$. Similarly, when we are talking about truth in some world, we do not have a rule reflecting the behavior of $\Box$, since that is not dependent on just the one world. Thus for instance, there is no introduction (or elimination) rule for this operator. All we can say is that

$$\frac{T(\Box(A \supset B)) \quad T(\Box A)}{T(\Box B)}\, Norm_T$$

And again we can verify these rules against the semantics.

Thus the second part of the signature is

$$\begin{aligned}\Sigma_2 \equiv\ & T{:}o \to *, \\ & impE{:}\Pi A^o B^o.\, T(imp\ A\ B) \to T(A) \to T(B)\,, \\ & impI{:}\Pi A^o B^o.\, (T(A) \to T(B)) \to T(imp\ A\ B)\,, \\ & FalseE{:}\Pi A^o.\, (T(imp\ A\ falsum) \to T(falsum)) \to T(A)\,, \\ & norm{:}\Pi A^o B^o.\, T(box\ (imp\ A\ B)) \to T(box\ A) \to T(box\ B)\end{aligned}$$

and we have

LEMMA 24. *The rules encoded in Σ_1, Σ_2 are sound with respect to the standard Kripke semantics of* K, *if we interpret $V(A)$ as in Lemma 23 and $T(A)$ as $c \Vdash A$ where c is a fixed constant.*

Connecting V and T

We have now formalized two separate judgments, defined by the predicates V and T, which we have to connect together. To this end, we introduce two rules, C and R, which intuitively allow us to introduce a validity judgment, given a truth judgment, and eliminate (or use) a validity judgment.

C states that if A is true in an arbitrary world, then it is valid.

$$\frac{T(A)}{V(A)}\, C$$

For this rule to be sound, we require that the world where A is true really is arbitrary, and this will hold so long as there are no other assumptions $T(A')$ current when we apply it. It is easy to see that this condition is ensured for any proof of the atomic proposition $V(A)$, so long as C is the only rule connecting T and V together, since the form of the rules then ensures that there can be no hypotheses $T(A')$ at a place where C is applied.

The addition of C yields a complete inference system for reasoning about valid formulae. However, the resulting deductive system is awkward: once we end up in the V fragment of the system, which by itself is essentially a Hilbert calculus, we are forced to stay there.

We thus extend our system with an elimination rule for V, to allow us to return to the natural deduction-like T fragment. Important to our justification of the rule C was that the premise followed in an *arbitrary* world. Any further rule that we add must not invalidate this assumption. However we observe that given $V(A)$, then, since A is valid, it *is* true in an arbitrary world, so adding $T(A)$ as an open assumption to an application of C does not harm the semantic justification of that rule application. We can encode this as the following rule R.

$$\frac{\begin{array}{cc} & [T(A)] \\ & \vdots \\ V(A) & V(B) \end{array}}{V(B)}\, R$$

Taken together, these rules complete our proposed encoding of K.

$$\begin{aligned} \Sigma_{\mathrm{K}_{MJ}} \equiv\ & \Sigma_1,\ \Sigma_2, \\ & C{:}\Pi A^o.\, T(A) \to V(A)\,, \\ & R{:}\Pi A^o B^o.\, V(A) \to (T(A) \to V(B)) \to V(B) \end{aligned}$$

To establish correctness formally, we begin by proving that:

PROPOSITION 25. *If A is a theorem of* K*, then $V(A)$ is a theorem of the proof calculus encoded as $\Sigma_{\mathrm{K}_{MJ}}$.*

Proof. We observe that if A is one of the listed axioms of K, then we can show that $T(A)$, and thus, by C, that $V(A)$. Therefore we need not declare these to be 'valid'. These, and the rules encoded in Σ_1 allow us to reconstruct any proof in Hilbert K (see also the remarks after Lemma 23 above). ■

Next that:

PROPOSITION 26. *If $V(A)$ is a theorem of the proof calculus encoded as $\Sigma_{\mathrm{K}_{MJ}}$, then A is a theorem of* K.

We prove a slight generalization, for which we need

LEMMA 27. *For an arbitrary set* Γ *of theorems of* K, *if* $T(A)$ *is a theorem of* $\Sigma_{\mathrm{K}_{MJ}}$ *extended with the axiom set* $\{\ T(x) \mid x \in \Gamma\ \}$, *then* A *is a theorem of* K.

Proof. First notice that only the rules encoded as $impI$, $impE$, $FalseE$ and $norm$ can occur in a proof of $T(A)$. By Lemma 24, reading $T(A)$ as $c \Vdash A$, only theorems of K follow from this fragment of the proof calculus extended with Γ, where Γ consists of theorems of K. ■

The generalization of Proposition 26 is then

LEMMA 28. *For an arbitrary set* Γ *of theorems of* K, *if* $V(A)$ *is a theorem of the proof calculus encoded as* $\Sigma_{\mathrm{K}_{MJ}}$ *extended with* $\{\ T(x) \mid x \in \Gamma\ \}$, *then* A *is a theorem of* K.

Proof. The proof is by induction on the size of a proof Π of $V(A)$. We need to consider three cases: (i) The last rule in the proof is an application of one of the rules encoded in Σ_1 from theorems $V(A_i)$, in which case, by appeal to the induction hypothesis, A_i are theorems of K and thus A is a theorem of K. (ii) The last rule is an application of C, in which case the sub-proof is a proof of the theorem $T(A)$ (there are no undischarged assumptions for the theorem $V(A)$ and hence $T(A)$) and by Lemma 27, A is a theorem of K. (iii) The last rule is an application of R to proofs Π_1 of $V(B)$ from $T(A)$ and Π_2 of the theorem $V(A)$. Since Π_2 is smaller than Π, by the induction hypothesis A is a theorem of K. Then we can transform Π_1 into a proof of the theorem $V(B)$ in the proof calculus formalized as $\Sigma_{\mathrm{K}_{MJ}}$ extended with $\{\ T(x) \mid x \in \Gamma \cup \{A\}\ \}$ by replacing any appeal to a hypothesis $T(A)$ in Π_1 by an appeal to the axiom $T(A)$. Since the result is a proof no bigger than Π_1, which in turn is smaller than Π, by appeal to the induction hypothesis, B is a theorem of K. ■

We can combine Propositions 25 and 26 as

THEOREM 29. A *is a theorem of* K *iff* $V(A)$ *follows from* $\Sigma_{\mathrm{K}_{MJ}}$.

Encoding other modal logics

We can extend the encoding of K easily to deal with S4; there are in fact several ways we can do this: one possibility (see Avron *et al.* [1992] for more discussion) is to add the rules

$$\frac{\begin{array}{c}[V(\Box A)]\\ \vdots\\ V(B)\end{array}}{V(\Box A \supset B)}\ \supset\text{-}I_V \qquad \text{and} \qquad \frac{V(\Box A)}{V(A)}\ \Box\text{-}E_V$$

(given these rules we can show that the $Norm_T$ rule is redundant). This produces an encoding that is closely related to the version of S4 suggested by Prawitz.

An alternative view of K_{MJ}

We have motivated and developed the K_{MJ} presentation of K using a Kripke semantics, as a parallel with K_L. Unlike K_L, however, the interpretation is implicit, not explicit (there is no mention of particular worlds in the final proof calculus) so we are not committed to it. In fact, and importantly, this presentation can be understood from an entirely different perspective that uses just the Hilbert axiomatization itself, as follows.

We observe in §7 that it is possible to prove a deduction theorem for classical propositional logic, justifying the $\supset$-I rule by proof-theoretic means in terms of the Hilbert presentation. If we examine that proof, then we can see that it is easily modified for the fragment of (our Hilbert presentation of) K *without* the rule Nec. But this is precisely the fragment of K that is defined by the T fragment of K_{MJ}. Equally, the system defined by V, can be seen as the full Hilbert calculus.

Thus we can alternatively view the two judgments of K_{MJ} not as indicating whether we are speaking of truth in some world, or truth in all worlds, but rather whether or not we are allowed to apply the deduction theorem. This perspective provides the possibility of an entirely different proof of the correctness of our encoding, based on the standard Hilbert encoding, and without any intervening semantic argument.

5.5 Some conclusions

We have presented two different encodings of two well-known modal logics in this section as examples of approaches to representing nonstandard logics in an $\rightarrow$-framework. Which approach is preferable depends, in the end, on how the resulting encoding will be used. K_L and $\mathrm{S4}_L$ make the semantic foundation of the presentation more explicit. This is advantageous if we take for granted the view that modal logics are the logics of Kripke models since the user is able to exploit the associated intuitions in building proofs. On the other hand this may be a problem if we want to use our encoding in circumstances where our intuitions are different. The opposite holds for K_{MJ} and $\mathrm{S4}_{MJ}$: it is more difficult to make direct use of any intuitions we might have from the Kripke semantics, but, since the proof systems involve no explicit, or even necessary, commitment to that interpretation, we have fewer problems in assuming another.[20]

The solutions we have examined, while tailored for modal logics, do have some generality. However, each different logic must be considered in turn and the approaches presented here may not always be applicable, or the amount of effort in modifying them may be considerable. For instance it is possible to interpret relevance logics in terms of a Kripke semantics that can be adopted as the basis of

[20] In fact it would be fairly easy to adapt a multiple judgment presentation of $\mathrm{S4}$ to the different circumstances of say relevance, or linear, propositional logic, which share many properties with traditional $\mathrm{S4}$. Such a project would be considerably more difficult, not to mention questionable, starting from $\mathrm{S4}_L$. This issue of commitment to a particular interpretation is discussed at length with regard to labelled deductive systems in general by Gabbay [1996].

a labelled deductive system, similar in style to (though considerably more complex than) K_L (see Basin *et al.* [1998b]). But such an implementation is tricky to use and we reach, eventually, the limits of encodings that are understandable and usable by humans.

6 CONSEQUENCE RELATIONS

In the previous sections we considered an abstraction of natural deduction calculi and in the following section we will consider an abstraction of Hilbert calculi. Here, we consider the third style of proof calculus we mention in the introduction: the sequent calculus. It turns out that, unlike the other two, little work has been done on systems that directly abstract away from sequent calculi in a way that we can use as a logical framework. This certainly does not mean that there has been no work on the principles of the sequent calculus, just that work has concentrated not on the concrete implementational aspects so much as on the abstract properties of the sequent relation, $\vdash$, which when investigated in isolation is called a *consequence relation*.

Consequence relations provide a powerful tool for systematically analyzing properties of a wide range of logics, from the traditional logics of mathematics to modal or substructural logics, in terms that we can then use as the starting point of an implementation. In fact it is often possible to encode the results of a sequent calculus analysis directly in an $\rightarrow$-framework.

What, then, is a consequence relation? There are several definitions in the literature (e.g. Avron [1991; 1992], Scott [1974] and Hacking [1979]); we adopt that of Avron, along with his vocabulary, where possible.

DEFINITION 30. A *consequence relation* is a binary relation between finite multisets of formulae Γ, Δ, usually written $\Gamma \vdash \Delta$, and satisfying at least $Basic$ and Cut in (2) and PL and PR in (3).[21]

This is, however, a very general definition. In fact most logics that we might be interested in encoding have natural presentations in terms of more constrained *ordinary* consequence relations:[22]

DEFINITION 31. A consequence relation is said to be *ordinary* if it satisfies the rules WL, CL, WR and CR of (3).

Examples of ordinary consequence relations are $\mathrm{LJ}^{\supset}$ and $\mathrm{LK}^{\supset}$ defined in §2.2,

[21] In the rules given in §2.2, the antecedent and succedent are *sequences* of formulae whereas here they are *multisets*. In practice, the permutation rules PL and PR are often omitted and multi-sets are taken as primitive, as here. This is not always possible though, e.g., in $\mathcal{R}$, where the ordering in the sequence matters. Note also that this definition does not take account of variables; for an extension to that case, see Avron [1992].

[22] We do not have space to consider the best known exception, the *substructural* logics, e.g. relevance and linear logic. However what we say in this section generalizes, given a suitably modified $\rightarrow$-framework, to these cases. Readers interested in the (non-trivial) technical details of this modification are referred to [Cervesato and Pfenning, 1996] and, especially, [Ishtiaq and Pym, 1998].

as is the encoding in terms of $\vdash$ of $\mathrm{NJ}^{\supset}$ in §3.1 (even though Cut is not a basic property of this presentation, we can show that it is admissible; i.e. we do not change the set of provable sequents if we assume it). Most of the traditional logics of mathematics can be presented as ordinary (in fact, *pure* ordinary, see below) consequence relations.

6.1 The meaning of a consequence relation

While it is possible to treat a consequence relation purely in syntactic terms, often one can be understood, and may have been motivated, by some idea of the 'meaning' of the formulae relative to one another. For instance we can read a sequent $\Gamma \vdash \Delta$ of $\mathrm{LK}^{\supset}$ as 'if all the formulae in Γ are true, then at least one of the formulae in Δ is true.' Because they have this reading in terms of truth, we call the systems defined by $\mathrm{LJ}^{\supset}$, $\mathrm{LK}^{\supset}$ and $\mathrm{NJ}^{\supset}$ 'truth' consequence relations. Notice that the meaning of 'truth' here is not fixed: when we say that something is true we might mean that it is classically true, intuitionistically true, Kripke-semantically true, relevance true, or even something else more exotic.

We can derive a different sort of consequence relation from a Hilbert calculus: if we assume (1) $Basic$, i.e. that $A \vdash A$, (2) that if A is an axiom then $\Gamma \vdash A$, and (3) that for each rule of proof

$$\frac{A_1 \ldots A_n}{A}$$

we have that

$$\frac{\Gamma_1 \vdash A_1 \ldots \Gamma_n \vdash A_n}{\Gamma_1, \ldots, \Gamma_n \vdash A}$$

then it is easy to show that the resulting system satisfies Cut, and that $\vdash A$ iff A is a theorem. This is not a truth consequence relation: the natural reading of $\Gamma \vdash A$ is 'if Γ are *theorems*, then A is a *theorem*'. We thus call $\vdash$ a *validity* consequence relation.

Of course truth and validity are not the only possibilities. We can define consequence relations any way we want,[23] the only restriction we might impose is that in order to be effectively mechanizable on a computer, the relation $\vdash$ should be recursively enumerable.

6.2 Ordinary pure consequence relations and $\rightarrow$-frameworks

Part of the problem with mechanizing consequence relations, if we formalize them directly, is their very generality. Many systems are based on ordinary consequence relations, and if an encoding forces us to deal explicitly with all the rules that

[23] For some examples of other consequence relations which can arise in the analysis of a modal logic, see Fagin *et al.* [1992].

formalize this, then proof construction will often require considerable and tedious structural reasoning. This may explain, in part, why there has been little practical work on logical frameworks based directly on consequence relations.[24] However another reason is that an $\rightarrow$-framework can very effectively encode many ordinary consequence relations directly. In the remainder of this section we explore this reduction, which clarifies the relationship between consequence relations and $\rightarrow$-frameworks as well as illuminating some of the strengths and weaknesses of $\rightarrow$-frameworks.

In order to use an $\rightarrow$-framework for representing consequence relations, it helps if we impose a restriction in addition to ordinaryness.

DEFINITION 32. We say that a consequence relation is *pure* if, given that

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \ldots \quad \Gamma_n \vdash \Delta_n}{\Gamma_0 \vdash \Delta_0}$$

holds, then there are Γ_i', Δ_i', which are sub-multisets of Γ_i, Δ_i, such that for arbitrary Γ_i'', Δ_i''

$$\frac{\Gamma_1', \Gamma_1'' \vdash \Delta_1', \Delta_1'' \quad \ldots \quad \Gamma_n', \Gamma_n'' \vdash \Delta_n', \Delta_n''}{\Gamma_0', \Gamma_0'' \vdash \Delta_0', \Delta_0''}.$$

Notice that the consequence relations discussed at the beginning of this section all satisfy the definition of purity; this is also the case for most of the logics we encounter in mathematics. In order to find counterexamples we must look to some of the systems arising in philosophical logic. For instance in modal logic (see §5, and the discussion by Avron [1991]) we get a natural truth consequence relation satisfying the rule

$$\frac{\vdash A}{\vdash \Box A}$$

but not, for arbitrary Γ,

$$\frac{\Gamma \vdash A}{\Gamma \vdash \Box A}$$

We shall not, here, consider frameworks that can handle general impure consequence relations satisfactorily.

Single conclusioned consequence

As we said earlier, most of the standard logics of mathematics have intuitive presentations as ordinary pure consequence relations. Avron [1991] adds one more restriction before claiming that

[24] There are, however, many computer implementations of particular sequent calculi, e.g. the PVS system for higher-order logic, by Owre *et al.* [1995], not to mention many tableau calculi, which are, essentially, sequent calculi.

> Every ordinary, pure, single-conclusioned [consequence relation] system can, e.g., quite easily be implemented in the Edinburgh LF.

We begin by considering the single conclusioned case, and then follow it with multiple conclusioned consequence relations and systems based on multiple consequence relations.

The encoding is uniform and consists of two parts. We first explain how to encode sequents and then rules. We can encode

$$A_1, \ldots, A_n \vdash A$$

as

$$T(A_1) \to \cdots \to T(A_n) \to T(A)$$

and it is easy to show that this satisfies Definition 31 of an ordinary consequence relation (in this case, single conclusioned). Notice how the structural properties of the consequence relation are directly reflected by the logical properties of $\to$.

We can then represent basic and derived rules expressed in terms of such a consequence relation, *assuming it is pure* as follows. Consider a rule of such a consequence relation $\vdash$, where, for *arbitrary* Γ_i:

$$\frac{\Gamma_1, A_{(1,1)}, \ldots, A_{(1,n_1)} \vdash A_1 \quad \cdots \quad \Gamma_m, A_{(m,1)}, \ldots, A_{(m,n_m)} \vdash A_m}{\Gamma_1, \ldots, \Gamma_m, A_{(0,1)}, \ldots, A_{(0,n_0)} \vdash A_0}$$

We can encode this as

$$\begin{aligned}(T(A_{(1,1)}) \to \cdots \to T(A_{(1,n_1)}) \to T(A_1)) \to \cdots \\ \to (T(A_{(m,1)}) \to \cdots \to T(A_{(m,n_m)}) \to T(A_m)) \\ \to T(A_{(0,1)}) \to \cdots \to T(A_{(0,n_1)}) \to T(A_0)\end{aligned}$$

leaving the Γ_i implicit (the condition of purity is important because it allows us to do this).

As an example, consider $\mathrm{LJ}^{\supset}$ from §2.2, for which we have the rules

$$\frac{\Gamma \vdash A \quad \Gamma', B \vdash C}{\Gamma, \Gamma', A \supset B \vdash C}\ \supset\text{-}L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset\text{-}R$$

We can encode these rules (assuming a suitable formalization of the language) as

$$T(A) \to (T(B) \to T(C)) \to T(A \supset B) \to T(C) \tag{24}$$

and

$$(T(A) \to T(B)) \to T(A \supset B)\,. \tag{25}$$

Then we can simulate the effect of applying $\supset$-L, i.e.

$$\frac{F_1, \ldots, F_m \vdash A \quad G_1, \ldots, G_n, B \vdash C}{F_1, \ldots, F_m, G_1, \ldots, G_n, A \supset B \vdash C},$$

encoded as (24), to the encoded assumptions

$$T(F_1) \to \cdots \to T(F_m) \to T(A) \tag{26}$$

and

$$T(G_1) \to \cdots \to T(G_n) \to T(B) \to T(C)\,. \tag{27}$$

By (26) and (24) we have

$$T(F_1) \to \cdots \to T(F_m) \to (T(B) \to T(C)) \to T(A \supset B) \to T(C) \tag{28}$$

which combines with (27) to yield

$$T(F_1) \to \cdots \to T(F_m) \to T(G_1) \to \cdots \to T(G_n) \to T(A \supset B) \to T(C)\,,$$

which encodes the conclusion of $\supset$-L.

From this, it is easy to see that our encoding of $\mathrm{LJ}^{\supset}$ is adequate. We show faithfulness by a modification of the techniques discussed above.

An observation about implementations

Note that the last step, 'combining' (28) with (27), actually requires a number of steps in the metalogic; e.g. shuffling around formulae by using the fact that, in the metalogic of the framework itself, $T(A) \to T(B) \to T(C)$ is equivalent to $T(B) \to T(A) \to T(C)$. But such reasoning must come out somewhere in any formalism of a system based on consequence relations. There is no getting around some equivalent of structural reasoning; the question is simply of how, and where, it is done, and how visible it is to the user.

In fact in actual implementations of $\to$-frameworks, e.g., Isabelle [Paulson, 1994], the cost of this structural reasoning is no greater than in a custom implementation since the framework theory itself will be implemented in terms of a pure, ordinary single conclusioned consequence relation; i.e. as a sequent or natural deduction calculus. If we write the consequence relation of the implementation as $\Longrightarrow$, then it is easy to see that an encoded sequent such as (26) can be quickly transformed into the logically equivalent

$$T(F_1), \ldots, T(F_m) \Longrightarrow T(A)$$

at which point it is possible to 'piggy-back' the rest of the proof off of $\Longrightarrow$, and thus make use of the direct (and thus presumably efficient) implementation of its structural properties.

Multiple conclusioned consequence

Having examined how we might encode ordinary pure single conclusioned consequence relations in an $\rightarrow$-framework, we now consider the general case of multiple conclusioned relations, which occur in standard presentations of many logics (e.g. $\mathrm{LK}^{\supset}$ described in §2.2).

There is no obvious correspondence between such relations and formulae in the language of $\rightarrow$. However, by refining our encoding a little, we can easily extend Avron's observation to the general case and thereby give a direct and effective encoding of multiple conclusioned consequence relations. We take as our example the system $\mathrm{LK}^{\supset}$. For larger developments and applications of this style, the reader is referred to [Pfenning, 2000].

A multiple conclusioned consequence relation is a pair of multisets of formulae, which we can refer to as 'left' and 'right'; i.e.

$$\overbrace{A_1, \ldots, A_n}^{left} \vdash \overbrace{B_1, \ldots, B_m}^{right}. \tag{29}$$

To encode this we need not one judgment T, but *two* judgments which we call T_L and T_R; we also define a new propositional constant E. We can encode (29) in an $\rightarrow$-framework as

$$T_L(A_1) \rightarrow \cdots \rightarrow T_L(A_n) \rightarrow T_R(B_1) \rightarrow \cdots \rightarrow T_R(B_m) \rightarrow E\,.$$

This is not quite enough to give us a consequence relation though; unlike in the single conclusioned case, we do not automatically get that $Basic$ is true. However we can remedy this by declaring that

$$T_L(A) \rightarrow T_R(A) \rightarrow E$$

and then we can show that the encoding defines an ordinary consequence relation.

We can then extend the style of encoding described to define, e.g., the right and left rule for implication in $\mathrm{LK}^{\supset}$, which are

$$(T_L(A) \rightarrow T_R(B) \rightarrow E) \rightarrow T_R(A \supset B) \rightarrow E$$

and

$$(T_R(A) \rightarrow E) \rightarrow (T_L(B) \rightarrow E) \rightarrow T_L(A \supset B) \rightarrow E\,.$$

As in the single conclusioned case, a necessary condition for this encoding is that the consequence relation is pure. It is also easy to show that the encoding is adequate and faithful with respect to the derivability of sequents in $\mathrm{LK}^{\supset}$.

Multiple consequence relation systems

Finally, we come to the problem of how to formalize systems based on more than one consequence relation. Here we will briefly consider just the single conclusioned case; the same remarks, suitably modified, also apply to the multiple conclusioned case.

One approach to analyzing a formal system (and quite useful if we want the analysis to be in terms of pure consequence relations) is, rather than using a single relation, to decompose it into a set of relations $\vdash_1$ to $\vdash_n$. We can encode each of these relations, and their rules, just like in the single relation case, using predicates T_1 to T_n; i.e.

$$A_1, \ldots, A_n \vdash_i A$$

as

$$T_i(A_1) \to \cdots \to T_i(A_n) \to T_i(A).$$

We encounter a new encoding problem with multiple consequence systems. These typically (always, if they are interesting) contain rules that relate consequence relations to each other, i.e., rules where the premises are built from different consequence relations than the conclusion. Consider the simplest example of this, which simply declares that one consequence relation is a subrelation of another (what we might call a $bridge$ rule):

$$\frac{\Gamma \vdash_1 A}{\Gamma \vdash_2 A}\ bridge$$

We can encode this as the pair of schemata

$$T_1(A) \to T_2(A) \tag{30}$$

and

$$(T_1(A) \to T_2(B)) \to T_2(A) \to T_2(B) \tag{31}$$

and show that the resulting encoding is properly closed. That is, given

$$T_1(A_1) \to \cdots \to T_1(A_n) \to T_1(B)$$

by (30) we get

$$T_1(A_1) \to \cdots \to T_1(A_n) \to T_2(B)$$

then by n applications of (31) we eventually get

$$T_2(A_1) \to \cdots \to T_2(A_n) \to T_2(B)\,.$$

As this example shows, working with a multiple consequence relation system in an $\rightarrow$-framework may require many metalevel steps to simulate a proof step in the object logic (here the number depends on the size of the sequent). More practical experience using such encodings is required to judge whether they are really usable in practice as opposed to just being a theoretically interesting way of encoding multiple consequence relation systems in the logic of $\rightarrow$.

In fact we have already encountered an example of this kind of an encoding: the multiple judgment encoding of modal logic developed in [Avron *et al.*, 1992; Avron *et al.*, 1998], described in §5.4, can be seen as the encoding of a two consequence relation system for truth and validity where T_1 is T, T_2 is V and $bridge$ is implemented by R and C.

7 DEDUCTIVE SYSTEMS AS INDUCTIVE DEFINITIONS

In the introduction we discussed two separate traditions of metatheory: metatheory as a unifying language and metatheory as proof theory. We have shown too how $\rightarrow$-frameworks fit into the unifying language tradition, and the way different logics can be encoded in them and used to carry out proofs. However, $\rightarrow$-frameworks are inadequate for proof theory: in exchange for ease of reasoning *within* a logic, reasoning *about* the logic becomes difficult or impossible.

In order better to understand this point, and some of the subtleties it involves, consider the following statements about the (minimal) logic of $\supset$.

1. $A \supset A$ is a theorem.
2. $A \supset B \supset C$ is true on the assumption that $B \supset A \supset C$ is true.
3. The deduction theorem holds for $\mathrm{HJ}^{\supset}$.
4. The deduction theorem holds for all extensions of $\mathrm{HJ}^{\supset}$ with additional axioms.

Statement 1 can be formalized in a metalogic as a statement about provability in any complete presentation of the logic of $\supset$; e.g. $\mathrm{NJ}^{\supset}$, $\mathrm{LJ}^{\supset}$ or $\mathrm{HJ}^{\supset}$. As a statement about provability we might regard it as, in some sense, a proof-theoretic statement. But as such, it is very weak since, by completeness, it must be provable irrespective of the deductive system used.

Statement 2 expresses a derived rule of the logic of $\supset$. Its formalization requires that we can reason about the truth of one formula relative to others. As explained in §6, representing this kind of a (truth) consequence relation can easily be reduced to a provability problem in an $\rightarrow$-framework. For example, in the λ^{p} encoding of $\mathrm{NJ}^{\supset}$ we prove that the type $\Pi A^o B^o C^o.\, pr(B \supset A \supset C) \rightarrow pr(A \supset B \supset C)$ is inhabited.

As with statement 1, the metalogic must be able to build proofs in the object logic (in this case though under assumptions, encoded using $\rightarrow$ in the metalogic), but proofs themselves need not be further analyzed.

Statement 3 (which we prove in this section) is an example of a metatheoretic statement that is more in the proof theory tradition. In order to prove it we must analyze the structure of arbitrary proofs in the deductive system of $\mathrm{HJ}^{\supset}$ using an inductive argument. The difference between this and the previous example is important: for a proof of statement 2 we need to know which axioms and rules are available in the formalized deductive system, while for a proof of statement 3 we also need to know that no other rules are present, since this is what justifies an induction principle over proofs (the rules of $\mathrm{HJ}^{\supset}$ can be taken as constituting an inductive definition). An $\rightarrow$-framework like λ^{p} contains no provisions for carrying out induction over the structure of an encoded deductive system.

Statement 4 is an extension of statement 3, which introduces the problem of *theory structuring*. Structuring object theories to allow metatheoretic results to be 'imported' and used in related theories is not a substantial problem in the kinds of metamathematical investigations undertaken by proof theorists, who are typically interested in proving particular results about particular systems. However, for computer scientists working on formal theorem proving, it is enormously important: it is good practice for a user reasoning about complex theories to formalize a collection of simpler theories (e.g. for numbers and arithmetic, sequences, relations, orders, etc.) that later are combined together as needed. So some kind of theory structuring facility is practically important and many systems provide support for it.[25]

Unfortunately, hierarchical structure in $\rightarrow$-frameworks is the result of an assumption (necessary anyway for other reasons) that the languages and deductive systems of encoded logics are 'open to extensions' (see §7.4 below), something that automatically rules out any arguments requiring induction on the structure of the language[26] or proofs of a theory, e.g. the deduction theorem. If we 'close' the language or deductive system by explicitly adding induction over the language or proofs, in order to prove metatheorems, it is unsound later to assume those theorems in extensions of the deductive system or language. In §7.4, we suggest that there is a way to avoid this problem if we formulate metatheorems in a metalogic based on inductive definitions.

These examples illustrate that there are different sorts of metatheoretic statements, which are distinguished by how conscious we have to be of the metatheoretic context in order to prove them. The central role of induction in carrying

[25] For example HOL [Gordon and Melham, 1993], Isabelle [Paulson, 1994] and their predecessor LCF [Gordon *et al.*, 1979] support simple theory hierarchies where theorems proven in a theory may be used in extensions.

[26] An example of a metatheorem for which we need induction over the language, not the derivations, is that *In classical logic, a propositional formula that contains only the $\Leftrightarrow$ connective is valid if and only if each propositional variable occurs an even number of times*.

out many kinds of the more general metatheoretic arguments is the reason we now consider logical frameworks based on inductive definitions.

7.1 Historical background: From Hilbert to Feferman

What kind of a metalogic is suited for carrying out metatheoretic arguments that require induction on the language or deductive system of the object logic? To answer this question we draw on experience gained by proof theorists dating back to Hilbert, in particular Post and Gödel, and later Feferman, who have asked very similar questions. We can also draw on practical experience over the last 30 years in the use of computers in work with formal systems.

The work of Post and Gödel

In the early part of this century, Post [1943] (see Davis [1989] for a short survey of Post's work) investigated the decidability of logics like that of *Principia Mathematica*. He showed that such systems could be formalized as (what we now recognize as) recursively enumerable classes of strings and that this provided a basis for metatheoretic analysis. Although Post's work is a large step towards answering our question, one important aspect, from our point of view, is missing from his formalization. There is no consideration of *formal* metatheory. Post was interested in a formal characterization of deductive systems, not of their metatheories; he simply assumed, reasonably for his purposes, that arbitrary mathematical principles could be adopted as necessary for the metatheory.

We cannot make the same assumption with a logical framework: we must decide first which mathematical principles we need, and then formalize them. The work of Post is thus complemented, for our purposes, by Gödel's [1931] work on the incompleteness of systems like Principia Mathematica, which shows that a fragment of arithmetic is sufficient for complex and general metatheory. Logicians have since been able to narrow that fragment down to the theory of primitive recursive arithmetic, which seems sufficient for general syntactic metatheory. [27]

The natural numbers, although adequate for Gödel's purposes, are too unwieldy to use as a logical framework. Indeed, a large part of Gödel's paper is taken up with a complicated technical development showing that arithmetic really can represent syntax. The result is unusable in a practical framework: the relationship between syntax and its encoding is only indirectly given by (relatively) complicated arithmetic functions and the numbers generated in the encoding can be enormous. This is in contrast to Post's strings (further investigated in the early sixties by mathematicians such as Smullyan [1961]), which have a simple and direct correspondence with the structures we want to encode, and a compact representation.

[27] This is a thesis, of course, not a theorem; and exceptions have to be made for, e.g., proof normalization theorems, for which (as a corollary of Godel's result itself) we know there can be no single general metatheory.

S-expressions and FS_0

It is possible to build a formal metatheory based on strings, in the manner of Post. However, experience in computer science in formalizing and working with large symbolic systems has shown that there is an even more natural language for modeling formal (and other kinds of symbolic) systems. The consensus of this experience is found in Lisp [McCarthy, 1981; Steele Jr. and Gabriel, 1996], which for more than 30 years has remained the most popular language for building systems for symbolic computation.[28] Further, Lisp is not only effective, but its basic data-structure, which has, in large part, contributed to its effectiveness, is remarkably simple: the *S-expression*. This is the data-type freely generated from a base type by a binary function 'Cons'. Experience with Lisp has shown that just about any syntactic structure can be mapped directly onto S-expressions in such a way that it is very easy to manipulate. In fact, we can even restrict the base type of S-expressions to be a single constant (often called 'nil') and still get this expressiveness. Further evidence for the effectiveness of Lisp and S-expressions is that one of the most successful theorem proving systems ever built, NQTHM [Boyer and Moore, 1981], verifies recursive functions written in a version of Lisp that uses precisely this class of S-expressions.

An example of a theory that integrates all the above observations is FS_0, due to Feferman. This provides us with a language in which we can define exactly all the recursively enumerable classes of S-expressions. Moreover, it permits inductive arguments over these inductively defined classes, and thus naturally subsumes both the theory of recursively enumerable classes and primitive recursive arithmetic. It has proved usable too in case-studies of computer supported metatheory, in the proof-theoretic tradition (see Matthews [1992; 1993; 1994; 1997b]). In the rest of this chapter we will use a slightly abstracted version of FS_0 for our discussion.

7.2 A theory of inductive definitions: FS_0

FS_0 is a simple minimal theory of inductive definitions, which we present here in an abstract form (we elide details in order to emphasize the general, rather than the particular, features). A full description of the theory is provided by Feferman [1990], while an implementation is described by Matthews [1996].

FS_0 is a theory of inductively defined sets, embedded in first-order logic and based on Lisp-style S-expressions. The class of S-expressions is the least class containing nil and closed under the pairing ($cons$) operator, which we write as an infix comma $(\cdot,\cdot)$, such that $nil \neq (a, b)$ for all S-expressions a and b; we also assume, for convenience, that comma associates to the right, so that $(a, (b, c))$ can

[28] This does not mean that there have not been successful programming languages that use strings as their basic data-structure; the SNOBOL family [Griswold, 1981] of languages, for example, is based on the theory of Markoff string transformation algorithms. However it is significant that SNOBOL, in spite of its mathematical elegance in many ways, has never been seen as a general purpose symbolic programming language like Lisp, or been adopted so enthusiastically by such a large and influential programming community.

be abbreviated to (a, b, c). We then have functions car and cdr, which return the first and second elements of a pair.

Comprehension over first-order predicates is available. We write

$$x \in S \Leftrightarrow P(x) \qquad \text{or} \qquad S \equiv \{\ x \mid P(x)\ \}$$

to indicate a set S so defined. Such definitions can be parameterized and the parameters are treated in a simple way, thus, for example, we can write

$$x \in S(a, b) \Leftrightarrow (x, a) \in b.$$

We can also define sets explicitly as inductive definitions using the $I(\cdot, \cdot)$ construction: if A and B are sets, then $I(A, B)$ is the least set containing A and closed under the rule

$$\frac{t_1 \quad t_2}{t}$$

where $(t, t_1, t_2) \in B$. Note that we only have inductive definitions with exactly two 'predecessors', but this is sufficient for our needs here and, with a little more effort, in general.

Finally, we can reason about inductively defined sets using the induction principle

$$\begin{aligned} Base \subseteq S \to \forall a,\ b,\ c.\, (b \in S \to c \in S \to \\ (a, b, c) \in Step \to a \in S) \to I(Base, Step) \subseteq S\,. \end{aligned} \quad (32)$$

This says that a set S contains all the members of a set $I(Base, Step)$ if it contains the members of $Base$ and whenever it contains two elements b and c, and (a, b, c) is an instance of the rule $Step$, then it also contains a. This induction principle applies to sets, not predicates, but it is easy, by comprehension, to generate one from the other, so this is not a restriction.[29]

7.3 A Hilbert theory of minimal implication

Having sketched a theory of inductive definitions, we now consider how it might actually be used, both to encode an object logic, and to prove metatheorems. As an example, we encode the theory $\mathrm{HJ}^{\supset}$ (of §2.1) and prove a deduction theorem.

The definition of HJ

The language L_{HJ} We define an encoding $\ulcorner \cdot \urcorner$ of the standard language of implicational logic L_{HJ} (as usual we distinguish metalevel ($\to$) from object level ($\supset$)

[29] In FS_0 comprehension is restricted to essentially Σ^0_1 predicates with the result that the theory is recursion-theoretically equivalent to primitive recursive arithmetic.

implication) as follows. We define two distinct S-expression constants (e.g. nil and (nil, nil)) which we call `atom` and `imp`, then we have

$$\begin{aligned} \ulcorner a \urcorner &= (\texttt{atom}, \ulcorner\!\ulcorner a \urcorner\!\urcorner) && (a \text{ atomic}) \\ \ulcorner a \supset b \urcorner &= (\texttt{imp}, \ulcorner a \urcorner, \ulcorner b \urcorner) \end{aligned}$$

(assuming $\ulcorner\!\ulcorner a \urcorner\!\urcorner$ for atomic a to be already, separately, defined). It is easy to see that $\ulcorner . \urcorner$ is an injection from $\mathrm{L_{HJ}}$ into the S-expressions, on the assumption that $\ulcorner\!\ulcorner . \urcorner\!\urcorner$ is. For the sake of readability, in the future we will abuse notation, and write simply $a \supset b$ when we mean the schema $(\texttt{imp}, a, b)$; i.e. a and b here are variables in the theory of inductive definitions, not propositional variables in the encoded language $\mathrm{L_{HJ}}$.

The theory HJ We now define a minimal theory of implication, HJ, as follows. We have two classes of axioms

$$x \in K \Leftrightarrow \exists a,\, b.\, x = (a \supset b \supset a)$$

and

$$x \in S \Leftrightarrow \exists a,\, b,\, c.\, x = ((a \supset b) \supset (a \supset b \supset c) \supset a \supset c)$$

and a rule of detachment

$$x \in Det \Leftrightarrow \exists a,\, b.\, x = (b, (a \supset b), a)$$

from which we define the theory HJ to be

$$\mathrm{HJ} \equiv I(K \cup S, Det)\,.$$

Using HJ

We can now use HJ to prove theorems of the Hilbert calculus $\mathrm{HJ}^{\supset}$ in the same way that we would use an encoding to carry out natural deduction proofs in an $\rightarrow$-framework. One difference is that we do not get a direct correspondence between proof steps in the encoded theory and steps in the derivation. However, in this case it is enough to prove first the following lemmas:

$$a \supset b \supset a \in \mathrm{HJ} \tag{33}$$

$$(a \supset b) \supset (a \supset b \supset c) \supset a \supset c \in \mathrm{HJ} \tag{34}$$

$$a \in \mathrm{HJ} \rightarrow (a \supset b) \in \mathrm{HJ} \rightarrow b \in \mathrm{HJ} \tag{35}$$

Here, and in future, we assume theorems are universally closed. From these, it follows that if A is a theorem of minimal implication, then $\ulcorner A \urcorner \in \mathrm{HJ}$.

For example, we show

$$a \supset a \in \mathrm{HJ} \tag{36}$$

with the derivation

1. $a \supset a \supset a \in \mathrm{HJ}$	by (33)
2. $a \supset (a \supset a) \supset a \in \mathrm{HJ}$	by (33)
3. $(a \supset a \supset a) \supset (a \supset (a \supset a) \supset a) \supset a \supset a \in \mathrm{HJ}$	by (34)
4. $(a \supset (a \supset a) \supset a) \supset a \supset a \in \mathrm{HJ}$	by (35),1,3
5. $a \supset a \in \mathrm{HJ}$	by (35),2,4

The steps of this proof correspond, one to one, with the steps of the proof that we gave for $\mathrm{HJ}^{\supset}$ in §2.1. As this example suggests, at least for propositional Hilbert calculi, inductive definitions support object theory where proofs in the metatheory closely mirror proofs in the object theory. (For logics with quantifiers the relationship is less direct, since we have to encode and explicitly reason about variable binding and substitution.)

Proving a deduction theorem for HJ

Let us now consider an example that requires induction over proofs themselves: the deduction theorem for HJ. We only sketch the proof; the reader is referred to Basin and Matthews [2000] for details. Informally, the deduction theorem says:

> If B is provable in HJ with the additional axiom A then $A \supset B$ is provable in HJ.

Note that this theorem relates different deductive systems: HJ and its extension with the axiom A. Moreover, as A and B are schematic, ranging over all formulae of HJ, the theorem actually relates provability in HJ with provability in infinitely many extensions, one for each proposition A.

To formalize this theorem we define

$$\mathrm{HJ}[\Gamma] \equiv I\,(K \cup S \cup \Gamma, Det)\,; \tag{37}$$

i.e. $\mathrm{HJ}[\Gamma]$ is the deductive system HJ where the axioms are extended by all formulae in the class Γ. Now we can formalize the deduction theorem as

$$b \in \mathrm{HJ}[\{a\}] \rightarrow (a \supset b) \in \mathrm{HJ}\,, \tag{38}$$

which in FS_0 can be transformed into

$$I(K \cup S \cup \{a\}, Det) \subseteq \{\; x \mid (a \supset x) \in \mathrm{HJ} \;\}\,.$$

This in turn can be proved by induction on the inductively defined set $I\,(K \cup S \cup \{a\}, Det)$ using (32). The proof proceeds as follows:

Base case We have to show $K \cup S \cup \{a\} \subseteq \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$. This reduces, via $x \in K \cup S \cup \{a\} \to (a \supset x) \in \mathrm{HJ}$, to showing that $(a \supset x) \in \mathrm{HJ}$ given either (i) $x \in K \cup S$ or (ii) $x \in \{a\}$. For (i) we have $x \in \mathrm{HJ}$ and $(x \supset a \supset x) \in \mathrm{HJ}$, and thus, by Det that $(a \supset x) \in \mathrm{HJ}$. For (ii) we have $x = a$ and thus have to show that $(a \supset a) \in \mathrm{HJ}$, which we do following the proof of (36) above.

Step case There is only one rule (Det) and thus one case: given $b \in \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$ and $(b \supset c) \in \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$, prove that $c \in \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$. This reduces to proving, given $(a \supset b) \in \mathrm{HJ}$ and $(a \supset b \supset c) \in \mathrm{HJ}$ that $(a \supset c) \in \mathrm{HJ}$. This in turn follows by observing that $((a \supset b) \supset (a \supset b \supset c) \supset a \supset c) \in \mathrm{HJ}$ by (34), from which, by (35) twice with the hypotheses, $(a \supset c) \in \mathrm{HJ}$.

Once we have proved the deduction theorem we can use it to build proofs where we reason under assumption in the style of natural deduction. This is useful, indeed in practice essential, if we really wish to use Hilbert calculi to prove anything. However it is also limited since this metatheorem can be applied *only* to HJ. Thus, we next consider how this limitation can be partially remedied.

7.4 Structured theory and metatheory

At the beginning of this section, we discussed two examples of the deduction theorem (statements 3 and 4) where the second stated that the deduction theorem holds not just in $\mathrm{HJ}^{\supset}$ but also in extensions. We return to this example, which illustrates an important difference between ID-frameworks and $\to$-frameworks.

Structuring in $\to$-frameworks

Let us first examine how theories can be structured in $\to$-frameworks. Consider the following: we can easily encode $\mathrm{HJ}^{\supset}$ as (assuming the encoding of the syntax is given separately) the axiom set $\Gamma_{\mathrm{HJ}^{\supset}}$.

$$T(A \supset B \supset A)$$
$$T((A \supset B) \supset (A \supset B \supset C) \supset A \supset C)$$
$$T(A) \to T(A \supset B) \to T(B)$$

Then A is a theorem of $\mathrm{HJ}^{\supset}$ iff $\Gamma_{\mathrm{HJ}^{\supset}} \vdash T(A)$, i.e., $T(A)$ is provable in the metalogic under the assumptions $\Gamma_{\mathrm{HJ}^{\supset}}$. Now consider the deduction theorem in this setting; we would have to show that

$$\Gamma_{\mathrm{HJ}^{\supset}} \vdash (T(A) \to T(B)) \to T(A \supset B)\,. \tag{39}$$

This is not possible, however.

In an $\to$-framework, a basic property of $\vdash$ is weakening; i.e. if $\Gamma \vdash \phi$ then $\Gamma, \Delta \vdash \phi$. This is very convenient for structuring theories: a theorem proven

under the assumptions Γ holds under extension with additional assumptions Δ. For example, Δ might extend our formalization of $\mathrm{HJ}^{\supset}$ to a classical theory of $\supset$ or perhaps to full first-order logic.[30] By weakening, given $\Gamma_{\mathrm{HJ}^{\supset}} \vdash \phi$ we immediately have $\Gamma_{\mathrm{HJ}^{\supset}}, \Delta \vdash \phi$. Thus we get a natural hierarchy on the object theories we define: theory T' is a subtheory of theory T when its axioms are a subset of those of T. This allows us to reuse proven metatheorems since anything proven for a subtheory automatically follows in any supertheory.

Consider, on the other hand, the extension Δ_{K} consisting of the axioms

$$T(A) \to T(\Box A)$$
$$T(\Box A \supset \Box(A \supset B) \supset \Box B)$$

with which we can extend $\Gamma_{\mathrm{HJ}^{\supset}}$ to a fragment of the modal logic K. The deduction theorem *does not* follow in K; therefore, since by faithfulness we have

$$\Gamma_{\mathrm{HJ}^{\supset}}, \Delta_{\mathrm{K}} \nvdash (T(A) \to T(B)) \to T(A \supset B)\,,$$

we must also have, by weakening and contraposition,

$$\Gamma_{\mathrm{HJ}^{\supset}} \nvdash (T(A) \to T(B)) \to T(A \supset B)\,.$$

This suggests that there is an either/or situation: we can have either hierarchically structured theories, as in an $\to$-framework, or general inductive metatheorems (like the deduction theorem), as in an ID-framework, but not both. In fact, as we will see, in an ID-framework things are not quite so clear-cut: there is the possibility both to prove metatheorems by induction and to use them in certain classes of extensions.

Structuring in an ID-framework

Part of the explanation of why we can prove (38), but not (39), is that it is not possible to extend the deduction theorem for HJ to arbitrary supertheories: (38) is a statement about HJ and it tells us nothing about anything else. However a theorem about HJ alone is of limited use: in practice we are likely to be interested in $\mathrm{HJ}^{\supset}$ as a fragment of some larger theory. We know, for instance, that the deduction theorem follows for many extensions of HJ (e.g. extensions to larger fragments of intuitionistic or classical logic). The problem is that the induction principle we use to prove the theorem is equivalent to a closure assumption, and such an assumption means that we are not able to use the theorem with extensions.

We seem to have confirmed, from the other side, the trade-off we have documented above for $\to$-frameworks: either we can have induction and no theory structuring (as theories are 'closed'), or vice versa. However, if we look harder,

[30] An extension of the deduction theorem to first-order logic, however, is not trivial—we have to treat a new rule, which introduces complex side conditions to the statement of the theorem (Kleene[1952] discusses one way to do this).

there is sometimes a middle way that is possible in ID-frameworks. The crucial point is that an inductive argument does not always rely on *all* the closure assumptions of the most general case. Consider the assumptions that are made in the proof of (38):

- The proof of the base case relies on the fact that the axioms $a \supset b \supset a$ and $a \supset a$ are available in HJ.
- The proof of the step case relies on the fact that the *only* rule that can be applied is Det, and that the axiom

  $$(a \supset b) \supset (a \supset b \supset c) \supset a \supset c$$

  is available.

What bears emphasizing is that we do not need to assume that no axioms other than those explicitly mentioned are available, only that no rules other than Det are available.

We can take account of this observation to produce a more general version of (38)

$$b \in \mathrm{HJ}[\Gamma \cup \{a\}] \to (a \supset b) \in \mathrm{HJ}[\Gamma], \tag{40}$$

which we can still prove in the same way as (38). We call this version *open-ended* since it can be used with any axiomatic extension Γ of HJ. In particular (38) is just (40) where we take Γ to be $\emptyset$.

Structuring theories with the deduction theorem Unlike (38), we can make effective use of (40) in a hierarchy of theories in a way similar to what is possible in an $\to$-framework. The metatheorem can be applied to any extension $\mathrm{HJ}[\Gamma]$ where Γ is a collection of axioms. The fact that in the $\to$-framework we can add new rules, not just new axioms, is not as significant as it at first appears, so long as we have that

$$T(A) \to T(B) \quad \text{iff} \quad T(A \supset B) \tag{41}$$

since we can use this to find an axiomatic equivalent of any rule schema built from $\to$ and T in terms of $\supset$.

The above observation, of course, only holds for theories that can be defined in terms of a single predicate T and which include a connective $\supset$ for which (41) is true.[31]

[31] And, of course, some encodings use more than one metalevel predicate; e.g. in §5.4 we introduce a second predicate V for which there is no equivalent of (41). For these systems we have rules for which no axiomatic equivalent is available. This does not, however, mean that ID-frameworks are necessarily less effective for structuring collections of theories; it just means that we have to be more sophisticated in the way we exploit (41). See, e.g., Matthews [1997b] for discussion of how we can do this by introducing an 'extra' layer between the ID-framework and the theory to be encoded.

A further generalization of the deduction theorem

We arrived at the open-ended (40) by observing that other axioms could be present. And as previously observed, no such generalization is possible with arbitrary rules, e.g., the deduction theorem does not hold in K (which requires extensions by rules, as opposed to axioms). However, a more refined analysis of the step case of the proof is possible, and this leads to a further generalization of our metatheorem.

In the step case we need precisely that the theory is closed under

$$\frac{A \supset B \quad A \supset C}{A \supset D}$$

for each instance of a basic rule

$$\frac{B \quad C}{D}.$$

In the case of Det (the only rule in $\mathrm{HJ}^{\supset}$) we can show this by a combination of Det and the S axiom.

Using our ID-framework we can explicitly formalize these statements as part of the deduction theorem itself, proving a further generalization. If we extend the notation of (37) with a parameter Δ for rules, i.e.,

$$\mathrm{HJ}[\Gamma, \Delta] \equiv I(K \cup S \cup \Gamma, Det \cup \Delta) \tag{42}$$

then for the base case we have

$$b \in \mathrm{HJ}[\Gamma \cup \{a\}, \Delta] \to (a \supset b) \in \mathrm{HJ}[\Gamma, \Delta] \tag{43}$$

and

$$(a \supset a) \in \mathrm{HJ}[\Gamma, \Delta] \tag{44}$$

while for the step case

$$\begin{aligned}(d, b, c) \in \Delta \to (a \supset b) \in \mathrm{HJ}[\Gamma, \Delta] \\ \to (a \supset c) \in \mathrm{HJ}[\Gamma, \Delta] \to (a \supset d) \in \mathrm{HJ}[\Gamma, \Delta]\,.\end{aligned} \tag{45}$$

The formulae (43) and (44) follow immediately for any $\mathrm{HJ}[\Gamma, \Delta]$, but (45) isn't always true. Thus, our third deduction theorem has the form

$$(45) \to b \in \mathrm{HJ}[\Gamma \cup \{a\}, \Delta] \to (a \supset b) \in \mathrm{HJ}[\Gamma, \Delta]\,, \tag{46}$$

which can be proved in the same way as (40). Note too that this metatheorem generalizes (40), since (40) is just (38) where Δ is $\emptyset$ and the antecedent, which is therefore true, has been removed.

The deduction theorem can even be further generalized, but doing so would take us too far afield. In [Basin and Matthews, 2000] we show how a further generalization of (46) can be specialized to modal logics that extend $\mathrm{S4}$. This generalization allows us to prove

$$b \in \mathrm{S4}[\Gamma \cup \{a\}] \to (\Box a \supset b) \in \mathrm{S4}[\Gamma].$$

That is, in $\mathrm{S4}$ we can prove a deduction theorem that allows us to reason under 'boxed' assumptions $\Box a$.

7.5 Admissible and derived rules

Our examples suggest that inductive definitions offer considerable power and simplicity in organizing metatheories. Each metatheorem states the conditions an extension has to satisfy for it to apply; so once proved, we need only check these conditions before making use of it. Most metatheorems require only that certain axioms and rules are available and therefore hold in all extensions with additional axioms and rules. Others depend on certain things being absent (e.g. rules that do not satisfy certain properties, in the case of the deduction theorem); in such cases, we can prove more restricted theorems that are still usable in appropriate extensions.

How does this kind of metatheory compare with what is possible in theorem provers supporting hierarchical theories? We begin by reviewing the two standard notions of proof-rules. Our definitions are those of Troelstra [1982, § 1.11.1] translated into our notation, where $\mathcal{T}[\Gamma, \Delta]$ is a deductive system $\mathcal{T}$ extended with sets of axioms Γ and rules Δ, e.g. (42).

Fix a language of formulae. A *rule* is an $n+1$-ary relation over formulae $\langle F_1, \ldots, F_n, F_{n+1}\rangle$ where the $F_1, \ldots, F_n$ are the *premises* and F_{n+1} the *conclusion.* A rule is *admissible* for $\mathcal{T}$ iff

$$\vdash_{\mathcal{T}[\emptyset,\emptyset]} F_1 \to \cdots \to \vdash_{\mathcal{T}[\emptyset,\emptyset]} F_n \to \vdash_{\mathcal{T}[\emptyset,\emptyset]} F_{n+1}\,, \qquad (\textit{adm})$$

and *derivable* for $\mathcal{T}$ iff

$$\forall \Gamma.\, \vdash_{\mathcal{T}[\Gamma,\emptyset]} F_1 \to \cdots \to \vdash_{\mathcal{T}[\Gamma,\emptyset]} F_n \to \vdash_{\mathcal{T}[\Gamma,\emptyset]} F_{n+1}\,. \qquad (\textit{der})$$

It follows immediately from the definitions that derivability implies admissibility; however, the converse does not always hold. It is easy to show that Troelstra's definition of derivability is equivalent to that of Hindley and Seldin [1986]; i.e. $\vdash_{\mathcal{T}[\{F_1,\ldots,F_n\},\emptyset]} F_{n+1}$, and that if a rule is derivable it holds in all extensions of $\mathcal{T}$ with new axioms and rules.

Whereas in $\to$-frameworks we can only prove derived rules, logical frameworks based on inductive definitions allow us to prove that rules are admissible, as well as reason about other kinds of rules not fitting the above categories. For example, the languages or deductive systems for the F_i can be different, like in the various

versions of the deduction theorem that we have formalized; our deduction theorems are neither derived nor admissible since their statements involve different deductive systems.

7.6 Problems with ID-frameworks

Our examples provide evidence that a framework based on inductive definitions can serve as an adequate foundation for carrying out metatheory in the proof theory tradition and can be used to structure metatheoretic development. However, some aspects of formal metatheory are more difficult than with an $\rightarrow$-framework. The most fundamental difficulty, and one that is probably already clear from our discussion in this section, is the way languages are encoded. This is quite primitive in comparison to what is possible in an $\rightarrow$-framework: for the propositional examples that we have treated here, the view of a language as a recursively enumerable class is direct and effective. But this breaks down for logics with quantifiers and other variable binding operators where the natural equivalence relation for syntax is no longer identity but equivalence under the renaming of bound variables (α-congruence). We have shown (in §4.2) that language involving binding has a natural and direct treatment in an $\rightarrow$-framework as higher-order syntax. Nothing directly equivalent is available in an ID-framework; we are forced to build the necessary facilities ourselves.

Since the user must formalize many basic syntactic operations in an ID-framework, any treatment of languages involving variable binding operators will be more 'primitive' than what we get in an $\rightarrow$-framework, but how much more primitive is not clear. So far, most experience has been with *ad hoc* implementations of binding (e.g. [Matthews, 1992]) but approaches that are both more sophisticated and more modular are possible, such as the binding structures proposed by Talcott [1993], a generalization of de Bruijn indices as an algebra. As yet, we do not know how effective such notations are.

The other property of ID-frameworks that might be criticized is that they are biased towards Hilbert calculi, which are recognized to be difficult to use. But metatheorems, in particular the deduction theorem, can play an important role in making Hilbert calculi usable in practice. And, if a Hilbert style presentation is not suitable, it may be possible to exploit a combination of the deduction theorem and the intuitions of $\rightarrow$-frameworks to provide direct encodings of natural deduction [Matthews, 1997b]. The same provisos about lack of experience with effective notations for handling bindings apply here though, since this work only discusses the propositional case.

8 CONCLUSIONS

This chapter does not try to give a final answer to the question of what a logical framework might be. Rather it argues that the question is only meaningful in terms

of some particular set of requirements, and in terms of 'goodness of fit'; i.e. the relationship between the properties of a proposed metalogic and logics we want to encode.

Our central theme has been the relationship between different kinds of deductive systems and their abstractions as metatheories or metalogics, which we can use to encode and work with instances of them. We have showed that a logic of minimal implication and universal quantification can be used to encode both the language and the proof-rules of a natural deduction or sequent calculus, and then described in detail a particular logic of this type, λ^{p}. As a contrast, we have also considered how (especially Hilbert) calculi can be abstracted as inductive definitions and we sketched a framework based on this view, FS_0. We then used the metatheoretic facilities that we get with an ID-framework like FS_0 to explore the relationship between metatheory and object theory, especially in the context of structured collections of theories.

The simple binary distinction between ID and $\rightarrow$-frameworks, which we make for the sake of space and explication, of course does not describe the whole range of frameworks that have been proposed and investigated. It does, however, help to define the space of possibilities and current research into frameworks can mostly be categorized in its terms.

For instance, research into the problem of the 'goodness of fit' relation between the metalogic and the object logic, especially for $\rightarrow$-frameworks, can be separated into two parts. We have shown that natural deduction calculi for standard mathematical (i.e., classical or intuitionistic, first or higher-order) logics fit well into an $\rightarrow$-framework. But the further we diverge from standard, mathematical, logics into philosophical (i.e. modal, relevance, etc.) logic the more complex and artificial the encodings become. In order to encode modal logics, for instance, we might introduce either multiple-judgment encodings or take explicit account of a semantics via a labelling. The particular encodings that we have described here are only some among a range of possibilities that could be imagined. A more radical possibility, not discussed here, can be found in, e.g., [Matthews, 1997a], where the possibility of extending a framework directly with a modal 'validity' connective is explored. However we do not yet know what the practical limits of these approaches are.[32] A similar problem of goodness of fit is also encountered in ID-frameworks, where the 'natural' deductive systems are Hilbert calculi and the 'obvious' encodings of consequence style systems are impractically unwieldy. Matthews [1997b] suggests how we might encode pure ordinary consequence relation based systems in such a framework in a way that is more effective than, and at least as intuitive as, the 'naive' approach of using inductively defined classes.

The particular problems of substructural logics (e.g. linear or relevance logics) have been the subject of substantial research. The consequence relations associated with these logics are not ordinary and hence cannot be encoded using tech-

[32] This is essentially a practical, not a theoretical question, since, as we pointed out earlier, an $\rightarrow$-framework can be used as a Turing complete programming language, so with sufficient ingenuity any deductive system can be encoded.

niques such as those suggested in §6. While labelled or multiple judgment presentations of these logics in an $\rightarrow$-framework are possible, they seem to be unwieldy; i.e. they 'fit' particularly badly. An alternative approach has been explored where the framework itself is modified: minimal implication is replaced or augmented in the framework logic with a substructural or linear implication, which does permit a good fit. In Ishtiaq and Pym [1998] and Cervesato and Pfenning [1996], systems are presented that are similar to λ^{p} except that they are based on linear implication, which is used to encode variants of linear and other relevance logics. There is also work that, rather than employing either $\rightarrow$ or ID-frameworks, attempts to combine features of both (e.g. McDowell and Miller [1997] and Despeyroux *et al.* [1996]).

In short, then, there are many possibilities and, in the end, no absolute solutions: the suitability of a particular logical framework to a particular circumstance depends on empirical as well as theoretical issues; i.e. before we can choose we have to decide on the range of object logics we envision formalizing, the nature of the metatheoretic facilities that we want, and the kinds of compromises that we are willing to accept.

David Basin
University of Freiburg, Germany.

Seán Matthews
IBM Unternehmensberatung GmbH, Frankfurt, Germany.

BIBLIOGRAPHY

[Abadi *et al.*, 1991] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Functional Programming*, 1:375–416, 1991.

[Aczel, 1977] Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, Amsterdam, 1977.

[Avron *et al.*, 1992] Arnon Avron, Furio Honsell, Ian Mason, and Robert Pollack. Using typed lambda calculus to implement formal systems on a machine. *J. Auto. Reas.*, 9:309–352, 1992.

[Avron *et al.*, 1998] Arnon Avron, Furio Honsell, Marino Miculan, and Cristian Paravano. Encoding modal logics in logical frameworks. *Studia Logica*, 60(1):161–208, 1998.

[Avron, 1990] Arnon Avron. Gentzenizing Shroeder-Heister's natural extension of natural deduction. *Notre Dame Journal of Formal Logic*, 31:127–135, 1990.

[Avron, 1991] Arnon Avron. Simple consequence relations. *Inform. and Comput.*, 92:105–139, 1991.

[Avron, 1992] Arnon Avron. Axiomatic systems, deduction and implication. *J. Logic Computat.*, 2:51–98, 1992.

[Barendregt, 1984] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 2nd (revised) edition, 1984.

[Barendregt, 1991] Henk Barendregt. Introduction to generalized type sytems. *J. Functional Programming*, 2:125–154, 1991.

[Barendregt, 1992] Henk Barendregt. Lambda calculi with types. In Samson Abramsky, Dov Gabbay, and Tom S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.

[Basin and Matthews, 2000] David Basin and Sean Matthews. Structuring metatheory on inductive definitions. *Information and Computation*, 162(1–2), October/November 2000.

[Basin *et al.*, 1997a] David Basin, Seán Matthews, and Luca Viganò. Labelled propositional modal logics: theory and practice. *J. Logic Computat.*, 7:685–717, 1997.

[Basin *et al.*, 1997b] David Basin, Seán Matthews, and Luca Viganò. A new method for bounding the complexity of modal logics. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Proc. Kurt Gödel Colloquium, 1997*, pages 89–102. Springer, Berlin, 1997.

[Basin *et al.*, 1998a] David Basin, Seán Matthews, and Luca Viganò. Labelled modal logics: Quantifiers. *J. Logic, Language and Information*, 7:237–263, 1998.

[Basin *et al.*, 1998b] David Basin, Seán Matthews, and Luca Viganò. Natural deduction for non-classical logics. *Studia Logica*, 60(1):119–160, 1998.

[Boolos, 1993] George Boolos. *The Logic of Provability*. Cambridge University Press, 1993.

[Boyer and Moore, 1981] Robert Boyer and J. Strother Moore. *A Computational Logic*. Academic Press, New York, 1981.

[Bull and Segerberg, 1984] Robert Bull and Krister Segerberg. Basic modal logic. In Gabbay and Guenthner [1983–89], chapter II.1.

[Cervesato and Pfenning, 1996] Iliano Cervesato and Frank Pfenning. A linear logical framework. In *11th Ann. Symp. Logic in Comp. Sci.* IEEE Computer Society Press, 1996.

[Church, 1940] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[Davis, 1989] Martin Davis. Emil Post's contributions to computer science. In *Proc. 4th IEEE Ann. Symp. Logic in Comp. Sci.* IEEE Computer Society Press, 1989.

[de Bruijn, 1972] Nicolas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[de Bruijn, 1980] Nicolas G. de Bruijn. A survey of the project Automath. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, New York, 1980.

[Despeyroux *et al.*, 1996] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. Technical Report CMU-CS-96-172, Dept. of Computer Science, Carnegie Mellon University, September 1996.

[Dummett, 1978] Michael Dummett. The philosophical basis of intuitionistic logic. In *Truth and other enigmas*, pages 215–247. Duckworth, London, 1978.

[Fagin *et al.*, 1992] Ronald Fagin, Joseph Y. Halpern, and Moshe Y. Vardi. What is an inference rule? *J. Symbolic Logic*, 57:1018–1045, 1992.

[Feferman, 1990] Solomon Feferman. Finitary inductive systems. In *Logic Colloquium '88*, pages 191–220. North-Holland, Amsterdam, 1990.

[Felty, 1989] Amy Felty. *Specifying and Implementing Theorem Provers in a Higher Order Programming Language*. PhD thesis, University of Pennsylvania, 1989.

[Felty, 1991] Amy Felty. Encoding dependent types in an intuitionistic logic. In Huet and Plotkin [1991].

[Gabbay and Guenthner, 1983–89] Dov Gabbay and Franz Guenthner, editors. *Handbook of Philosophical Logic, vol. I–IV*. Reidel, Dordrecht, 1983–89.

[Gabbay, 1996] Dov Gabbay. *Labelled Deductive Systems, vol. 1*. Clarendon Press, Oxford, 1996.

[Gardner, 1995] Philippa Gardner. Equivalences between logics and their representing type theories. *Math. Struct. in Comp. Science*, 5:323–349, 1995.

[Gentzen, 1934] Gerhard Gentzen. Untersuchen über das logische Schließen. *Math. Z.*, 39:179–210, 405–431, 1934.

[Gödel, 1931] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatsh. Math.*, 38:173–198, 1931.

[Goguen and Burstall, 1992] Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specifications and programming. *J. Assoc. Comput. Mach.*, pages 95–146, 1992.

[Gordon and Melham, 1993] Michael J. Gordon and Tom Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, Cambridge, 1993.

[Gordon *et al.*, 1979] Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. Springer, Berlin, 1979.

[Griswold, 1981] Ralph E. Griswold. A history of the Snobol programming languages. In Wexelblat [1981], pages 601–660.

[Hacking, 1979] Ian Hacking. What is logic? *J. Philosophy*, 76(6), 1979.

[Harper *et al.*, 1993] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. Assoc. Comput. Mach.*, 40:143–184, 1993.

[Hindley and Seldin, 1986] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*. Cambridge University Press, Cambridge, 1986.

[Howard, 1980] William Howard. The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus, and Formalism*. Academic Press, New York, 1980.

[Huet and Plotkin, 1991] Gérard Huet and Gordon Plotkin, editors. *Logical Frameworks*. Cambridge University Press, Cambridge, 1991.

[Ishtiaq and Pym, 1998] Samin Ishtiaq and David Pym. A relevant analysis of natural deduction. *J. Logic Computat.*, 8:809–838, 1998.

[Kleene, 1952] Stephen C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

[Kripke, 1963] Saul A. Kripke. Semantical analysis of modal logic I: normal propositional modal logic. *Z. Math. Logik Grundlag. Math*, 8:67–96, 1963.

[Martí-Oliet and Meseguer, 2002] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In *Handbook of Philosophical Logic*. second edition, 2002.

[Matthews *et al.*, 1993] Seán Matthews, Alan Smaill, and David Basin. Experience with FS_0 as a framework theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Environments*. Cambridge University Press, Cambridge, 1993.

[Matthews, 1992] Seán Matthews. *Metatheoretic and Reflexive Reasoning in Mechanical Theorem Proving*. PhD thesis, University of Edinburgh, 1992.

[Matthews, 1994] Seán Matthews. A theory and its metatheory in FS_0. In Dov Gabbay, editor, *What is a Logical System?* Clarendon Press, Oxford, 1994.

[Matthews, 1996] Seán Matthews. Implementing FS_0 in Isabelle: adding structure at the metalevel. In Jacques Calmet and Carla Limongelli, editors, *Proc. Disco'96*. Springer, Berlin, 1996.

[Matthews, 1997a] Seán Matthews. Extending a logical framework with a modal connective for validity. In Martín Abadi and Takayasu Ito, editors, *Proc. TACS'97*. Springer, Berlin, 1997.

[Matthews, 1997b] Seán Matthews. A practical implementation of simple consequence relations using inductive definitions. In William McCune, editor, *Proc. CADE–14*. Springer, Berlin, 1997.

[McCarthy, 1981] John McCarthy. History of Lisp. In Wexelblat [1981], pages 173–197.

[McDowell and Miller, 1997] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax. In *Proc. 12th IEEE Ann. Symp. Logic in Comp. Sci.*, pages 434–446. IEEE Computer Society Press, 1997.

[Meseguer, 1989] José Meseguer. General logics. In Heinz-Dieter Ebbinghaus, J. Fernandez-Prida, M. Garrido, D. Lascar, and M. Rodriguez Artalejo, editors, *Logic Colloquium, '87*, pages 275–329. North-Holland, 1989.

[Nederpelt *et al.*, 1994] Rob P. Nederpelt, Herman J. Geuvers, and Roel C. de Vrijer, editors. *Selected papers on Automath*. Elsevier, Amsterdam, 1994.

[Ohlbach, 1993] Hans-Jürgen Ohlbach. Translation methods for non-classical logics: an overview. *Bulletin of the IGPL*, 1:69–89, 1993.

[Owre *et al.*, 1995] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. Software Eng.*, 21:107–125, 1995.

[Paulson, 1994] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, Berlin, 1994.

[Pfenning, 1996] Frank Pfenning. The practice of logical frameworks. In Helene Kirchner, editor, *Proc. CAAP'96*. Springer, Berlin, 1996.

[Pfenning, 2000] Frank Pfenning. Structural cut elimination I. intuitionistic and classical logic. *Information and Computation*, 157(1–2), March 2000.

[Pollack, 1994] Randy Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994.

[Post, 1943] Emil Post. Formal reductions of the general combinatorial decision problem. *Amer. J. Math.*, 65:197–214, 1943.

[Prawitz, 1965] Dag Prawitz. *Natural Deduction*. Almqvist and Wiksell, Stockholm, 1965.

[Prawitz, 1971] Dag Prawitz. Ideas and results in proof theory. In J. E. Fensted, editor, *Proc. Second Scandinavian Logic Symp.*, pages 235–307. North-Holland, Amsterdam, 1971.

[Pym and Wallen, 1991] David J. Pym and Lincoln Wallen. Proof-search in the λΠ-calculus. In Huet and Plotkin [1991], pages 309–340.

[Schroeder-Heister, 1984a] Peter Schroeder-Heister. Generalised rules for quantifiers and the completeness of the intuitionistic operators $\&$, $\vee$, $\supset$, $\bot$, $\forall$, $\exists$. In M. M. Richter et al., editors, *Computation and proof theory*. Springer, Berlin, 1984.

[Schroeder-Heister, 1984b] Peter Schroeder-Heister. A natural extension of natural deduction. *J. Symbolic Logic*, 49:1284–1300, 1984.

[Scott, 1974] Dana Scott. Rules and derived rules. In S. Stenlund, editor, *Logical Theory and Semantical Analysis*, pages 147–161. Reidel, Dordrecht, 1974.

[Simpson, 1992] Alex K. Simpson. Kripke semantics for a logical framework. In *Proc. Workshop on Types for Proofs and Programs*, Båstad, 1992.

[Smullyan, 1961] Raymond Smullyan. *Theory of Formal Systems*. Princeton University Press, 1961.

[Steele Jr. and Gabriel, 1996] Guy L. Steele Jr. and Richard P. Gabriel. The evolution of Lisp. In Thomas J. Bergin and Richard G. Gibson, editors, *History of Programming Languages*, pages 233–330. ACM Press, New York, 1996.

[Sundholm, 1983] Göran Sundholm. Systems of deduction. In Gabbay and Guenthner [1983–89], chapter I.2.

[Sundholm, 1986] Göran Sundholm. Proof theory and meaning. In Gabbay and Guenthner [1983–89], chapter III.8.

[Talcott, 1993] Carolyn Talcott. A theory of binding structures, and applications to rewriting. *Theoret. Comp. Sci.*, 112:99–143, 1993.

[Troelstra, 1982] A. S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. Springer, Berlin, 1982.

[van Benthem, 1984] Johan van Benthem. Correspondence theory. In Gabbay and Guenthner [1983–89], chapter II.4.

[Wexelblat, 1981] Richard L. Wexelblat, editor. *History of Programming Languages*. Academic Press, New York, 1981.

GÖRAN SUNDHOLM

PROOF THEORY AND MEANING

Dedicated to Stig Kanger on the occasion of his 60th birthday

The meaning of a sentence determines how the truth of the proposition expressed by the sentence may be proved and hence one would expect proof theory to be influenced by meaning-theoretical considerations. In the present chapter we consider a proposal that also reverses the above priorities and determines meaning in terms of proof. The proposal originates in the criticism that Michael Dummett has voiced against a realist, truth-theoretical, conception of meaning and has been developed largely by him and Dag Prawitz, whose normalisation procedures in technical proof theory constitute the main technical basis of the proposal.

In a subject not more than 20–30 years old, and were much work is currently being done, any survey is bound to be out of date when it appears. Accordingly I have attempted not to give a large amount of technicalities, but rather to present the basic underlying themes and guide the reader to the ever-growing literature. Thus the chapter starts with a general introduction to meaning-theoretical issues and proceeds with a fairly detailed presentation of Dummett's argument against a realist, truth-conditional, meaning theory. The main part of the chapter is devoted to a consideration of the alternative proposal using 'proof-conditions', instead of truth-conditions, as the key concept. Finally, the chapter concludes with an introduction to the type theory of Martin-Löf.

I am indebted to Professors Dummett, Martin-Löf and Prawitz, and to my colleague Mr. Jan Lemmens, for many helpful conversations on the topics covered herein and to the editors for their infinite patience. Dag Prawitz and Albert Visser read parts of the manuscript and suggested many improvements.

1 THEORIES OF MEANING, MEANING THEORIES AND TRUTH THEORIES

A *theory of meaning* gives, one might not unreasonably expect, a general account of, or view on, the very concept of meaning: what it is and how it functions. Such theories *about* meaning, however, do not hold undisputed rights to the *appellation*; in current philosophy of language one frequently encounters discussions of theories of meaning for particular languages. Their task is to specify the meaning of all the sentences of the language in question. Following Peacocke [1981] I shall use the term 'meaning theory' for the latter, language-relative, sort of theory and reserve 'theory of meaning' for

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, 165–198.

the former. Terminological confusion is, fortunately, not the only connection between meaning theories and theories of meaning. On th contrary, the main reason for the study and attempted construction of meaning theories is that one hopes to find a correct theory of meaning through reflection on the various desiderata and constraints that have to be imposed on a satisfactory meaning theory. The study of meaning theories, so to speak, provides the *data* for the theory of meaning. In the present chapter we shall mainly treat meaning theories and some of their connection with (technical) proof theory and, consequently, we shall only touch on the theory of meaning in passing. (On the other hand the whole chapter can be viewed as a contribution to the theory of meaning.)

There is, since Frege, a large consensus that the sentence, rather than the word, is the primary bearer of (linguistic) meaning. The sentence is the least unit of language that can be used to *say* anything. Thus the theory of meaning directs that sentence-meaning is to be central in meaning theories and that word-meaning is to be introduced derivatively: the meaning of a word is the way in which the work contributes to the meaning of the sentences in which it occurs. It is natural to classify the sentences of a language according to the sort of linguistic act a speaker would perform through an utterance of the sentence in question, be it an assertion, a question or a command. Thus, in general, the meaning of a sentence seems to comprise (at least) two elements, because to know the meaning of — in order to understand an utterance of — the sentence in question one would have to know, first to what category the sentence belongs, i.e. one would have to know what sort of linguistic act that would be performed through an utterance of the sentence, and secondly one would have to know the *content* of the act.

This diversity of sentence-meaning, together with the idea that word-meaning is to be introduced derivatively (as a way of contributing to sentence-meaning), poses a certain problem for the putative meaning-theorist. If sentences from different categories have different *kinds* of meaning, it appears that the meaning of a word will vary according to the category of the sentences in which it occurs: uniform word-meanings are ruled out. But this is unacceptable as anyone familiar with a dictionary knows. The word 'door', say, has the same meaning in the three sentences 'Is the door open?', 'The door is open.', and 'Open the door!'. This *prima facie* difficulty is turned into a tool for investigating what internal structure ought to be imposed on a satisfactory meaning theory.

A meaning theory will have to comprise at least two parts: the *theory of sense* and the *theory of force*. The task of the latter is to identify the sort of act performed through an utterance of a sentence and the former has to specify the content of the acts performed. In order to secure the uniformity of word meaning the theory of sense has to be formulated in terms of some on *key concept*, in terms of which the content of all sentences is to be given,

and the theory of force has to provide uniform, general, principles relating speech act to content. The meaning of a word is then taken as the way in which the word contributes to the content of the sentences in which it occurs (as given by the key concept in the theory of sense).

The use of such a notion of key concept also allows the meaning theories to account for certain (iterative) unboundedness-phenomena in language, e.g. that whenever A and B are understood sentences, then also 'A and B' would appear to be meaningful. This is brought under control in the meaning theory by expressing the condition for the application of the key concept P to 'A and B' in terms of P applied to A and P applied to B.

The most popular candidate for a key concept has undeniably been *truth*: the content of a sentence is given by its 'truth-condition'. One can, indeed, find many philosophers who have subscribed to the idea that meaning is to be given in terms of truth. Examples would be Frege, Wittgenstein, Carnap, Quine and Montague. It is doubtful, however, if they would accept that the way in which truth serves to specify meaning is as a key concept in a meaning theory (that is articulated into sense and force components respectively). Such a conception of the relation between meaning and truth has been advocated by Donald Davidson, who, in an important series of papers, starting with [1967], and now conveniently collected in his [1984], has proposed and developed the idea that meaning is to be studied via meaning theories. Davidson is quite explicit on the role of truth. It is going to take its rightful place within the meaning theory in the shape of a truth theory in the sense of Tarski [1956, Ch. VIII]. Tarski showed, for a given formal language L, how to define a predicate '$\mathrm{True}_{\mathrm{L}}(x)$' such that for every sentence S of L it is provable from the definition that

(1) $\mathrm{True}_{\mathrm{L}}(\bar{S})$ iff $f(S)$.

Here '$\bar{S}$' is a *name* of, and $f(S)$ a *translation* of, the object-language sentence W in the language of the meta-theory (= the theory in which the truth definition is given and where all instances of (1) must hold). *Using* the concept of meaning (in the guise of 'translation' from object-language to meta-language) Tarski gave a precise definition of what it is for a sentence of L to be true. Davidson reverses the theoretical priorities. Starting with a *truth theory* for L, that is a theory the language of which contains $\mathrm{True}_{\mathrm{L}}(x)$ as a primitive, and where for each sentence S of L

(2) $\mathrm{True}_{\mathrm{L}}(\bar{S})$ iff p.

holds for some sentence p of the language of the truth theory, he wanted to extract meaning from truth. Simply to consider an arbitrary truth theory will not do not capture meaning, though. It is certainly true that

(3) $\overline{\text{Snow is white}}$ is true-in-English iff snow is white

but, unquestionably and unfortunately, it is equally true that

(4) $\overline{\text{Snow is white}}$ is true-in-English iff grass is green

and the r.h.s. of (4) could not possibly by any stretch of imagination be said to provide even a rough approximation of the meaning of the English sentence

Snow is white.

Furthermore, a theory that had all instances of (2) as axioms would be unsatisfactory also in that it used infinitely many unrelated axioms; the theory would, it is claimed, be 'unlearnable'.

Thus one might attempt to improve on the above simple-minded (2) by considering truth theories that are formulated in a meta-language that contains the object-language and that give their 'T-theories' (the instances of (2)), not as axioms, but as derivable from homophonic recursion clauses, e.g.

(5) for all $\bar{A}$ and $\bar{B}$ of L,
$\text{True}_{\text{L}}(\overline{A \text{ and } B}$ iff $\text{True}_{\text{L}}(\bar{A}$ and $\text{True}_{\text{L}}(\bar{B})$

and

(6) for all $\bar{A}$ of L,
$\text{True}_{\text{L}}(\overline{\text{not-}A}$ iff not-$\text{True}_{\text{L}}(\bar{A})$.

Here one uses the word mentioned in the sentence on the l.h.s. when giving the condition for its truth on the r.h.s.; cf. the above remarks on the iterative unboundedness phenomena.

The treatment of quantification originally used Tarski's device of 'satisfaction relative to assignment by sequences', where, in fact, one does not primarily recur on truth, but on satisfaction, and where truth is defined as satisfaction by all sequences. The problem which Tarski solved by the use of the sequences and the auxiliary notion of satisfaction was how to capture the right truth condition for 'everything is A' even though the object language does not contain a name for everything to be considered in the relevant domain of quantification. Another satisfactory solution which goes back to Frege, would be to use quantification over finite extensions L^{+} of L by mans of new names. The interested reader is referred to [Evans, 1977, Section 2] or to [Davies, 1981, Chapter VI] for the (not too difficult) technicalities. A very extensive and careful canvassing of various alternative approaches to quantificational truth-theories is given by Baldwin [1979]. If we bypass the problem solve by Tarski and consider, say, the language of arithmetic, where the problem does not arise as the language contains a numeral for each element of the intended domain of quantification the universal-quantifier clause would be

(7) for all $\bar{A}$ of L,
$\mathrm{True}_{\mathrm{L}}(\overline{\text{for every number } x, A(x)})$ iff for every numeral $\bar{k}$,
$\mathrm{True}_{\mathrm{L}}(\overline{A(\bar{k}/x)})$.

(here '$A(\bar{k}/x)$' indicates the result of substituting the numeral k for the variable x.)

Unfortunately it is still not enough to consider these homophonic, finitely axiomatised truth theories in order to capture meaning. The basic clauses of a homophonic truth theory will have the form, say,

(8) for any name $\bar{t}$ of L,
$\mathrm{True}_{\mathrm{L}}(\overline{t \text{ is red}})$ iff whatever t refers to is red.

If we now change this clause to

(9) for any $\bar{t}$ in L,
$\mathrm{True}'_{\mathrm{L}}(\overline{t \text{ is red}})$ iff whatever t refers to is red and grass is green

and keep homophonic clauses for $\mathrm{True}'_{\mathrm{L}}$ with respect to 'and' 'not', etc., the result will still be a finitely axiomatised and correct ('true') truth theory for L. We could equally well have chosen any other true contingent sentence instead of 'grass is green'.) Seen from the perspective of 'real meaning' the truth condition of the primed theory is best explained as

(10) $\mathrm{True}'_{\mathrm{L}}(\bar{S})$ iff S and grass is green.

The fact that a true, finitely axiomatised, homophonic truth-theory does not necessarily provide truth conditions that capture meaning was first observed by Foster and Loar in 1976. Various remedies and refinements of the original Davidsonian programme have been explored. We shall briefly consider an influential proposal due to John McDowell [1976; 1977; 1978].

The above attempts to find a meaning theory via truth start with a (true) truth theory and go on to seek further constraints that have to be imposed in order to capture meaning. McDowell, on the other hand, reverses this strategy and starts by considering a satisfactory theory of sense. Such a theory has to give content-ascriptions to the sentences $\bar{S}$ of the language L, say in the general form

(11) $\bar{S}$ is Q iff p,

where p is a sentence of the meta-language that gives the content of $\bar{S}$, and, furthermore, the theory has to interact with a theory of force in such a way that the interpreting descriptions, based on the contents as assigned in (11), do in fact make sense of what speakers say and do when they utter sentences containing $\bar{S}$. A meaning theory, and thus also its theory

of sense, is part of an overall theory of understanding, the task of which is to make sense of human behaviour (and not just these speech-acts). If the theory of sense can serve as a content-specifying core in such a general theory, then (11) guarantees that the predicate Q is (co-extensional with) truth. But not only that is true; the pathological truth-theories that were manufactured for use in the Foster–Loar counter-examples are ruled out from service as theories of sense because their use would make the meaning theory issue incomprehensible, or outright false, descriptions of what people do. A theory of sense which uses a pathological truth-theory does not make sense. Thus we see that while an adequate theory of sense will be a truth theory, the opposite is false: not every truth theory for a language will be a theory of sense for the language.

In conclusion of the present section let us note the important fact that the Tarski homophonic truth-theories are completely neutral with respect to the underlying logic. The T-theorems are derivable from the basic homophonic recursion clauses using intuitionistic logic only (in fact even minimal logic will do).

No attempt has been made in the present section to achieve either completeness or originality. The very substantial literature on the Davidsonian programme is conveniently surveyed in two texts, [Platts, 1979] and [Davies, 1981], where the latter pays more attention to the (not too difficult) technicalities. Many of the important original papers are included in [Evans and McDowell, 1976], with an illuminating introduction by the editors, and [Platts, 1980], while mention has already been made of [Davidson, 1984]'s collection of essays.

2 INTERMEZZO: CLASSICAL TRUTH AND SEQUENT CALCULI

(Intended for readers of the method 'semantic tableaux', cf. Section 6 of Hodges' chapter or section 3 of Sundholm's chapter, both in Volume 1 of this *Handbook*.)

It is by now well-known that perhaps the easiest way to prove the completeness of classical predicate logic is to search systematically for a counter-model (or, more precisely, a falsifying 'semi-valuation', or 'model set') to the formula, or sequent, in question. This systematic search proceeds according to certain rules which are directly read off as necessary conditions from the relevant semantics. For instance, in order to falsify $\forall xA(x) \to B$, one needs to verify $\forall xA(x)$ and falsify B, and in order to verify $\forall xA(x)$ one has to verify $A(t)$ for every t, etc. Thus the rules for falsification, in fact, also concern rules for verification and *vice versa* (consider verification of, e.g. $\neg B$), and for each logical operator there will be two rules regulating the systematic search for a counter-model, one for verification and one for falsification. These rules turn out to be identical with Gentzen's [1934–1935] left and

right introduction rules for the same operators. In some cases the search needs to take alternatives into account, e.g. $A \to B$ is verified by falsifyng A or verifying B. Thus one has two possibilitieis. The failure of the search along a possibility is indicated by that the rules would force one to assign both truth and falsity to one and the same formula. This corresponds, of course, to the axioms of Gentzen's sequent calculi. This method, where failure of existence of counter-models is equivalent to existence of a sequent calculus proof-tree, was discovered independently by Beth, Hintikka, Kanger and Schütte in the 1950s and a brilliant exposition can be found in [Kleene, 1967, Chapter VI], whereas [Smullyan, 1968] is the canonical reference for the various ways of taking the basic insight into account. Prawitz [1975] is a streamlined development of the more technical aspects which provides an illuminating answer to the question as to why the rules that generate counter-models turn out to be identical with the sequent calculus rules. There one also finds a good introduction to the notion of semi-valuation which has begun to play a role in recent investigations into the semantics of natural language (cf. [van Benthem and van Eijck, 1982] for an interesting treatment of the connection between recent work on 'partial structures' in the semantics of natural language and the more proof-theoretical notions that derive from the 'backwards' completeness proofs).

These semantical methods for proving completeness also lend themselves to immediate proof-theoretical applications. The *Cut-free* sequent calculus is complete, but cut is a sound rule. Hence it is derivable. A connection with the topic of our chapter is forged by reversing these proof-theoretic uses of semantical methods. Instead of proving the completemess via semantics, one could start by postulating the completeness of a cut-free formalism, and *read off a semantic* from the left and right introduction rules. (Proof theory determines meaning.) Such an approach was suggested by Hacking [1979] in an attempt towards a criterion for logical constanthood. Unfortunately, his presentation is marred by diverse technical infelicities (cf. [Sundholm, 1981]), and the problem still remains open how to find a workable proposal along these lines.

3 DUMMETT'S ARGUMENT AGAINST A TRUTH-CONDITIONAL VIEW ON MEANING

In the present section I attempt to set out one version of an argument due to Michael Dummett to the effect that truth cannot adequately serve as a key concept in a satisfactory meaning theory. Dummett has presented his argument in many places (cf. the note at the end of the section) and the presentation I offer is not to be attributed to him. In particular, the emphasis on manifestation that can be found in the present version of Dummett's argument I have come to appreciate through the writings of Colin McGinn

[1980] and Crispin Wright [1976]. Dummett's most forceful exposition is still his [1976], which will be referred to as "WTM2".

Dummett's views on the role and function of meaning theories are only in partial agreement with those presented in section 1. The essential difference consists mainly in the strong emphasis on what it is to know a language that can be found in Dummett's writings, and as a consequence his meaning theories are firmly cast in an epistemological mould: "questions about meaning are best interpreted as qeustions of understanding: a dictum about what the meaning of a sentence consists is must be construed as a thesis about what is is to *know* its meaning" (WTM2, p. 69). The task of the meaing theoerist is to give a theeoretical (propositional) representation of the complex practical capacity one has when one knows how to speak a language. The knowledge that a speaker will have of the propositions that constitute the theoretical representation in question will, in the end, have to be *imp'licit* knowledge. Indeed, one cannot demand that a speaker should be able to articulate explicitly; those very principles that constitute the theoretical representation of his practical mastery. Thus a meaning theory that gives such a theoretical representation must also comprise a part that would state what it is to know the other parts implicitly.

The inner structure of a meaning theory that could serve the aims of Dummett will have to be different from the simple bipartite version considered in section 1. Dummett's meaning theories are to be structured as follows. There is to be (ia) a core *theory of semantic value*, which states the condition for the application of the key concept to the sentences of the language, and, furthermore, there must be (ii) a *theory of force*, as before. In between these two, however, there must be (ib) a *theory of sense*, whose task it is to state what it is to know what is stated in the theory of semantic value, i.e. what it is to know the condition for the application of the key concept to a sentence. Thus the theory of sense in the proposals from section 1 does not correspond to the theory of sense$_D$ — 'D' for Dummett — but to the theory of semantic value. (The Fregean origin of Dummett's tripartite structure should be obvious. For further elaboration cf., his [1981].) The theory of sense$_D$ has no matching part in the theories from section 1. The corresponding match is as follows:

	Dummett		*(Davidson-)McDowell*
(ia)	Theory of semantic value (applies key concept to sentences)	(i)	Theory of sense
(iv)	Theory of sense$_D$ (states what it is to know the theory of sematnc value)		
(ii)	The theory of force	(ii)	The theory of force

This difference is what lies at the heart of the matter in the discussion between Dummett and McDowell of whether a theory of meaning ought to be 'modest' or 'fullblooded' (cf. [McDowell, 1977, Section X]: should one demand that the mreaning theory must give a link-up with practical capacities *independently of, and prior to, the theory of force?*.

One should also note here that the right home for the theory of sense$_D$ is not quite clear. Here I have made it part of the meaning theory. It could perhaps be argued that a statement of wherein knowledge of meaning consists is something that had better be placed within a theory of meaning rather than in a meaning theory. Dummett himself does not draw the distinction between meaning theories and theories of meaning and one can, it seems to me, find traces of *both* notions in what Dummett calls a 'theory of meaning'.

Dummett's argument against the truth-theoretical conception of meaning makes essential use of the assumption that the meaning theories must contain a theory of sense$_D$, which Dummett explicates in terms of how it can be manifested: since the knowledge is implicit, possession thereof can be construed only in terms of how one manifests tht knowledge. Furthermore, this implicit knowledge of meaning, or more precisely, of the condition for applying the key concept to individual sentences, must be *fully* manifested in use. This is Dummett's transformation of Wittgenstein's dictum that meaning is use. Two reasons can be offered (cf. [McGinn, 1980, p. 20]). First, knowledge is one of many propositional attitudes and these are, in general, only attributed to agents on the basis of how they are manifested. Secondly, and more importantly, we are concerned with (implicit) knowledge of *meaning* and meaning is, *par excellence*, a vehicle of (linguistic) communication. If there were some componenets of the implicit knowlege that did not become fully manifest in use, they could not matter for communication and so they would be superfluous.

It was already noted above that the Tarskian truth-theories are completely neutral with respect to the logical properties of truth. What laws are obeyed is determined by the logic that is applied in the meta-theory, whereas the T-clauses themselves offer no information on this point. Dummett's argument is brought to bear not so much against Tarskian truth as against the possibility that the key concept could be 'recognition-transcendent'. Classical, *bivalent* truth is characterised by the law of bivalence that every sentence is either true or false independently of our capacity to decide, or find out, whichever is the case. Thus, in general, the truth-conditions will be such that they can obtain without us recognising that they do. There are a number of critical cases which produce such undecidable truth-conditions. (It should be noted that 'undecidable' is perhaps not the best choice here with its connotations from recursive function theory.) Foremost among these is undoubtedly quantification over infinite or unbounded domains. Fermat's last theorem and the Reimann hypothesis are both famous examples from

mathematics and their form is pruely universal $\forall x A(x)$, with decidable matrix $A(x)$. An existential example would be, say, 'Somewhere in the universe there is a little green stone-eater'. Other sorts of examples are given by, respectively, counterfactual conditionals and claims about sentience in others, e.g. 'Ronald Reagan is in pain'. A fourth class is given by statements about (remote) past and future time, e.g. 'A city will be built here in a thousand years', or 'Two seconds before Brutus stabbed Casesare thirty-nine geese cackled on the Capitol'.

The knowledge one has of how to apply the key concept cannot in its entirety be statable, explicit knowledge and so the theory of sense_D will have to state, for at least some sentences, how one manifests knowledge of the condition for applying the key concept to them, in ways other than stating what one knows eplicitly. Let us call the class of these 'the non-statable fragment'. (Questions of the 'division of linguistic labour' may arise here. Is *the* fragment necessarily unique? Cf. [McGinn, 1980, p. 22].)

Assume now for a *reductio* that bivalent, possibly recognition-transcendent, truth-conditions can serve as key concept in a (Dummettian) meaning theory. Thus the theory of sense_D has to state how one fully manifests knowledge of possibly recognition-transcendent truth-conditions. The 'possibly' can be removed: there are sentences in the non-statable fragment with undecidable truth-conditions. In order to see this, remember the four classes of undecidable sentences that were listed above. Demonstrably, undecidable sentences are present in the language and they must be present already in the non-statable fragment, because "the existence of such sentences cnnot be due solely to the occurrence of sentences introduced by purely verbal explanations: a language all of whose sentences were decidable would continue to have this property when enriched by expressions so introduced" (WTM2, p. 81). An objection that may be (and has been) raised here is that one could start with a decidable fragment, e.g. the atomic sentences of arithmetic and get the undecidability through addition of new sentence-operators such as quantifiers. That is indeed so, but is not relevant here, where one *starts with a larger language* that, as a matter of fact contains undecidable sentences and then isolates a fragment within this language that also will have this property. Decidable sentences used for definitions could only provide decidable sentences and hence some of the sentences of the full language would be left out. Also it is not permissible to speak of adding, say, the quantifiers as their nature is *sub judice*: the meaning of a quantifier is not something independent of the rest of the language but, like any other word, its meaning is the way it contributes to the meaning of the sentences in which it occurs.

Now the argument is nearly at its end. The theory of sense_D would be *incomplete* in that it could not state what it is to manifest fully implicit knowledge of the recognition-transcendent truth-condition of an undecidable sentence. If the theory attempted to do this, an observational void would

exist without observational warrant. We, as theorists, would be guilty of theoretical slack in our theory, because we could never see the agents manifest their implicit knowledge in response to the truth-conditions obtaining (or not), because *ex hypothesi*, they obtain unrecognisably. The agents, furthermore, could not see them obtain and so, independently of whether or not the theorist can see them response, they cannot manifest their knowledge in response to the truth-condition. (This is a point where the division of linguistic labour may play a role.)

Before we proceed, it might be useful to offer a short schematic summary of Dummett's argument as set out above. (Page refrences in brackets are to WTM2.)

1. To understand a language is to have knowledge of meaning. (p. 69)

2. Knowledge of meaning must in the end be implicit knowledge. (. 70)

3. Hence the meaning theory must contain a part, call it theory of sense$_D$, that specifies 'in what having this knowledge consists, i.e. what counts as a manifestion of that knowledge. (pp. 70–71 and p. 127)

4. There are sentences in the language such that the speaker manifests his knowledge of their meaning in ways other than stating the meaning in other words. (The non-statable fragment is non-empty.) (p. 81)

5. Assume now that bivalent truth can serve as key concept. Bivalent truth-conditions are sometimes undecidable and hence recognition-transcendent. (p. 81)

6. Already in the non-statable fragment there must be sentences with recognition-transcendent truth-conditions. (p. 81)

7. Implicit knowledge of recognition-transcendent truth-conditions cannot be manifested, and so the theory of sense$_D$ is incomplete. (p. 82)

Supplementary notes concerning the argument:

a. Dummett's argument is quite general and does not rest at all on any specific features of the language concerned. When it is applied to a particular area of discourse, or for a particular class of statements, it will lead to a metaphysical anti-realism for the area in question. Many examples of this can be found in Dummett's writings. Thus [1975] and [1977] both develop the argument within the philosophy of mathematics. The intuitionistic criticism of classical reasoning, and the ensuing explanations of the logical constants offered by Heyting, provided the main inspiration for Dummett's work on anti-realism. It should be stressed, however, and as is emphasised by Dummett himself

in [1975], that the semantical argument in favour of a constructivist philosophy of mathematics is very far from Brouwer's own position.

In Dummett [1968–1969] another one of the four critical classes of sentneces is studied, viz. those concerning time, and in WTM2, Section 3, a discussion of counterfactual conditionals can be found, as well as a discussion of certain reductionist versions of anti-realism. They arise when the truth of statement A is reduced to the (simultaneous) truth of a certain possibly infinite class of reduction-sentences M_A. If it so happens that the falsity of the conjunction $\bigwedge\!\!\bigwedge M_A$ dos not entail the truth of the conjunction $\bigwedge\!\!\bigwedge M\neg_A$, then bivalence will fail for the statement A. Examples of such reductionist versions of anti-realism can be found in phenomenalistic reductions of material objects or of sentience in others.

b. It should be noted that Dummett's anti-realism, while verificationist in nature, must not be conflated with logico-empiricist verificationism. With a lot of simplification the matter can be crudely summarised by noting that for the logical empiricists classical logic was sacrosanct and certain sentences have non-verifiable classical truth-conditions. Hence they have no meaning. Dummett reverses this reasoning: obviously meaningful sentences have no good meaning if meaning is construed truth-conditionally. Hence classical meaning-theories are wrong.

c. As one should expect, Dummett's anti-realist argument has not been allowed to remain uncontroverted. JohnMcDowell has challenged the demand that the meaning theories should comprise a theory of sense_D. In his [1977] and [1978] the criticism is mainly by implication as he is there more concerned with the development of the positive side of his own 'modest' version of a meaning theory, whereas in [1981] he explicitly questions the cogency of Dummett's full-blooded theories. McDowell's [1978a] is an answer to Dummett's [1969], and McDowell in his turn has found a critic in Wright [1980a].

Colin McGinn has been another persistent critic of Dummett's anti-realism and he has launched counter-arguments against most aspects of Dummett's position, cf. e.g. his [1980], [1979] and [1982].

Crispin Wright [1982] challenges Dummett by observing that a Strict Finitist can criticise a Dummettian constructivist in much the same way as a Platonist and so the uniquely priviliged position that is claimed for constructivism (as the only viable alternative to classical semantics) is under pressure.

d. Another sort of criticiam is offered by Dag Prawitz [1977, 1978], who, like Wright, is in general sympathy with large parts of Dummett's meaning-theoretical position. Prawitz questions the demand for *full*

manifestation and suggests that the demand for a theory of sense$_D$ be replaced by an *adequacy condition* on meaning theories T:

> if T is to be adequate, it must be possible
> to derive in T the impliction
> if P knows the meaning of A, then P shows behaviour B_A.
> Prawitz [1978, p. 27]

(Here "B_A" is a kind of behaviour counted as a sign of grasping the meaning of A.)

The difference between this adequacy criterion and the constraints that McDowell imposes on his modest theories is not entirely clear to me. Only if the behaviour is to be shown before, and independently of, the theory of force (whose task it is to issue just the interpreting descriptions that tell what behaviour was exhibited by P) could something like a modification of Dummett's argument be launched and even then it does not seem certain that the desired conclusion can be reached.

e. In the presentation of Dummett's argument I have relied solely on WTM2. The anti-realist argument can be found in many places though, e.g. [1973, chapter 14], [1969], [1975] and [1975a] as well as the more recent [1982]. It should be noted that Dummett often cf. e.g., [1969], lays equal or more stress on the acquisition of knowledge rather than its manifestion. Most of the articles mentioned are conveniently reprinted in TE.

Wright [1981], a review of TE, gives a good survey of Dummett's work. Similarly, in his book [1980] Wright offers extensive discussion of anti-realist themes.

The already mentioned McGinn [1980] and Prawitz [1977], while not in entire agreement with Dummett, both give excellent expositions of the basic issues. It is a virtually impossible task to give a complete survey of the controversy around Dummett's anti-realist position. In recent years almost every issue of the *Journal of Philosophy, Mind* and *Analysis*, as well as the *Proceedings of the Aristotelean Society*, contains material that directly, or indirectly, concerns itself with the Dummettian argument.

4 PROOF AS A KEY CONCEPT IN MEANING THEORIES

As was mentioned above the traditioal intuitionistic criticism of clasical mathematical reasoning, cf. e.g., van Dalen (see Volume 5 of the second edition of this *Handbook*) was an important source of inspiration for Dummett's anti-realist argument and it is also to intuitionism that he turns in

his search for an alternative key-concept ot be used in the meaning theories in place of the bivalent, recogntion-transcendent truth-conditions.

The simplest technical treatment of the truth-conditions approach to semantics is undoubtedly provided by the standard truth-tables (which, of course, are incorprated in the Tarski-treatment for, say, full predicate logic) and it is the corresponding constructive 'proof-tables' of Heyting that offer a possibility for Dummett's positive proposal. Heyting's explanations of the logical constants, cf. his [1956, Chapter 7] and [1960], can be set out roughly as follows:

A proof of the proposition	is given by
$A \wedge B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \to B$	a method for obtaining proofs of B from proof of A
$\bot$	nothing
$\forall x \in D A(x)$	a method which for every individual d in D provides a proof of $A(d)$
$\exists x \in D A(x)$	an individual d in D and a proof of $A(d)$.

There are various versions of the above table of explanations, e.g. the one offered by Kreisel [1962], where 'second clauses' have been included in the explanations for implication and universal quantification to the effect that one has to include also a proof that the methods really have the properties required in the explanations above. The matter is dealt with at length in [Sundholm, 1983], where an attempt is made to sort out the various issues involved and where extensive bibliographical information can be found, cf. also Section 7 below on the type theory of Martin-Löf.

In the above explanations the meaning of a proposition is given by its 'proof condition' and, as was emphatically stressed by Kreisel [1962], in some sense, 'we recognise a proof when we see one'. Thus it seems that the anti-realistic worries of Dummett can be alleviated with the use of proof as a key concept in meaning theories. (I will return to this question in the next section.) Independently of the desired immunity from anti-realist strictures, however, there are a number of other points that need to be taken into account here.

First among these is a logical gem invented by Prior [1960]. In the Heyting explanations the meaning of a proposition is given by its proof-condition. Conversely, does every proof-condition give a proposition? A positive answer to this question appears desirable, but the notion 'proof-condition'

needs to be much more elucidated if any headway is to be made here. Prior noted that if by 'proof-conditions' one understands 'rules that regulate deductive practice' then a negative answer is called for. Let us introduce a proposition-forming operator, or connective '*tonk*' by stipulating that its deductive practice is to be regulated by the following Natural Deduction rules (I here alter Prior's rules inessentially):

$$\text{tonk } I \quad \frac{A}{A \text{ tonk } B} \qquad \frac{B}{A \text{ tonk } B}$$

$$\text{tonk } E \quad \frac{A \text{ tonk } B}{A} \qquad \frac{A \text{ tonk } B}{B}.$$

As Prior observes one then readily proves false conclusion from true premises by means of first tonk I and then tonk E. In fact, given these two rule *any two propositions are logically equivalent* via the following derivation:

$$\dfrac{\dfrac{\dfrac{A^1}{A \text{ tonk } B}(\text{tonk } I)}{B} \qquad \dfrac{\dfrac{B^2}{A \text{ tonk } B}(\text{tonk } I)}{A}}{A \leftrightarrow B}\, 1,2(\leftrightarrow)I$$

Thus tonk leads to extreme egalitarianism in the underlying logic: from a logical point of view there is only one proposition. This is plainly absurd and something has gone badly wrong. Hence it is clear (and only what could be expected) that some constraints are needed for how the proof-conditions are to be understood; 'rules regulating deductive practice' is simply too broad. There is quite a literature dealing with tonk and the problems it causes: [Stevenson, 1961; Wagner, 1981; Hart, 1982] and, perhaps most importantly from our point of view, [Belnap, 1962], more about which below. The relevance of the tonk-problem for our present interests, was as far as I know, first noted by Dummett [1973, Chapter 13].

A second point to consider is the so-called *paradox of inference*, cf. Cohen and Nagel [1934, pp. 173–176]. This 'paradox' arises because of the tension between (a) the fact that the truth of the conclusion is already contained in the truth of the premises, and (b) the fact that logical inference is a way to gain 'new' knowledge. Cohen and Nagel formulate it thus:

> *If in an inference the conclusion is not contained in the premise, it cannot be valid; and if the conclusion is not different from the premise, it is useless; but the conclusion cannot be contained in the premises and also possess novelty; hence inferences cannot be both valid and useful* [1934, p. 173]

So there is a tension between the legitimacy (the validity) and the utility of an inference, and one could perhaps reformulate the question posed by the 'paradox' as: How can logic function as a useful epistemological tool? For an inference to be legitimate, the process of recognising the premises as true must already have accomplished what is needed for the recognition of the truth of the conclusion, but if it is to be useful the recognition of the truth of the conclusion does not have to be present when the truth of the premises is ascertained. This is how Dummett poses the question in [1975a].

How does one use reasoning to gain new truths? By starting with known premises and drawing further conclusions. In most cases the use of valid inference has very little to do with how one would normally set about to verify the truth of something. For instance, the claim that I have seven coins in my pocket is best established by means of counting them. It would be possible, however, to deduce this fact from a number of diverse premises and some axioms of arithmetic. (The extra premises would be, say that I began the day with a £50 note, and I have then made such and such purchases for such and such sums, receiving such and such notes and coins in return, etc.) This would be a highly *indirect* way in comparison with the straightforward counting process. The utility of logical reasoning lies in that it provides indirect means of learning the truth of statements. Thus in order to account for this usefulness it seems that there must be a gap between the most direct ways of learning the truth and the indirect ways provided by logic. If we now explain meaning in terms of proof, it seems that we close this gap. The direct means, given directly by the meaning, would coincide, so to speak, with the indirect means of reasoning. The indirect means have then been made a part of the direct means of reasoning. (One should here compare the difference between direct and indirect means of recognising the truth with the solution to the 'paradox' offered by Cohen and Nagel [1934] that is formulated in terms of a concept called 'conventional meaning'.)

The constraints we seek on our proof-explanations thus should take into account, on the one hand, that one must not be too liberal as witnessed by tonk, and, on the other hand, one must not make the identification between proof and meaning so tight that logic becomes useless.

Already Belnap [1962] noted what was wrong with tonk from our point of view. The (new) deductive practice that results from adding *tonk* with its stipulative rules, is not *conservative* over the old one. Using Dummett's [1975] terminology, there is no *harmony* between the grounds for asserting, and the consequences that may be drawn from, a sentence of the form A *tonk* B. The introduction and elimination rules must, so to speak, match, not just in that each connective has introduction and elimination rules but also in that they must not interfere with the previous practice. Hence it seems natural to let one of the (two classes of) rules serve as meaning-giving and let the other one be chosen in such a way that it(s members) can be justified according to the meaning-explanation. Such a method of proceeding would

also take care of the 'paradox' of inference: one of the two types of rules would now serve as the direct, meaning-given (because meaning-giving!) way of learning the truth and the other would serve to provide the indirect means (in conjunction with other justified rules, of course).

The introduction rules are the natural choice for our purpose, since they are *synthesising* rules; they explain how a proof of, say $A\$B$, can be formed in terms of given proofs of A and of B, and thus some sort of compositionality is present (which is required for a key concept). Tentatively then, the meaning of a sentence is given by what counts as a *direct* (or *canonical*) *proof* of it. Other ways of formulating the same explanation would be to say that the meaning is given by the *direct grounds for asserting*, or by what counts as a *direct verification* of, the sentence in question. An (indirect) proof of a sentence would be a method, or program, for obtaining a direct proof.

In order to see that a sentence is true one does not in general have to produce the direct grounds for asserting it and so the desired gap between truth and truth-as-established-by-the-most-direct-means is open. Note that one could still say that the meaning of a sentence is given by its truth-condition, although the latter, of course, has to be understood in a way different from that of bivalent, and recognition-transcendent, truth: if a sentence is true it is possible to give a proof of it and this in turn can be used to produce a *direct proof*. Thus in order to explain what it is for a sentence to be true one has to explain what a direct proof of the sentence would be and, hence, one has to explain the meaning of the sentence n order to explain its truth-condition.

All of this is highly programmatic and it remains to be seen if, and how, the notion of *direct* (canonical) *proof* (verification, ground for asserting) can be made sense of also outside the confined subject-matter of mathematics. In the next section I shall attempt to spell out the Heyting explanations once again, but now in a modified form that closely links up with the discussion in the present section and with the so-called normalisation theorems in Natural Deduction style proof theory.

5 THE MEANING OF THE LOGICAL CONSTANTS AND THE SOUNDNESS OF PREDICATE LOGIC

In the present section, where knowledge of Natural Deduction rules is presupposed, we reconsider Heyting's explanations and show that the introduction and elimination rules are sound for the intended meaning.

Thus we assume that A and B are *meaningful sentences*, or *propositions*, and, hence that we know what proofs (and direct proofs) are for them.

The *conjunction* $A \wedge B$ is a proposition, such that a canonical proof of $A \wedge B$ has the form:

$$\dfrac{\begin{array}{cc} D_1 & D_2 \\ A & B \end{array}}{A \wedge B}$$

where D_1 and D_2 are (not necessarily direct) proofs of Q and B, respectively. On the basis of this meaning-explanation of the proposition $A \wedge B$, the rule $(\wedge I)$ is seen to be valid. We have to show that whenever the two premises A and B are true then so is $A \wedge B$. When A and B are true, they are so on the basis of proof and hence there can be found two proofs D_1 and D_2 respectively of A and B. These proofs can then be used to obtain a canonical proof of $A \wedge B$, which therefore is true.

Consider the elimination rule $(\wedge E)$, say, $\frac{A \wedge B}{B}$, and assume that $A \wedge B$ is true. We have to see that B is true. $A \wedge B$ is true on the basis of a proof D, which by the above meaning-explanation can be used to obtain a canonical proof D_3 of the form specified above. Thus D_2 is a proof of B and thus B is true.

Next we consider the *implication* $A \to B$, which is a proposition that is true if B is true on the assumption that A is true. Alternatively we may say that a canonical proof of $A \to B$ has the form

$$\dfrac{\begin{array}{c} A^1 \\ D \\ B \end{array}}{A \to B^1}$$

where D is a proof of B using the assumption A. Again, the introduction rule $(\to I)$ is sound, since what has to be shown is that *if* B is true on the hypothesis that A is true, *then* $A \to B$ is true. But this is directly granted by the meaning explanation above. For the elimination rule we consider

$$\dfrac{A \to B \quad A}{B}$$

and suppose that we have proofs D_1 and D_2 of respectively $A \to B$ an A. As D_1 is a proof it can be used to obtain a canonical proof D_3 and thus we can find a hypothetical proof D of B from A. But then

$$\begin{array}{l} D_2 \\ A \\ D \\ B \end{array}$$

is a proof of B and thus$B$ is true and $(\rightarrow E)$ is a valid rule.

The *disjunction* $A \vee B$ is a proposition, with canonical proofs of the forms

$$\frac{\begin{array}{c} D_1 \\ A \end{array}}{A \vee B} \quad \text{and} \quad \frac{\begin{array}{c} D_2 \\ B \end{array}}{A \vee B}$$

where D_1 and D_2 are proofs of respectively A and B. The introduction rules are immediately seen to be valid, since they produce canonical proofs of their true premise. For the elimination rule, we assume that $A \vee B$ is true,that C is true on assumption that A is true, and that C is true on assumption that B is true. Thus there are proofs D_1, D_2 and D_3 of, respectively $A \vee B, C$ and C, where the latter two proofs are hypothetical, depending on respectively A and B. The proof D_1 can be used to obtain a canonical proof D_4 of $AveeB$ in one of the two forms above, say the right, and so D_4 contains a subproof D_5, that is a proof of B. Then we readily find a proof of C by combining D_5 with the hypothetical D_3 to get a proof of C, which thus is a true proposition.

The *absurdity* $\perp$ is a proposition which has *no* canonical proof. We have to see that the rule $\frac{\perp}{A}$ is valid. Thus, we have to see that whenever the proposition $\perp$ is true, then also A is true. But $\perp$ is never true, since a proof of $\perp$ could be used to obtain a *canonical* proof of $\perp$ and by the explanation above there are no direct proofs of $\perp$.

The *universal quantification* $(\forall x \in M)A(x)$ is a proposition such that its canonical proofs have the form

$$\frac{\begin{array}{c} x \in M^1 \\ D \\ A(x) \end{array}}{(\forall x \in M)A(x)^1}$$

that is, the proof of D of the premise is a hypothetical, free-variable, proof of $A(x)$ from the assumption that $x \in M$. Again the introduction rule is valid, since if $A(x)$ is true on the hypothesis that $x \in M$, there can be found a hypothetical proof of $A(x)$ from assumption $x \in M$, and thus we immediately obtain a canonical proof of $(\forall x \in M)A(x)$. For the elimination rule $(\forall E)$ consider

$$\frac{(\forall x \in M)A(x) \quad d \in M}{A(d)}$$

and suppose that the premises are true. Thus proofs D_1 and D_2 of, respectively, $(\forall x \in M)A(x)$ and $d \in M$, can be found. As D_1 is a proof it can be

used to obtain a direct proof of its conclusion, and hence we can extract a hypothetical proof of D_3 of $A(x)$ from assumption $x \in M$. Combining D_2 with the free-variable proof D_3 gives a proof

$$\begin{array}{ll} D_2 & (\text{`}D_3(d/x)\text{' indicates the} \\ d \in M & \text{result of substituting} \\ D_3(d/x) & d \text{ for } x \text{ in } D_3.) \\ A(d) & \end{array}$$

of $A(d)$, so the rule $(\forall E)$ is sound.

Finally the *existential quantification* $(\exists x \in M)A(x)$ is a proposition such that its canonical proofs have the $(\exists I)$ form

$$\frac{\begin{array}{cc} D_1 & D_2 \\ A(d) & d \in M \end{array}}{(\exists x \in M)A(x)}$$

Again the introduction rule is immediately seen to be valid as it produces canonical proofs of its conclusion from proofs of the premises. For the elimination rule $(\exists E)$ consider the situation that $(\exists x \in MA(x)$ is true, and that C is true on the assumptions that x is in M and $A(x)$ is true. Thus there can be found a proof D_3 of $(\exists x \in M)A(x)$ and a hypothetical free-variable proof D_4 if C from hypotheses $x \in M$ and $A(x)$. The proof D_3 can be used to obtain a canonical proof of the form above, and combining the proofs D_1 and D_2 with the hypothetical free-variable proof D_4 we obtain a proof of D:

$$\begin{array}{cc} D_2 & D_1 \\ d \in M & A(d) \\ & D_4(d/x) \\ & C \end{array}$$

Thus the rules of the intuitionistic predicate logic are all valid; no corresponding validation is known for, say, the classical law of Bivalence $A \vee \neg A$ where $\neg A$ is defined as $A \to \bot$.

The above treatment has been less precise and complete than would be desirable owing to limitations of space. First, questions of syntax have been left out especially where the quantifier rules are concerned, and secondly a whole complex of problems that arises from the fact that we need to know that $A(x)$ is a proposition for any x in M in order to know that, say, $(\forall x \in M)A(x)$ is a proposition has been ignored. The interested reader is referred to the type theory of Martin-Löf [1984] for detailed consideration and careful treatment of (analogues to) these and other lacunae, e.g. how to treat atomic sentences in our presentation.

The above explanations of why the rules of predicate logic are valid all follow the same pattern. The introduction rules are immediately seen to be valid, since canonical proofs are given introductory form. The elimination rules are then seen to be valid by noting that the introduction and elimination rules have the required harmony. The canonical grounds for asserting a sentence do contain sufficient grounds also for the consequences that may be drawn via the elimination rules for the sentence in question. Thus, in fact, we have here made use of the *reduction steps* first isolated and used by Prawitz [1965, 1971], in his proofs of the normalisation theorems for Natural Deduction-style formalisations.

Prawitz has in a long series of papers [1973, 1974, 1975, 1978 and 1980] been concerned to use this technical insight for meaning-theoretical purposes. His main concern, however, has been to give an explication of the notion of valid argument rather than to give direct meaning explanations in terms of proof. In the presentation here, which is inspired by Martin-Löf's meaning-explanations for his type theory, I have been more concerned with the task of giving constructivistic meaning-explanations while relying on the standard explication of validity as preservation of truth for a justification of the standard rules of inference.

One should, however, stop to consider the extent to which the above explanations constitute a meaning theory in the sense of section 1 above. In particular, in section 4 a promise was given to return to the question of decidability. Is it in fact true that the notion of proof is decidable? On our presentation at least this much is true: if we already have a proof it is decidable if it is in canonical form. As to the general question I would be inclined to think that the notion of proof is *semi-decidable*, in that we recognise a proof when we see one, but when we don't see one that does not necessarily mean that there is no proof there. One can compare the situation with understanding a meaningful sentence: we understand a meaningful sentence when we see (or hear!) one but if we don't understand that does not necessarily mean that there is nothing there to be understood. Failure to understand a meaningful sentence seems parallel to failure to follow, or grasp, a proof. Such a position, then, would not make the 'proof-condition' recognition-transcendent; when it obtains it can be seen to obtain, but when it is not seen to obtain no judgement is given (unless, of course it is seen not to obtain). Apart from the question of decidability, an important difference is that in explanations such as the above there is no mention of implicit knowledge and the like. It seems correct to speak of a theoretical representation of a (constructivistic) deductive practice, but it seems less natural to say that these explanations are known to everyone who draws logical inferences.

We used the notion of canonical proof as a key concept in order to provide the explanations, and in the literature one can find a number of alternatives as to how one ought to specify these, cf. the papers by Prawitz listed

above. In particular, one might wish to insist that all parts of a canonical proof should also be canonical (as is the case with the so-called normal derivations obtained by Prawitz in his normalization theorem [1971]). The choice I opted for here was motivated by, first, the success of the meaning-explanations of Martin-Löf in his type theory and, secondly, the fact that in Hallnäs [1983] a successful normalisation of strong set-theoretic systems is carried out using an analogous notion of normal derivation (Tennant's [1982] and his book [1978] are also interesting to the set-theoretically curious; in the former a treatment of the paradoxes is offered along Natural Deduction lines, and the latter contains a neat formulation of the rules of set theory.)

Finally, we should note that the explanations offered here have turned the formal system into an *interpreted formal system* (modulo not inconsiderable imprecision in the formulation of syntax and explanations). This is the main reason for the avoidance of Greek letters in the present Chapter.

6 QUESTIONS OF COMPLETENESS

In section 5 the meaning of the logical constants was explained and the standard deductive practice justified. In the case of classical, bivalent logic we know that the connectives $\wedge, \vee$ and $\neg$ are complete in that any truth-function can be generated from them. Does the corresponding property hold here? Clearly the answer is dependent on how the canonical proofs may be formed. It was shown by Prawitz [1978] and, independently of him, by Zucker and Tragresser [1978] that if we restrict ourselves to purely schematic means for obtaining canonical proofs (and for logical constants this does not seem unreasonable), then an affirmative answer is possible to the above question. As a typical example consider e.g. this Sheffer-stroke (which of course makes sense constructively as well). This is given the introduction rule $(|I)$

$$\dfrac{\begin{array}{c}A^1 \ldots B^2 \\ \vdots \\ \bot\end{array}}{A|B^{1,2}}$$

A definition using $\wedge, \rightarrow$ and $\bot$ is found by putting $A|B =_{\text{def}} A \wedge B \rightarrow \bot$. If there are more premise-derivations in the introduction rule (= the rule for how canonical proofs may be obtained) for each of these one will get an implication of the above sort and they are all joined together by conjunctions. (Here it is presupposed that the rules have only finitely many premises. This does not seem unreasonable.) Finally, if there are more introduction

rules than one, the conjunctions are put together into a disjunction. (Here it is presupposed that there are only finitely many introduction rules. Again this does not seem unreasonable)(.

Only one case remains, namely that there are no introduction rules. Then there are no canonical proofs to be found and we have got the absurdity. Thus the fragment based on $\rightarrow, \wedge, \vee$ and $\bot$ is complete. For further details refer to the two original papers above as well as Schroeder-Heister [1982]. It should be noted that by Hendry [1981] we know that $A \wedge B$ is equivalent also intuitionistically to $(A \leftrightarrow B) \leftrightarrow (A \vee B)$ and that $A \rightarrow B$ is equivalent to $B \leftrightarrow A \vee B$. Thus also $\leftrightarrow, \vee$ and $\bot$ are complete.

The standard elimination-rules $(\wedge E)$ can be replaced by the following rule:

$$\dfrac{\begin{array}{cc} & A^1 \ldots B^2 \\ & \vdots \\ A \wedge B & C \end{array}}{C}\,{}^{1,2}$$

which rule seems quite well-motivated by the analogy with the introduction rule $\frac{A \quad B}{A \wedge B}$: everything which can be derived from the two premises A and B used as assumptions can also be derived from $A \wedge B$ alone. The $(\vee E)$ rule has exactly this general pattern and the intuitionistic absurdity rule is a degenerate case without minor premise C:

$$\dfrac{\bot}{C}$$

Only implication does not obey the above pattern. Here the premise of the introduction rule is not just a sentence, but a hypothetical judgement that B is true whenever A is true. Thus, we have a sort of rule as premise: from A go to B, in symbols $A \Rightarrow B$. If we may use such rules as *dischargable* assumptions, one an keep the standard pattern also for implication, viz.

$$\dfrac{\begin{array}{cc} & A \Rightarrow B^1 \\ & \vdots \\ A \rightarrow B & C \end{array}}{C}\,{}^{1}$$

whereas if we try to do the same using implication for the arrow $\Rightarrow$, we end up with the triviality

$$\dfrac{A \to B \qquad \begin{array}{c} A \to B^{1} \\ \vdots \\ C \end{array}}{C}\,1$$

which does not allow us to derive even *modus ponens*.

Using the rule with the higher level assumption $A \Rightarrow B$ one can derive $(\to E)$ as follows:

$$\dfrac{A \to B \qquad \begin{array}{c} A \\ \hline B \end{array}(A \Rightarrow B)^{1}}{B}\,1$$

Given the use of the rule $A \Rightarrow B$ as an assumption, from premise Q we can proceed to conclusion B, and the use of the major premise $A \to B$ allows to *discharge the use of the rule* $A \Rightarrow B$.

This type of higher-level assumptions was introduced by Schroeder-Heister [1981] and it is a most interesting innovation in Natural Deduction-formulations of logic, cf. also his [1982] and [1983]. The elimination rule that the Prawitz method gives to the Sheffer-stroke would be

$$\dfrac{A|B \qquad \begin{array}{c} A \wedge B \to \bot^{1} \\ \vdots \\ C \end{array}}{C}\,1$$

which follows the above pattern, but uses implication and conjunction. With the Schroeder-Heister conventions the rule can be given as

$$\dfrac{A|B \qquad \begin{array}{c} (A, B \to \bot)^{1} \\ \vdots \\ C \end{array}}{C}\,1$$

In words, if C is true under the assumptions that we may go from the premise A and B to conclusion $\bot$, then C is a consequence of $A|B$ alone.

In Schroeder-Heister [1984] an extension of the above results is given and completeness is established also for the predicate calculus language.

The other question of completeness is also considered by Schroeder-Heister [1983]: is every valid inference derivable from the introduction and

elimination rules? This question gets a positive answer, but the concept of validity is extremely restrictive, i.e. the rule $\frac{(A \wedge B) \wedge C}{A}$ is not a valid rule, cf. [1983, p. 374], which (given the concept of validity used in the present paper) it obviously must be. Thus I would consider the problem, first posed by Prawitz [1973], to establish the completeness of the predicate logic, for the present sort of meaning explanations, still to be open.

7 THE TYPE THEORY OF MARTIN-LÖF

Frege [1893], in the course of carrying out his logicist programme, designed a full-scale, completely formal language that was intended to suffice for mathematical practice. By today's standards, an almost unique feature of his attempt to secure a foundation of mathematics is that he uses an *interpreted* formal language for which he provides careful meaning explanations. The language proposed was, as we now know, not wholly successful, owing to the intervention of Russell's paradox. (The effects of the paradox on Frege's explanations of meaning are explored in Aczel [1980] and, from a different perspective, in Thiel [1975] and Martin [1982].) As the formal logic of Frege (and Whitehead–Russell) was transformed gradually into mathematical logic, notably by Tarski and Gödel, interest in the task of giving meaning explanations for interpreted formal languages faded out and after World War II the current distinction between syntax and (Tarskian, model-theoretic) semantics has become firmly entrenched.

The type theory of Martin-Löf [1975, 1982, 1984] represents a remarkable break with this tradition in that it returns to the original Fregean paradigm: interpreted formal language with careful explanations of meaning. Owing to limitations of space I shall not be able to give a detailed, precise description of the system here, (a task for which Martin-Löf [1984] uses close to a hundred ages), but will confine myself to trying to convey the basic flavour of the system.

A possible route to Martin-Löf's theory is through further examination of Heyting's explanations of the meaning of the logical constants. Our tentative semantics in section 5 above made tacit use of a refinement of the explanations: the proof-tables do not give just proofs but *canonical*, or *direct* proofs. A further refinement can be culled from Heyting's own writings. (In Sundholm [1983] a fairly detailed examination of Heyting's writings on this topic is offered.) According to Heyting, in order to prove a theorem one has to carry out certain constructions, 'die gewissen Bedingungen genügen', namely that it produces a mathematical object with certain specified properties, cf. e.g. his remarks on the proposition

> "Euler's constant is rational"

in [1931, p. 113]. In Martin-Löf's system, the proof-tables are extended to

contain also the information about the objects that need to be constructed in order to establish the truth of the propositions in question. Thus, taking both refinements into account, the meaning of a proposition is explained by telling what a canonical object for the proposition would be. (A canonical object is not needed in order to assert the proposition; an object (method program) that can be evaluated to canonical form is enough. For more details here, see Martin-Löf [1984].) In fact, according to Martin-Löf, one also has to tell when two such objects are equal. On the other hand, when one defines a set constructively, one has to specify what the canonical elements are and what it is for two elements of the set to be equal elements. Thus, the explanations of what propositions are and of what sets are, are completely analogous and Martin-Löf's system does not differentiate between the two notions.

In ordinary formal theories, that are formulated in the predicate calculus, the derivable objects are *propositions* (or, rather, they are well-formed formulae, i.e. the formalistic counterparts of propositions). This leads to certain difficulties for the standard formulation where logical inference is a relation between proposition. As was already observed by Frege, the correct formulation of *modus ponens* is

$$\frac{A \to B \text{ is true} \qquad A \text{ is true}}{B \text{ is true}};$$

It is simply not correct to say that the proposition B follows from the propositions $A \to B$ and Q. What is correct is that the *truth* of the proposition B follows from the *truth* of $A \to B$ and the *truth* of A. Thus *the premises and conclusions of logical inferences are not propositions but judgements as to the truth of the propositions.* Furthermore, as Martin-Löf notes, that in order to keep the rules formal, one should also include the information that A and B are propositions in the premises of the rules, e.g.

$$\frac{A \text{ is a prop.} \quad B \text{ is a prop.} \quad A \text{ is true}}{A \vee B \text{ is true}}$$

is how $\vee$-introduction should be set out. Therefore, as the premises of inferences are judgements, and remembering the identification of propositions and sets, on finds two main sorts of judgements in the theory, namely

(a) A set ('A is a set')

and

(b) $a \in A$ ('a is an element of the set A')

(In fact, there are two further forms of judgement, namely 'A s the same set as B' and 'a and b are equal elements of the set A'.)

In accordance with the above discussion, (a) also does duty for 'A is a proposition' and (b) can also be read as 'the (proof-)object a is of the right sort for the proposition A, meets the condition specified by the proposition A'. This reading of (b) is, constructively, a longhand for the judgement '(the proposition) A (is) *true*', which is used whenever it is convenient to suppress the extra information contained is the proof-object. A third reading, deriving from Heyting and Kolmogorov, is possible, where (a) is taken in the sense 'A is a *task* (or *problem*)' and (b) in the sense 'a is a method for carrying out the task A (solving the problem A)'. When the task-aspect is emphasises, another reading would be 'a is a *program* that meets the *specification* A' and the type-theoretical language of Martin-Löf [1982] has, owing to this possibility, had considerable influence as a programming language.

Some feeling for the interaction between propositions and proof-objects may be obtained through consideration of the simple example of *conjunction*. The *proposition* $A \wedge B$ (or *set* $A \times B$) is explained, on the assumption that A and B are propositions, by laying down that a canonical element of $A \times B$ is a pair (a, b) where $a \in A$ and $b \in B$. Thus the $\times$-introduction rule is correct:

$$\frac{a \in A \quad b \in B}{(a, b) \in A \times B}.$$

Using the shorthand reading, when the proof-objects are left out, we also see that the rule of $\wedge$-introduction is correct:

$$\frac{A \text{ true} \quad B \text{ true}}{A \wedge B \text{ true}}.$$

For the $\wedge$-eliminations, we need the use of the *projection-functions* p and q that are associated with the pairing-function. Consider the rule

$$\frac{A \wedge B \text{ true}}{A \text{ true}}.$$

Restoring proof-objects, we see that from an element $c \in A \wedge B$, one has to find an element of A. But c is an element of $A \wedge B$, and so c is equal to (is a method for finding, can be evaluated, or executed, to) a *canonical* element $(a, b) \in A \wedge B$. Applying the projection p, we see that $p(c) = p((a, b)) = a \in A$, so the proper formulation will be

$$\frac{c \in A \wedge B}{p(c) \in A}.$$

It should be mentioned, however, that the conjunction is not a primitive set-formation operation in the language of Martin-Löf. On the contrary, a suitable candidate can be defined from other sets and the appropriate rules derived.

A slightly more complex example is provided by the *universal quantification* $(\forall x \in A)B[x]$ and *implication* $A \to B$, both of which are treated as variants of the Cartesian product $(\Pi x \in A)B[x]$ of a family of sets. This product may be formed only on the assumption that we have a family of sets over A, that is, provided that $B[x]$ isa set, whenever $x \in A$. Thus the formation rule will take the form

$$\frac{\begin{array}{c} x \in A^1 \\ \vdots \\ A \text{ set} \quad B[x] \text{ set} \end{array}}{(\Pi x \in A)B[x] \text{ set}}\,.$$

(This serves to illustrate the important circumstance that the basic judgements may depend on assumptions. Better still, we should say that the right premise is a *hypothetical* judgement $B[x]$ set (provided that $x \in A$).) In order to understand the Π-formation rule one needs to know what a canonical element of $(\Pi x \in A)B[x]$ would be; this is told by the Π-introduction rule

$$\frac{\begin{array}{c} x \in A^1 \\ \vdots \\ b[x] \in B[x] \end{array}}{\lambda x.b[x] \in (\Pi x \in A)B[x]}$$

that is, the canonical elements are *functions* $\lambda x b[x]$, such that $b[x] \in B[x]$ provided that $x \in A$. Just as in the case of conjunction, where the elimination rule was taken care of by matching the pairing function with a projection, one will obtain the elimination rule through a similar match between λ-abstraction and function-*application*, *ap*. Thus the rule take the form

$$\frac{f \in (\Pi x \in A)B[x] \quad a \in A}{ap(F, a) \in B[a/x]}\,.$$

(In order to understand this rule one makes use of an important connection between abstraction and application, namely the law

$$ap(\lambda x.b[x], a) = b[a/x].$$

For the details of the explanation, refer to Martin-Löf [1982] or [1984].)

If the set (proposition) $B[x]$ does not depend on x the product is written as the set of functions B^A (as the proposition $A \to B$). The rules are obvious, with the exception of $\to$-formation:

$$\frac{\begin{array}{c}[A \text{ true}^1] \\ \vdots \\ A \text{ prop} \quad B \text{ prop}\end{array}}{A \to B \text{ prop} \quad {}^1}.$$

Here the formation rule is stronger than the usual rule (where A and B both have to be propositions) because the right premise is weaker in that B has to be a proposition only when A is true. This concept of implication has been used by Stenlund in an elegant theory of definite descriptions, cf. his [1973] and [1975].

The other quantification is taken care of by means of the *disjoint union* of a family of sets. The Σ-formation rule takes the form

$$\frac{\begin{array}{c}[x \in A^1] \\ \vdots \\ A \text{ set} \quad B[x] \text{ set}\end{array}}{(\Sigma x \in A)B[x] \text{ set} \quad {}^1}.$$

The canonical elements are given by the Σ-introduction rule

$$\frac{a \in A \quad b \in B[a/x]}{(a, b) \in (\Sigma x \in A)B[x]}.$$

On the propositional reading, where the disjoint union is written as the quantifier $(\exists x \in A)B[x]$, we see that in order to establish an existence claim one has to (i) exhibit a suitable witness $a \in A$ and (ii) supply a suitable proof-object b that the witness $a \in A$ does, in fact, satisfy the condition imposed by $B[x]$. The inclusion of the proof-object b allows yet a third use for the disjoint union, namely that of restricted comprehension-terms. What would, on a constructive reading, be meant by 'an element of the set of x's in A such that $B[x]$'? At least one would have to include a witness $a \in A$ and information ($= a$ proof-object) establishing that a satisfies the condition $B[x]$. Thus the canonical elements of the restricted comprehension-term $\{x \in A : B[x]\}$ coincide with the canonical elements of the disjoint sum. This representation of 'such that' provides the key to the actual development of, say, the theory of real numbers given the set N

of natural numbers. A real number will be an element of N^N such that it obeys a Cauchy-condition.

At this point I will refrain from further development of the language and instead I shall apply the type-theoretic abstractions that have been introduced so far to the notorious 'donkey-sentence'

(*) Every man who owns a donkey beats it.

The problem here is, of course, that formulations within ordinary predicate logic do not seem to provide any way to capture the back-reference of the pronoun 'it'. A simple-minded formalisation yields

(**) $\forall x(\mathrm{Man}(x) \wedge \exists y(\mathrm{Donkey}(y) \wedge \mathrm{Own}(x,y)) \rightarrow \mathrm{Beats}(x,?))$.

There seems to be no way of filling the place indicated by '?', as the donkey has been quantified away by 'y'.

Using the disjoint-union manner of representation for restricted comprehension-terms one finds that 'a man who owns a donkey' is an element of the set

$$\{x \in \mathrm{MAN} : (\exists y \in \mathrm{DONKEY})\mathrm{OWN}[x,y]\}.$$

Such an element, when in canonical form, is a pair (m,b), where $m \in$ MAN and b is a proof-object for $(\exists y \in \mathrm{DONKEY})\mathrm{OWN}[m/x,y]$. Thus b, in its turn, when brought to canonical form, will be a pair (d,c), where d is a DONKEY and c a proof-object for $\mathrm{OWN}[m/x,d/y]$. Thus for an element z of the comprehension-term 'MAN who OWNs a DONKEY' the left projection $p(z)$ will be a man and the right projection $q(z)$ will be a pair whose left projection $p(q(z))$ will be the witnessing donkey. Putting it all together we get the formulation

(***) $(\forall z \in \{x \in \mathrm{MAN} : (\exists y \in \mathrm{DONKEY})\mathrm{OWN}[x,y]\})\mathrm{BEAT}[p(z), p(q(z))]$.

In this manner, then, the type-theoretic abstractions suffice to solve the problem of the pronominal back-reference in (*). It should be noted here that there is nothing *ad hoc* about the treatment, since all the notions used have been introduced for mathematical reasons in complete independence of the problem posed by (*). One the other hand one should stress that it is not at all clear that one can export the 'canonical proof-objects' conception of meaning outside the confined area of constructive mathematics. In particular the treatment of atomic sentences such a 'OWN$[x,y]$' is left intolerably vague in the sketch above and it is an open problem how to remove that vagueness.

Martin-Löf's type theory has attracted a measure of metamathematical attention. Peter Aczel [1977, 1987, 1980, 1982], in particular, has been a tireless explorer of the possibilities offered by the type theory. Other papers of interest are Diller [1980], Diller and Troelstra [1984] and Beeson [1982].

NOTE ADDED IN PROOF (OCTOBER 1985)

Per Martin-Löf's 'On the meanings of the logical constants and the justifications of the logical laws' in *Atti degli incomtri di logica matematica vol. 2*, Scuolo di Specializzazione in Logica Matematica, Dipartimento di Mattematica, Università di siena, 1985, pp. 203–281, was not available during the writing of the present chapter. In these lectures, Martin-Löf deals with the topics covered in sections 4–6 above in great detail and carries the philosophical analysis considerably further.

University of Nijmegen, The Netherlands.

EDITOR'S NOTE 2001

[For the most recent coverage of Martin-Löf's type theory, see the chapter by B. Nordström, K. Peterson and J. M. Smith in S. Abramsky, D. Gabbay and T. S. E. Maibaum, eds., *Handbook of Logic in Computer Science*, volume 5, pp. 1–37, Oxford University Press, 2000.]

BIBLIOGRAPHY

[Aczel, 1977] P. Aczel. The strength of Martin-Löf's type theory with one universe. In *Proceedings of the Symposium on Mathematical Logic (Oulo 1974)*, S. Mietinen and J. Väänänen, eds. pp. 1–32, 1977. Report No 2, Department of Philosophy, Univeristy of Helsinki.

[Aczel, 1978] P. Aczel. the type theoretic interpretation of constructive set theory. In *Logic Colloquium 1977*, A. Macintyre *et al.*, eds. pp. 55–66, 1978.

[Aczel, 1980] P. Aczel. Frege structures and the notions of proposition, truth and set. In *The Kleene Symposium*, J. Barwise *et al.*, eds, pp. 31–59. North-Holland, Amsterdam, 1980.

[Aczel, 1982] P. Aczel. The type theoretic interpretation of constructive set theory: choice principles. In *The L. E. J. Brouwer Centenary Symposium*, A. S. Troelstra and D. van Dealen, eds. pp. 1–40. North-Holland, Amsterdam, 1982.

[Baldwin, 1979] T. Baldwin. Interpretations of quantifiers. *Mind*, **88**, 215–240, 1979.

[Beeson, 1982] M. Beeson. Recursive models for constructive set theories. *Annals Math.Logic*, **23**, 127–178, 1982.

[Belnap, 1962] N. D. Belnap. Tonk, plonk and plink. *Analysis*, **22**, 130–134, 1962.

[van Benthem and van Eijck, 1982] J. F. A. K. van Benthem and J. van Eijck. The dynamics of interpretation. *J. Semantics*, **1**, 3–20, 1982.

[Cohen and Nagel, 1934] M. R. Cohen and E. Nagel. *An Introduction to Logic and Scientific Method*, Routledge and Kegan Paul, London, 1934.

[Davidson, 1967] D. Davdison. Truth and meaning. *Synthese*, **17**, 304–323, 1967.

[Davidson, 1984] D. Davidson. *Inquiries into Truth and Interpretation*, Oxford University Press, 1984.

[Davies, 1981] M. K. Davies. *Meaning, Quantification, Necessity*. Routledge and Kegan Paul, London, 1981.

[Diller, 1980] J. Diller. Modified realisation and the formulae-as-types notion. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J. P. Seldin and R. Hindley, eds. pp. 491–502. Academic Press, London, 1980.

[Diller and Troelstra, 1984] J. Diller and A. S. Troelstra. Realisability and intuitionistic logic. *Synthese*, **60**, 153–282, 1984.
[Dummett, 1968–1969] M. Dummett. The reality of the past. *Proc. Aristot. Soc.*, **69**, 239–258, 1968–69.
[Dummett, 1973] M. Dummett. *Frege*. Duckworth, London, 1973.
[Dummett, 1975] M. Dummett. The philosophical basis of intuitionistic logic. In *Logic Colloquium '73*, H. E. Rose and J. Shepherdson, eds. pp. 5–40. North-Holland, Amsterdam, 1975.
[Dummett, 1975a] M. Dummett. The justification of deduction. *Proc. British Academy*, **LIX**, 201–321, 1975.
[Dummett, 1976] M. Dummett. What is a theory of meaning? (II). [WTM2] In [Evans and McDowell, 1976], pp. 67–137, 1976.
[Dummett, 2000] M. Dummett. *Elements of Intuitionism*, 2nd edition, Oxford University Press, 2000. (First edition 1977.)
[Dummett, 1978] M. Dummett. *Truth and Other enigmas*, [TE], Duckworth, London, 1978.
[Dummett, 1981] M. Dummett. Frege and Wittgenstein. In *Perspectives on the Philosophy of Wittgenstein*, I. Block, ed. pp. 31–42. Blackwell, Oxford, 1981.
[Dummett, 1982] M. Dummett. Realism. *Synthese*, **52**, 55–112, 1982.
[Evans, 1977] G. Evans. Pronouns, quantification and relative clauses (I). *Canadian J. Phil.*, reprinted in Platts [1980, pp. 255–317].
[Evans and McDowell, 1976] G. Evans and J. McDowell, eds. *Truth and Meaning*. Oxford University Press, 1976.
[Foster, 1976] J. A. Foster. Meaning and truth theory. In [Evans and McDowell, 1976, pp. 1–32].
[Frege, 1893] G. Frege. *Grundgesetze der Arithmetik*, Jena, 1893.
[Gentzen, 1934–1935] G. Gentzen. Untersuchungen über das logische Schliessen. *Math —eitschrift*, **39**, 176–210, 405–431, 1934–1935.
[Hacking, 1979] I. Hacking. What is logic? *J. Philosophy*, **76**, 285–319, 1979. Reproduced in *What is a Logical System?*, D. Gabbay, ed. Oxford University Press, 1994.
[Hallnäs, 1983] L. Hallnäs. *On Normalization of Proofs in Set Theory*, Dissertation, Univeristy of Stockholm, Preprint No. 1, Department of Philosophy, 1983.
[Hart, 1982] W. D. Hart. Prior and Belnap. *Theoria*, **XLVII**, 127–138, 1982.
[Hendry, 1981] H. E. Hendry. Does IPC have binary indigenous Sheffer function? *Notre Dame J. Formal Logic*, **22**, 183–186, 1981.
[Heyting, 1931] A. Heyting. Die intuitionistische Grundlegung der Mathematik. *Erkenntnis*, **2**, 106–115, 1931.
[Heyting, 1956] A. Heyting. *Intuitionism*, North-Holland, Amsterdam, 1956.
[Heyting, 1960] A. Heyting. Remarques sur le constructivisme. *Logique et Analyse*, **3**, 177–182, 1960.
[Kleene, 1967] S. C. Kleene. *Mathematical Logic*, John Wiley & Sons, New York, 1967.
[Kreisel, 1962] G. Kreisel. Foundations of intuitionistic logic. In *Logic, Methodology and Philosophy of Science*, E. Nagel *et al.*, eds. pp. 198–210. Stanford University Press, 1962.
[Loar, 1976] B. Loar. Two theories of meaning. In [Evans and McDowell, 1976, pp. 138–161].
[McDowell, 1976] J. McDowell. Truth conditions, bivalence and verificationism. In [Evans and McDowell, 1976, pp. 42–66].
[McDowell, 1977] J. McDowell. On the sense and reference of a proper name. *Mind*, **86**, 159–185, 1977. Also in [Platts, 1980].
[McDowell, 1978] J. McDowell. Physicalism and primitive denotation: Field on Tarski. *Erkenntnis*, **13**, 131–152, 1978. Also in [Platts, 1980].
[McDowell, 1978a] J. McDowell. On 'The reality of the past'. In *Action and Interpretation*, C. Hookway and P. Pettit, eds. p. 127–144. Cambridge University Press, 1978.
[McDowell, 1981] J. McDowell. Anti-realism and the epistemology of understanding. In *Meaning and Understanding*, H. Parrett and J. Bouveresse, eds. pp. 225–248. de Gruyter, Berlin, 1981.

[McGinn, 1979] C.McGinn. An *a priori* argument for realism. *J. Philosophy*, **74**, 113–133, 1979.

[McGinn, 1980] C. McGinn. Truth and use. In [Platts, 1980, pp. 19–40].

[McGinn, 1982] C. McGinn. Realist semantics and content ascription. *Synthese*, **52**, 113–134, 1982.

[Martin, 1982] E. Martin, Jr. Referentiality in Frege's *Grundgesetze*. *History and Philosophy of Logic*, **3**, 151–164, 1982.

[Martin-Löf, 1975] P. Martin-Löf. An intuitionistic theory of types. In *Logic Colloquium '73*, H. E. Rose and J. Shepherdson, eds. pp. 73–118. North-Holland, Amsterdam, 1975.

[Martin-Löf, 1982] P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, L. J. Cohen *et al.*, eds. pp. 153–175, North-Holland, Amsterdam, 1982.

[Martin-Löf, 1984] P. Martin-Löf. *Intuitionistic Type Theory*. Notes by Giovanni Sambin of a series of lectures given in Padova, June 1980, Bibliopolis, Naples, 1984.

[Peacocke, 1981] C. A. B. Peacocke. The theory of meaning in analytical philosophy. In *Contemporary Philosophy, vol. 1*, G. Flöistad, ed. pp. 35–36. M. Nijhoff, The Hague, 1981.

[Platts, 1979] M. de B. Platts. *Ways of Meaning*. Routledge and Kegan Paul, London, 1979.

[Platts, 1980] M. de B. Platts. *Reference, Truth and Reality*. Routledge and Kegan Paul, London, 1980.

[Prawitz, 1965] D. Prawitz. *Natural Deduction*. Dissertation, University of Stockholm, 1965.

[Prawitz, 1971] D. Prawitz. Ideas and results in proof theory. In *Proceedings of the Second Scandinavian Logic Symposium*, J.-E. Fenstad, ed. pp. 235–308. North-Holland, Amsterdam, 1971.

[Prawitz, 1973] D. Prawitz. Towards a foundation of general proof theory. In *Logic, Methdology and Philosophy of Science IV*, P Suppes *et al.*, eds. pp. 225–250. North-Holland¡ Amsterdam, 1973.

[Prawitz, 1975] D. Prawitz. Comments on Gentzen-type procedures and the classical notion of truth. In *Proof Theory Symposium* J. Diller and G. H. Müller, eds. pp. 290–319. Lecture Notes in Mathematics 500, Springer, Berlin, 1975.

[Prawitz, 1977] D. Prawitz. Meaning and Proofs. *Theoria*, **XLIII**, 2–40, 1977.

[Prawitz, 1978] D. Prawitz. Proofs and the meaning and completeness of the logical constants. In *Essays on Mathematical and Philosophical Logic*, J. Hintikka *et al.*, eds. pp. 25–40. D. Reidel, Dordrecht, 1978.

[Prawitz, 1980] D. Prawitz. Intuitionistic logic: a philosophical challenge. In *Logic and Philosophy*, G. H. von Wright, ed. pp. 1–10. M. Nijhoff, The Hague, 1980.

[Prior, 1960] A. n. Prior. The runabout inference ticket. *Analysis*, **21**, 38–39, 1960.

[Schroeder-Heister, 1981] P. Schroeder-Heister. *Untersuchungen zur regellogischen Deutung von Aussagenverknüpfungen*. Dissertation, University of Bonn, 1981.

[Schroeder-Heister, 1982] P. Schroeder-Heister. Logische Konstanten und Regeln. *Conceputs*, **16**, 45–60, 1982.

[Schroeder-Heister, 1983] P. Schroeder-Heister. The completeness of intuitionistic logic with respect to a validity concept based on an inversion principle. *J. Philosophical Logic*, **12**, 359–377, 1983.

[Schroeder-Heister, 1984] P. Schroeder-Heister. generalised rules for quantifiers and the completeness of the intuitionistic operators $\&, \vee, \supset, \bot, \forall, \exists$. In *Computation and Proof Theory*, M. Richter, *et al.*, eds. pp. 399–426. Lecture notes in Mathematics, 1104, Springer, Berlin, 1984.

[Smullyan, 1968] R. Smullyan. *First Order Logic*, Springer-Verlag, Berlin, 1968.

[Stenlund, 1973] S. Stenlund. *The Logic of Description and Existence*. Department of Philosophy, University of Uppsala, 1973.

[Stenlund, 1975] S. Stenlund. Descriptions in intuitionistic logic. In *Proceedings of the Third Scandinavian Logic Symposium*, S. Kanger, ed. pp. 197–212. North-Holland, Amsterdam, 1975.

[Stevenson, 1961] J. T. Stevenson. Roundabout the runabout inference-ticket. *Analysis*,m **21**, 124–128, 1961.
[Sundholm, 1981] G. Sundholm. Hacking's logic. *J. Philosophy*, **78**, 160–168, 1981.
[Sundholm, 1983] G. Sundholm. Constructions, proofs and the meaning of the logical constants. *J. Philosophical Logic*, **12**, 151–172, 1983.
[Tarski, 1956] A. Tarski. *Logic, Semantics, Metamathematics*, Oxford University Press, 1956.
[Tennant, 1978] N. Tennant. *Natural Logic*, Edinburgh University Press, 1978.
[Tennant, 1982] N. Tennant. proof and paradox. *Dialectica*, **36**, 265–296, 1982.
[Thiel, 1975] C. Thiel. Zur Inkonsistenz der Fregeschen Mengenlehre. In *Frege und die moderne Grundlagenfirschung*, C. Thiel, ed. pp. 134–159. A. Hain, Meisenheim, 1975.
[Wagner, 1981] S. Wagner. Tonk. *Notre Dame J. Formal Logic*, **22**, 289–300, 1981.
[Wright, 1976] C. Wright. Truth conditions and criteria. *Proc. Arist. Soc.*, supp **50**, 217–245, 1976.
[Wright, 1980] C. Wright. *Wittgenstein on the foundations of Mathematics*. Duckworth, London, 1980.
[Wright, 1980a] C. Wright. Realism, truth-value links, other minds and the past. *Ratio*, **22**, 112–132, 1980.
[Wright, 1981] C. Wright. Dummett and revisionism. *Philosophical Quarterly*, **31**, 47–67, 1981.
[Wright, 1982] C. Wright. Strict finitism. *Synthese*, **51**, 203–282, 1982.
[Zucker and Tragresser, 1978] J. Zucker and R. S. Tragresser. The adequacy problem for inferential logic. *J. Philosophical Logic*, **7**, 501–516, 1978.

DOV GABBAY AND NICOLA OLIVETTI

GOAL-ORIENTED DEDUCTIONS

1 INTRODUCTION

The topic of this chapter is to present a general methodology for automated deduction inspired by the logic programming paradigm. The methodology can and has been applied to both classical and non-classical logics. It comes without saying that the landscape of non-classical logics applications in computer science and artificial intelligence is now wide and varied, and this Handbook itself is a witness this fact. We will survey the application of goal-directed methods to classical, intuitionistic, modal, and substructural logics. For background information about these logical systems we refer to other chapters of this Handbook and to [Fitting, 1983; Anderson and Belnap, 1975; Anderson *et al.*, 1992; Gabbay, 1981; Troelstra, 1969; Dummett, 2001; Restall, 1999]. Our treatment will be confined to the propositional level.[1]

In the area of automated deduction and proof-theory there are several objectives which can be pursued. Methods suitable for one task are not necessarily the best ones for another. Consider propositional classical logic and the following tasks:

1. check if a randomly generated set of clauses is unsatisfiable;
2. given a formula A check whether A is valid;
3. given a set Γ containing say 5,000 formulas and a formula A check whether $\Gamma \vdash A$;
4. (saturation) given a set of formulas Γ generate all atomic propositions which are entailed by Γ;
5. (abduction) given a formula A and a set of formulas Γ such that $\Gamma \nvdash A$, find a minimal set of atomic propositions S such that $\Gamma \cup S \vdash A$ and satisfies some other constraints.

It is not difficult to see that all these problems can be reduced one to the other. However, it is quite likely that we need different methods to address each one of them efficiently. Consider task 3: Γ may represent a 'deductive database' and A a query. It might be that the formulas of Γ have a simple/uniform structure and only a small subset of the formula of Γ are relevant for getting a proof of A (if any): thus a general general

[1] The reader of the chapter of Basin and Matthews on logical frameworks can regard our chapter as a goal directed logical framework done in the *object level*.

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, 199–285.

SAT-algorithm applied $\Gamma \cup \{\neg A\}$ might not be the most natural method, we would prefer a method capable of concentrating on the relevant data and ignoring the rest of Γ. Similar considerations applies to the other tasks. For instance in the case of abduction, we would likely calculate the set of abductive assumptions S from failed attempts to prove the data A, rather than guess an S arbitrarely and then check if it works. Moreover, in some applications we are not only interested to know whether a formula is valid or not, but also to find (and inspect) a proof of it in an understandable format. The goal-directed approach to deduction is useful to support deduction from large databases, abduction procedures, and proof search.

In a few words, the goal-directed paradigm is the same as the one underlying logic programming. The deduction process can be described as follows: we have a structured collection of formulas (called a database) Δ and a goal formula A, and we want to know whether A follows from Δ or not, in a specific logic. Let us denote by

$$\Delta \vdash^{?} A$$

the query 'does A follows from Δ?' (in a given logic). The deduction is *goal-directed* in the sense that the next step in a proof is determined by the form of the current goal: the goal is stepwise decomposed, according to its logical structure, until we reach its atomic constituents. An atomic goal q is then matched with the 'head' of a formula $G' \to q$ (if any, otherwise we fail) in the database, and its 'body' G' is asked in turn. This step can be understood as a resolution step, or as a generalized Modus Tollens. We will see that we can extend this backward reasoning, goal-directed paradigm to most non-classical logics. We can have a logic programming-like proof system presentation for classical, intuitionistic, modal, and substructural logics.

Here is a plan of the chapter: we start revising Horn classical logic as a motivating example , we then consider intuitionistic logic and full classical logic. Then we consider modal logics and substructural logics.

Notation and basic notions

Formulas

By a propositional language $\mathcal{L}$, we denote the set of propositional formulas built from a denumerable set Var of propositional variables by applying the propositional connectives $\neg, \wedge, \vee, \to, \bot$.

Unless stated otherwise, we denote propositional variables (also called *atoms*) by lower case letters, and arbitrary formulas by upper case letters. We assign a *complexity* $cp(A)$ to each formula A (as usual):

$cp(q) = 0$ if q is an atom,
$cp(\neg A) = 1 + cp(A)$,
$cp(A * B) = cp(A) + cp(B) + 1$, where $* \in \{\wedge, \vee, \to\}$.

(*Formula substitution*) We define the notion of substitution of an atom q by a subformula B within a formula A. This operation is denoted by $A[q/B]$.

$$p[q/B] = \begin{cases} p \text{ if } p \neq q \\ B \text{ if } p = q \end{cases}$$

$$(\neg A)[q/B] = \neg A[q/B]$$

$$(A * C)[q/B] = A[q/B] * C[q/B] \text{ where } * \in \{\wedge, \vee, \rightarrow\}.$$

Implicational formulas
In much of the chapter we will be concerned with *implicational formulas*. These formulas are generated from a set of atoms by the only connective $\rightarrow$. We adopt some specific notations for them. We sometimes distinguish the *head* and the *body* of an implicational formula.[2] The head of a formula A is its rightmost nested atom, whereas the body is the *list* of the antecedents of its head. Given a formula A, we define $Head(A)$ and $Body(A)$ as follows:

$Head(q) = q$, if q is an atom,
$Head(A \rightarrow B) = Head(B)$.

$Body(q) = (\,)$, if q is an atom,
$Body(A \rightarrow B) = (A) * Body(B)$,

where $(A) * Body(B)$ denotes the list beginning with A followed by $Body(B)$.

Dealing with implicational formulas, we assume that implication associates on the right, i.e. we write

$$A_1 \rightarrow A_2 \rightarrow \ldots \rightarrow A_{n-1} \rightarrow A_n,$$

instead of

$$A_1 \rightarrow (A_2 \rightarrow \ldots \rightarrow (A_{n-1} \rightarrow A_n) \ldots).$$

It turns out that every implicational formula A can be written as

$$A_1 \rightarrow A_2 \rightarrow \ldots \rightarrow A_n \rightarrow q,$$

where we obviously have.

$$Head(A) = q \quad \text{and} \quad Body(A) = (A_1, \ldots, A_n).$$

[2] This terminology is reminiscent of logic programming [Lloyd, 1984].

2 PROPOSITIONAL HORN DEDUCTION

To explain what we mean by goal-directed deduction style, we begin by recalling standard propositional Horn deductions. This type of deduction is usually interpreted in terms of classical resolution, but it is not the only possible interpretation.[3] The data are represented by a set of propositional Horn clauses, which we write as

$$a_1 \wedge \ldots \wedge a_n \to b.$$

(or as $a_1 \to \ldots \to a_n \to b$, according to the previous convention). The a_i are just propositional variables and $n \geq 0$. In case $n = 0$, the formula reduces to b. This formula is equivalent to:

$$\neg a_1 \vee \ldots \vee \neg a_n \vee b.$$

Let Δ be a set of such formulas, we can give a calculus to derive formulas, called 'goals' of the form $b_1 \wedge \ldots \wedge b_m$. The rules are:

$\Delta \vdash^? b$ succeeds if $b \in \Delta$;
$\Delta \vdash^? A \wedge B$ is reduced to $\Delta \vdash^? A$ and $\Delta \vdash^? B$;
$\Delta \vdash^? q$ is reduced to
$\Delta \vdash^? a_1 \wedge \ldots \wedge a_n$, if there is a clause in Δ of the form

$$a_1 \wedge \ldots \wedge a_n \to q.$$

The main difference from the traditional logic programming convention is that in the latter conjunction is eliminated and a goal is kept as a sequence of atoms $b_1, \ldots, b_m$. The computation does not split because of conjunction, all the subgoals b_i are kept in parallel, and when some b_i succeeds (that is $b_i \in \Delta$) it is deleted from the sequence. To obtain a real algorithm we should specify in which order we scan the database when we search for a clause whose head matches the goal. Let us see an example.

EXAMPLE 1. Let Δ contain the following clauses

(1) $a \wedge b \to g$,
(2) $t \to g$,
(3) $p \wedge q \to t$,
(4) $h \to q$,
(5) $c \to d$,
(6) $c \wedge f \to a$,
(7) $d \wedge a \to b$,
(8) $a \to p$,
(9) $f \wedge t \to h$,
(10) c,
(11) f.

[3] For a survey on foundations of logic programming, we refer to [Lloyd, 1984] and to [Gallier, 1987].

A derivation of g from Δ can be displayed in the form of a tree and it is shown in Figure 1. The number in front of every non-leaf node indicates the clause of Δ which is used to reduce the atomic goal in that node.

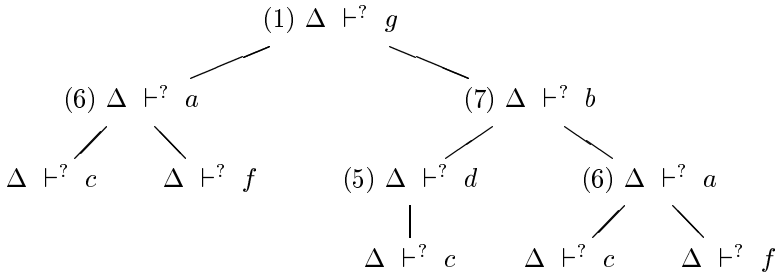


Figure 1. Derivation of Example 1.

We can make a few observations. First, we do not need to consider the whole database, it might even be infinite, and the derivation would be exactly the same; irrelevant clauses, that is those whose 'head' do not match with the current goal are ignored. The derivation is driven by the goal in the sense that each step in the proof simply replaces the current goal with the next one.

Notice also that in this specific case there is no other way to prove the goal, and the sequence of steps is entirely determined.

Two weak points of the method can also be noticed. Suppose that when asking for g we use the second formula, then we continue asking for t, then for h, and then we are lead to ask for t again. We are in a loop. An even simpler situation is the following

$$p \to p \vdash^? p.$$

We can keep on asking for p without realizing that we are in a loop. To deal with this problem we should add a mechanism which ensures termination.

Another problem which has a bearing on the efficiency of the procedure is that a derivation may contain redundant subtrees. This occurs when the same goal is asked several times. In the previous example it happens with the subgoal a. In this case, the global derivation contains multiple subderivations of the same goal. It would be better to be able to remember whether a goal has already been asked (and succeeded) in order to avoid the duplication of its derivation. Whereas the problem of termination is crucial in the evaluation of the method (if we are interested in getting an answer eventually), the problem of redundancy will not be considered in this chapter. However, avoiding the type of redundancy we have described has a

dramatic effect on the efficiency of the procedure, for redundant derivations may grow exponentially with the size of the data.

Although the goal-directed procedure does not necessarily produce the shortest proofs, and does not always terminate, still it has the advantage that proofs, when they exist, are easily found. In the next sections we will see how to extend these type of proof systems to several families of non-classical logics.

3 INTUITIONISTIC AND CLASSICAL LOGICS

3.1 Intuitionistic logic

Intuitionistic logic is the most known alternative to classical logic. For background motivation and information we refer to [Troelstra, 1969; Gabbay, 1981]. The reason why we initiate our tour from intuitionistic logic is *simplicity*. A proof procedure for the propositional implicational fragment of intuitionistic logic is just a minor extension of the Horn case. Morever, the relation with the semantics, the role of cut, the problems, and the possible refinments are better understood for intuitionistic logic.

We recall a Hilbert style axiomatisation of the intuitionistic propositional calculus and the standard Kripke semantics for it. The axiomatization is obtained by considering the following set of axioms and rules we denote by **I**:[4]

1. $(A \to B \to C) \to (A \to B) \to A \to C$
2. $A \to B \to A$
3. $A \to B \to (A \wedge B)$
4. $A \wedge B \to A$
5. $A \wedge B \to B$
6. $(A \to C) \to (B \to C) \to (A \vee B \to C)$
7. $A \to A \vee B$
8. $A \to B \vee A$
9. $\bot \to A$.

In addition, it contains the *Modus Ponens* rule:

$$\frac{\vdash A \quad \vdash A \to B.}{\vdash B}$$

Negation is considered as a derived operator, by $\neg A =^{def} A \to \bot$.

Given a set of formulas Δ, we can define A is derivable from Δ, $\Delta \vdash A$ by the axiom systems above in the customary way.

[4] This axiom system is *separated*, that is to say, any theorem containing $\to$ and a set of connectives $S \subseteq \{\wedge, \vee, \neg, \bot\}$ can be proved by using the implicational axioms together with the axiom groups containing just the connectives in S.

If add to **I** any of the axioms below we get classical logic **C**:

Alternative axioms for classical logic

1. (Peirce's axiom) $((A \to B) \to A) \to A$
2. (double negation) $\neg\neg A \to A$,
3. (excluded middle) $\neg A \vee A$,
4. ($\vee, \to$-distribution) $[(A \to (B \vee C)] \to [(A \to B) \vee C]$.

In particular, the addition of Peirce's axiom to the implicational axioms of intuitionistic logic give us an axiomatisation of classical implication.

We introduce a standard model-theoretic semantics of intuitionistic logic, called Kripke semantics.

DEFINITION 2. Given a propositional language $\mathcal{L}$, a Kripke model for $\mathcal{L}$ is a structure of the form $M = (W, \leq, V)$, where W is a non-empty set, $\leq$ is a reflexive and transitive relation on W, V is a function of type: $W \longrightarrow Pow(Var_{\mathcal{L}})$, that is V maps each element of W to a set of propositional variables of $\mathcal{L}$. We assume the following conditions:

(1) $w \leq w'$ implies $V(w) \subseteq V(w')$;

(2) $\perp \notin V(w)$, for all $w \in W$.

Given $M = (W, \leq, V)$, $w \in W$, for any formula A of $\mathcal{L}$, we define 'A is true at w in M', denoted by $M, w \models A$ by the following clauses:

- $M, w \models q$ iff $q \in V(w)$;
- $M, w \models A \wedge B$ iff $M, w \models A$ and $M, w \models B$;
- $M, w \models A \vee B$ iff $M, w \models A$ or $M, w \models B$;
- $M, w \models A \to B$ iff for all $w' \geq w$, if $M, w' \models A$ then $M, w' \models B$;
- $M, w \models \neg A$ iff for all $w' \geq w$ $M, w' \not\models A$.

We say that A is *valid* in M if $M, w \models A$, for all $w \in W$ and we denote this by $M \models A$. We say that A is *valid* if it is valid in every Kripke model M. We also define a notion of entailment between sets of formulas and formulas. Let $\Gamma = \{A_1, \ldots, A_n\}$ be a set of formulas and B be a formula, we say that Γ *entails* B denoted by $\Gamma \models B$[5] iff for every model $M = (W, \leq, V)$, for every $w \in W$

if $M, w \models A_i$ for all $A_i \in \Gamma$, then $M, w \models B$.

[5]To be precise, we should write $\models_{\mathbf{I}}$ (and $\vdash_{\mathbf{I}}$) to denote validity and entailment (respectively, provability and logical consequence) in intuitionistic logic **I**. To avoid burdening the notation, we usually omit the subscript unless there is a risk of confusion.

THEOREM 3. *For any set of formulas* Γ *and formula* B, $\Gamma \vdash B$ *iff* $\Gamma \models B$. *In particular* $\vdash_{\mathbf{I}} B$ *iff* $\models_{\mathbf{I}} B$.

Classical interpretations can be thought as degenerated Kripke models $M = (W, \leq, V)$, where $W = \{w\}$.

3.2 Rules for intuitionistic implication

We start by presenting a goal-directed system for the implicational fragment of intuitionistic logic. We will then refine it and we will expand it with the other connectives later on. We give rules in this section to prove statements of the form $\Delta \vdash A$, where Δ is a set of implicational formulas and A is an implicational formula. We use the usual conventions and we write Γ, A for $\Gamma \cup \{A\}$ and $\Gamma \cup \Delta$ for $\Gamma \cup \Delta$. Our rules hence manipulate queries Q of the form:

$$\Delta \ \vdash^{?} \ A.$$

We call Δ the database and A the goal of the query Q. We use the symbol $\vdash^{?}$ to indicate that we do not know whether the query succeeds or not. On the other hand the success of Q means that $\Delta \vdash A$ according to intuitionistic logic.

DEFINITION 4.

- (success) $\Delta \ \vdash^{?} \ q$ succeeds if $q \in \Delta$. We say that q is used in this query.
- (implication) from $\Delta \ \vdash^{?} \ A \to B$ step to

  $$\Delta, A \ \vdash^{?} \ B$$

- (reduction) from $\Delta \ \vdash^{?} \ q$
  if $C \in \Delta$, with $C = D_1 \to D_2 \to \ldots \to D_n \to q$
  (that is $Head(C) = q$ and $Body(C) = (D_1, \ldots D_n)$) then step to

  $$\Delta \ \vdash^{?} \ D_i, \text{ for } i = 1, \ldots, n.$$

  We say that C is used in this step.

A *derivation* $\mathcal{D}$ of a query Q is a tree whose nodes are queries. The root of $\mathcal{D}$ is Q, and the successors of every non-leaf query are determined by exactly one applicable rule (*implication or reduction*) as described above.

We say that $\mathcal{D}$ is *successful* if the *success rule* may be applied to every leaf of $\mathcal{D}$. We finally say that a query Q *succeeds* if there is a successful derivation of Q.

By definition, a derivation $\mathcal{D}$ might be an infinite tree. However if $\mathcal{D}$ is successful then it must be finite. This is easily seen from the fact that,

in case of success, the height of $\mathcal{D}$ is finite and every non-terminal node of $\mathcal{D}$ has a finite number of successors, because of the form of the rules. Moreover, the databases involved in a deduction need not be finite. In a successful derivation only a finite number of formulas from the database will be used in the above sense.

Notice that the success of a query is defined in a non-deterministic way: a query succeeds if there is a successful derivation. To transform the proof rules into a deterministic algorithm one should give a method to search a successful derivation tree. In this respect we agree that when we come to an atomic goal we first try to apply the success rule and if it fails we try the reduction rule. Then the only choice we have is to indicate which formula, among those of the database whose head matches the current atomic goal, we use to perform a reduction step, if there are more than one. Thinking of the database as a list of formulas, we can choose the first one and remember the point up to which we have scanned the database as a backtracking point. This is exactly as in conventional logic programming [Lloyd, 1984].

EXAMPLE 5. We check

$$b \to d, a \to p, p \to b, (a \to b) \to c \to a, (p \to d) \to c \vdash b.$$

Let $\Gamma = \{b \to d, a \to p, p \to b, (a \to b) \to c \to a, (p \to d) \to c\}$, a successful derivation of $\Gamma \ \vdash^? \ b$ is shown in Figure 2. A quick explanation: (2) is obtained by reduction wrt. $p \to b$, (3) by reduction wrt. $a \to p$, (4) and (8) by reduction wrt. $(a \to b) \to c \to a$, (6) by reduction wrt. $p \to b$, (7) by reduction wrt. $a \to p$, (9) by reduction wrt. $(p \to d) \to c$, (11) by reduction wrt. $b \to d$, (12) by reduction wrt. $p \to b$.

We state some simple, but important, properties of the deduction procedure defined above. The proof of them is left to the reader as an exercise.

PROPOSITION 6.

1. *(Identity)* $\Delta \ \vdash^? \ G$ *succeeds if* $G \in \Delta$;
2. *(Monotony)* $\Delta \ \vdash^? \ G$ *succeeds implies* $\Delta, \Gamma \ \vdash^? \ G$ *succeeds;*
3. *(Deduction Theorem)* $\Delta \ \vdash^? \ A \to B$ *succeeds iff* $\Delta, A \ \vdash^? \ B$ *succeeds.*

The soundness of the proof procedure with respect to the semantics can be proved easily by induction on the height of the computation.

THEOREM 7. *If* $\Delta \ \vdash^? \ A$ *succeeds then* $\Delta \models A$.

The completeness can be proved in a number of ways. Here we give a semantic proof with respect to the Kripke semantics. The technique is standard: we define a canonical model and we show that provability by the proof procedure coincides with truth in the canonical model.

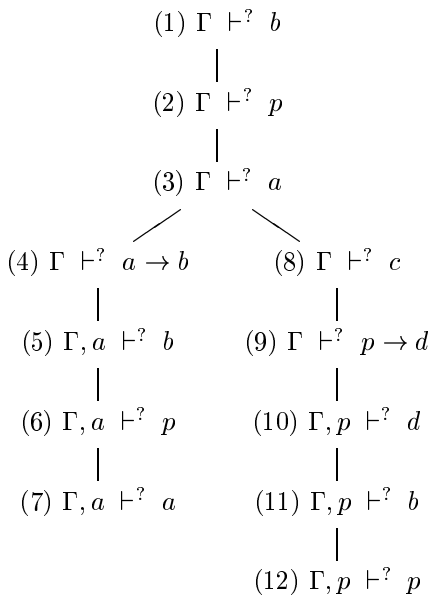


Figure 2. Derivation for Example 5.

The canonical model M is defined as follows: $M = (W, \subseteq, \emptyset, V)$, where W is the set of finite databases Δ over $\mathcal{L}$ and the evaluation function V is defined by stipulating

$$V(\Delta) = \{p \text{ atom} : \Delta \vdash^? p \text{ succeeds}\}.$$

By the (Monotony property) it is easy to see that M satisfies the increasingness condition (2) of Definition 2. The important property is expressed by the following proposition.

PROPOSITION 8 (Canonical Model Property). *For any $\Delta \in W$ and formula $A \in \mathcal{L}$, we have:*

$$M, \Delta \models A \textit{ iff } \Delta \vdash^? A \textit{ succeeds.}$$

Let us attempt a proof of the above proposition, we prove the two directions by a simultaneous induction on the complexity of the formula A. If A is an atom then the claim holds by definition.
Let $A \equiv B \to C$. Consider first the direction ($\Rightarrow$), suppose $M, \Delta \models B \to C$, consider $\Delta' = \Delta, B$. Then $\Delta \subseteq \Delta'$. By (Identity), we have $\Delta' \vdash^? B$ succeeds, by induction hypothesis we get $M, \Delta' \models B$. Thus we get $M, \Delta' \models C$

and by induction hypothesis $\Delta' \vdash^? C$ succeeds. Thus $\Delta, B \vdash^? C$ succeeds. By (Deduction Theorem), we finally get $\Delta \vdash^? B \to C$ succeeds.

Conversely ($\Leftarrow$), suppose that $\Delta \vdash^? B \to C$ succeeds, by (Deduction theorem) we get that $\Delta, B \vdash^? C$ succeeds. Let Γ be such that $\Delta \subseteq \Gamma$ and $M, \Gamma \models B$, we have to show that $M, \Gamma \models C$. By induction hypothesis we have $\Gamma \vdash^? B$ succeeds. We know that

(1) $\Gamma \vdash^? B$ and (2) $\Delta, B \vdash^? C$ succeed.

We could easily conclude if from (1) and (2), we could infer that (3) $\Delta, \Gamma \vdash^? C$ succeeds: namely, since $\Delta \subseteq \Gamma$, (3) is equivalent to $\Gamma \vdash^? C$ succeeds. Thus by induction hypothesis we would get $M, \Gamma \models C$.

The question is: is it legitimate to conclude (3) from (1) and (2)? The answer is 'yes' and it will be shown hereafter. This property is well-known and is called *Cut*. Thus, given the properties of deduction theorem, identity, the canonical model property can be derived from cut. We may observe that the opposite also holds, i.e. that cut can be derived by the canonical model property. To see this suppose that $\Gamma, B \vdash^? C$ and $\Delta \vdash^? B$ succeeds. We get $\Gamma \vdash^? B \to C$ succeeds. Thus by the canonical model property we get $M, \Gamma \models B \to C$ and $M, \Delta \models B$. Since, trivially, $\Delta \subseteq \Gamma \cup \Delta$, by the condition (1) of Definition 2, we have $M, \Gamma \cup \Delta \models B$; so that we obtain $M, \Gamma \cup \Delta \models C$, being $\Gamma \subseteq \Gamma \cup \Delta$. By the canonical model property we can conclude that $\Gamma, \Delta \vdash^? C$ succeeds. The equivalence between the canonical model property and cut has been observed in [Miller, 1992].

The completeness is an immediate consequence of the canonical model property.

THEOREM 9 (Completeness for **I**). *If $\Delta \models A$, then $\Delta \vdash^? A$ succeeds.*

Proof. If $\Delta \models A$ holds, the entailment holds in particular in the canonical model M. thus for every $\Gamma \in W$ if $M, \Gamma \models B$ for every $B \in \Delta$, we have $M, \Gamma \models A$.
By (identity) we have $\Delta \vdash^? B$ succeeds for every $B \in \Delta$. Thus by the canonical model property $M, \Delta \models B$ for every $B \in \Delta$. We hence obtain $M, \Delta \models A$ and by the canonical model property again $\Delta \vdash^? A$ succeeds. ■

We have still to show that cut is admissible.

THEOREM 10 (Admissibility of Cut). *If $\Delta, A \vdash^? B$ and $\Gamma \vdash^? A$ succeed, then also $\Delta, \Gamma \vdash^? B$ succeeds.*

Proof. Assume (1) $\Delta, A \vdash^? B$ and (2) $\Gamma \vdash^? A$ succeed.

The theorem is proved by induction on lexicographically-ordered pairs (c, h), where $c = cp(A)$, and h is the height of a successful derivation of (1), that is of $\Delta, A \vdash^? B$. Suppose first $c = 0$, then A is an atom p, and we proceed by induction on h. If $h = 0$, B is an atom q and either $q \in \Delta$ or

$q = p = A$. In the first case, the claim trivially follows by Proposition 6. In the second case it follows by hypothesis (2) and Proposition 6.

Let now $h > 0$, then (1) succeeds either by the implication rule or by the reduction rule. In the first case, we have that $B = C \to D$ and from $\Delta, A \vdash^? C \to D$ we step to $\Delta, A, C \vdash^? D$, which succeeds by a derivation h' shorter than h. Since $(0, h') < (0, h)$, by the induction hypothesis we get that $\Delta, \Gamma, C \vdash^? D$, succeeds, whence $\Delta, \Gamma \vdash^? C \to D$ succeeds too. Let (1) succeed by reduction with respect to a formula $C \in \Delta$. Since A is an atom, $C \neq A$. Then $B = q$ is an atom. We let $C = D_1 \to \ldots \to D_k \to q$. We have for $i = 1, \ldots, k$

$$\Delta, A \vdash^? D_i \text{ succeeds by a derivation of height } h_i < h.$$

Since $(0, h_i) < (0, h)$, we may apply the induction hypothesis and obtain

$$(a_i)\ \Delta, \Gamma \vdash^? D_i \text{ succeeds, for } i = 1, \ldots, k.$$

Since $C \in \Delta \cup \Gamma$, from $\Delta, \Gamma \vdash^? q$ we can step to (a_i) and succeed. This concludes the case of $(0, h)$.

If c is arbitrary and $h = 0$ the claim is trivial. Let $c > 0$ and $h > 0$. The only difference with the previous cases is when (1) succeeds by reduction with respect to A. Let us see that case. Let

$$A = D_1 \to \ldots \to D_k \to q \text{ and } B = q.$$

Then we have for $i = 1, \ldots, k$ $\Delta, A \vdash^? D_i$ succeeds by a derivation of height $h_i < h$. Since $(c, h_i) < (c, h)$, we may apply the induction hypothesis and obtain

$$(b_i)\ \Delta, \Gamma \vdash^? D_i \text{ succeeds for } i = 1, \ldots, k.$$

By hypothesis (2) we can conclude that

(3) $\Gamma, D_1, \ldots, D_k \vdash^? q$ succeeds by a derivation of arbitrary height h'.

Notice that each D_i has a smaller complexity than A, that is $cp(D_i) = c_i < c$. Thus $(c_1, h') < (c, h)$, and we can cut on (3) and (b_1), so that we obtain

(4) $\Delta, \Gamma, D_2, \ldots, D_k \vdash^? q$ succeeds with some height h''.

Again $(c_2, h'') < (c, h)$, so that we can cut (b_2) and (4). By repeating the same argument up to k we finally obtain

$$\Delta, \Gamma \vdash^? q \text{ succeeds.}$$

This concludes the proof. ■

We have given a semantic proof of the completeness of the proof procedure for the implicational fragment of intuitionistic logic. We have given all details (although they are rather easy) since this easy case works as a paradigm for other logics and richer languages. To summarize, the recipe for the semantic proof is the following: (1) give the right formulation of cut and show that it is admissible, (2) define a canonical model, (3) prove the canonical model property as in Proposition8 and derive the completeness as in Theorem 9.
There are however other ways to prove the completeness depending on the chosen presentation of the logic. If we have a Hilbert-style axiomatization, we can show that every atomic instance of any axiom succeeds, and then show that the set of succeeding formulas is closed under formula substitution and modus ponens (supposing that these properties hold, which is the case for all the logics considered in this chapter). To prove the closure under MP we need again cut. Another possibility, if we have a presentation of the logic in terms of consequence relation rules, like a sequent calculus, we can show that every rules is admissible.

3.3 Loop-free and bounded resource deduction

In the previous section, we have introduced a goal-directed proof method for intuitionistic implicational logic. This method does not give a decision procedure, as it can easily loop: consider the trivial query

$$q \to q \ \vdash^? \ q$$

this query is reduced to itself by the reduction rule, so that the computation does not stop. There are two different strategies to deal with looping computation and termination.

One possibility is to detect the loop and stop the computation as soon as it is detected. The other possibility is to *prevent* any loop by constraining the use of the formulas. To perform loop-checking we need to consider the sequence of goals *and* the relative database from which they have been asked. A moment of reflection shows that it is enough to record only the atomic goals, since the non-atomic ones will always reduce to the same atomic goals by the implication rule. The other relevant information is the database from which they are asked. The same atomic goal may be asked from different databases and such a repetition does not mean that the computation necessarily loops:

EXAMPLE 11.

$$\begin{array}{rcl} & \vdash^? & ((c \to a) \to c) \to (c \to a) \to a \\ (c \to a) \to c & \vdash^? & (c \to a) \to a \\ (c \to a) \to c, c \to a & \vdash^? & a \end{array}$$

$$
\begin{array}{rcl}
(c \to a) \to c, c \to a & \vdash^? & c \\
(c \to a) \to c, c \to a & \vdash^? & c \to a \\
(c \to a) \to c, c \to a, c & \vdash^? & a \\
(c \to a) \to c, c \to a, c & \vdash^? & c
\end{array}
$$

success

As you can see the goal **a** repeats twice along the same (and unique) branch, but the second time is asked from an increased database (c has been added). This does not represent a case of loop; we have a loop when we reask the same (atomic) goal from the same database. To detect the loop we need also to record the involved databases. For each computation branch we could record every pair *(atomic goal-database)* in a history list, so that, before making a reduction step, we check whether the same pair is already in the history list. This loop checking ensures termination. The reason is simple, but important. In the case of intuitionistic logic, the database can be regarded as a *set* of formulas (as we have done) so that adding one or more times the same formula does not matter, i.e. the database does not change. This gives decidability: there cannot be infinitely many different databases occurring in one computation branch (supposing that the initial one is finite) since at most a database can contain all subformulas of the initial query. Thus any loop will be detected.

However, the fact that the database is a *set* of formulas can be used to devise a more efficient loop checking mechanism in which we do not have to record the database itself. The idea is simple: we have a loop whenever we repeats the same goal from the same data. Thus we need to record only the atomic goals which are asked from the same database. Whenever we change the database we clear the history. The database changes (grows) when we add a formula not occurring in it by means of the $\to$-rule. This improved loop-checking procedure has been proposed in [Heudering *et al.*, 1996] to the purpose of obtaining a terminating sequent calculus for intuitionistic logic. Let H be the list of past atomic goals. The computation rules are modified as follows:

Rule 1 for $\to$
$\Delta \vdash^? A \to B, H$ succeeds
if $A \notin \Delta$ and $\Delta, A \vdash^? B, \emptyset$ succeeds.

Rule 2 for $\to$
$\Delta \vdash^? A \to B, H$ succeeds
if $A \in \Delta$ and $\Delta \vdash^? B, H$ succeeds.

Reduction Rule
$\Delta \vdash^? q, H$ succeeds
if $q \notin H$ and for some $C_1 \to \ldots \to C_n \to q$ in Δ we have that for all i
$\Delta \vdash^? C_i, H * (q)$ succeeds.

EXAMPLE 12.

$$\begin{array}{rcl}
 & \vdash^? & ((p \to q) \to q) \to (q \to p) \to p,\ \emptyset \\
(p \to q) \to q & \vdash^? & (q \to p) \to p,\ \emptyset \\
(p \to q) \to q, q \to p & \vdash^? & p,\ \emptyset \\
(p \to q) \to q, q \to p & \vdash^? & q,\ (p) \\
(p \to q) \to q, q \to p & \vdash^? & p \to q,\ (p, q) \\
(p \to q) \to q, q \to p, p & \vdash^? & q,\ \emptyset \\
(p \to q) \to q, q \to p & \vdash^? & p \to q,\ (q) \\
(p \to q) \to q, q \to p & \vdash^? & q,\ (q) \\
 & & fail
\end{array}$$

In this way one is able to detect a loop (the same atomic goal repeats from the same database), without having to record each pair (database goal).

As we have remarked at the beginning of this section loop-checking is not the only way to ensure termination. A loop is created because a formula used in one reduction step remains available for further reduction steps. It can be used as many times as we wish.

Let us adopt the point of view that each database formula can be used at most once. Thus our rule for reduction becomes

from $\Delta \vdash^? q$,
if there is $B \in \Delta$, with $B = C_1 \to \ldots \to C_n \to q$, step to
$\Delta - \{B\} \vdash^? C_i$ for $i = 1, \ldots, n$.

The item B is thus thrown out as soon as it is used.

Let us call such a computation *locally linear computation*, as each formula can be used at most once in each path of the computation. That is why we are using the word 'locally'. One can also have the notion of (globally) linear computation, in which each formula can be used exactly once in the entire computation tree.

Since we take care of usage of formulas, it is natural to regard multiple copies of the same formula as *distinct*. This means that databases can now be considered as *multisets* of formulas. In order to keep the notation simple, we use the same notation as in the previous section. From now on, Γ, Δ, etc. will range on multisets of formulas, and we will also write Γ, Δ to denote the union multiset of Γ and Δ, that is $\Gamma \sqcup \Delta$. To denote a multiset $[A_1, \ldots A_n]$, if there is no risk of confusion we will simply write $A_1, \ldots A_n$.

We present three notions of proof: (1) the goal-directed computation for intuitionistic logic, (2) the locally linear goal-directed computation (LL-computation), (3) linear goal-directed computation.

DEFINITION 13. We give the computation rules for a query: $\Gamma \vdash^? G$, where Γ is a multiset of formulas and G is a formula.

- (success) $\Delta \vdash^? q$ immediately succeeds if the following holds:
  1. for intuitionistic and LL- computation, $q \in \Delta$,
  2. for linear computation $\Delta = q$.
- (implication) From $\Delta \vdash^? A \to B$, we step to

  $$\Delta, A \vdash^? B.$$
- (reduction) If there is a formula $B \in \Delta$ with

  $$B = C_1 \to \ldots \to C_n \to q$$

  then from $\Delta \vdash^? q$, we step, for $i = 1, \ldots, n$ to

  $$\Delta_i \vdash^? C_i,$$

  where the following holds
  1. in the case of intuitionistic computation, $\Delta_i = \Delta$;
  2. in the case of locally linear computation, $\Delta_i = \Delta - [B]$;
  3. in the case of linear computation $\sqcup_i \Delta_i = \Delta - [B]$.

It can be shown that the monotony property holds for the LL-computation whereas it does not for the linear computation. Let $\mathbf{Q}_L$, $\mathbf{Q}_{LL}$ and $\mathbf{Q}_I$ denote respectively the set of succeeding queries in the linear, in the locally linear, and in the intuitionistic computation, then we have

$$\mathbf{Q}_L \subseteq \mathbf{Q}_{LL} \subseteq \mathbf{Q}_I$$

The examples below shows that these inclusions are proper.

EXAMPLE 14.

1. We reconsider Example 11: $c \to a, (c \to a) \to c \vdash^? a$
   The formula $c \to a$ has to be used twice in order for a to succeed. Thus the query fails in the locally linear computation, but it succeeds in the intuitionistic one. This example can be generalized as follows, let:
2. Let $A_0 = c$
   $A_{n+1} = (A_n \to a) \to c$.

   Consider the following query:

   $$A_n, c \to a \vdash^? a$$

   The formula $c \to a$ has to be used $n + 1$ times locally.

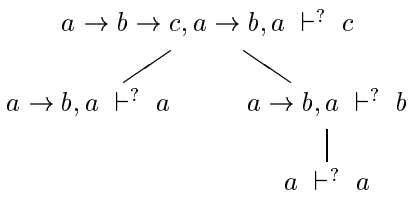


Figure 3. Derivation for Example 15.

EXAMPLE 15. Let us do the full LL-computation for

$$a \to b \to c, a \to b, a \vdash^? c$$

The formula a has to be used twice globally, but not locally on each branch of the computation. Thus the locally linear computation succeeds. On the other hand, in the linear computation case, this query fail, as a must be used globally twice.

The example above shows that the *locally linear* proof system of Definition 13 is not the same as the *linear* proof system. First, we do not require that all assumptions must be used, the condition on the success rule. A more serious difference is that we do our 'counting' of how many times a formula is used separately on each path of the computation and not globally for the entire computation. The counting in the linear case is global, as can be seen by the condition in the reduction rule.

Another example, the query $A, A \to A \to B \vdash^? B$ will succeed in our locally linear computation because A is used once on each of two parallel paths. It will not be accepted in the linear computation because A is used globally twice. This is ensured by the condition in the reduction rule.

Linear computation as defined in Definition 13 corresponds to linear logic implication, [Girard, 1987], in the sense that the procedure of linear computation is sound and complete for the implicational fragment of linear logic. This will be shown in Section 5 within the broader context of substructural logics.

The above examples show that we do not have completeness with respect to intuitionistic provability for locally linear computations. Still, the locally linear computation is attractive, because if the database is finite it is always terminating. We shall see in the next section how we can compensate for the use of the locally linear rule (i.e. for throwing out the data) by some other means. However even if the locally linear computation is not complete for the full intuitionistic implicational fragment, one may still wonder whether it works in some particular and significant case. A significant (and well-known) case is shown in the next proposition.

We recall that a formula A is *Horn* if it is an atom, or has the form $p_1 \to \ldots \to p_n \to q$, where all p_i and q are atoms.

PROPOSITION 16. *The locally-linear procedure is complete for Horn formulas.*

3.4 Bounded restart rule for intuitionistic logic

We have seen that the locally linear restriction does not retain completeness with respect to intuitionistic provability, as there are examples where formulas need to be used locally several times. We show that we can retain completeness for intuitionistic logic by adding another computation rule. The new rule is called the *bounded restart rule*.

Let us examine more closely why we needed in Example 11 the formula $c \to a$ several times. The reason was that from other formulas, we got the goal $\vdash^? a$ and we wanted to use $c \to a$ to continue to the goal $\vdash^? c$. The formula $c \to a$ was no longer available because it had already been used. In other words, $\vdash^? a$ had already been asked and $c \to a$ was used. This means that the next goal after $\vdash^? a$ in the history was $\vdash^? c$.

If H is the history of the atomic goals asked, then somewhere in H there is $\vdash^? a$ and immediately afterwards $\vdash^? c$.

We can therefore compensate for the reuse of $c \to a$ by allowing ourselves to go back in the history to where $\vdash^? a$ was, and allow ourselves to ask all atomic goals that come afterwards. We call this type of move *bounded restart*.

The previous example suggests the following new computation with bounded restart rule.

DEFINITION 17. [Locally linear computation with bounded restart] In the computation with bounded restart, the queries have the form $\Delta \vdash^? G, H$, where Δ is a multiset of formulas and the history H is a sequence of atomic goals. The rules are as follows:

- (success) $\Delta \vdash^? q, H$ succeeds if $q \in \Delta$;
- (implication) from $\Delta \vdash^? A \to B, H$ step to $\Delta, A \vdash^? B, H$;
- (reduction) from $\Delta \vdash^? q, H$ if $C = D_1 \to D_2 \to \ldots \to D_n \to q \in \Delta$, then we step to

  $$\Delta - [C] \vdash^? D_i, H * (q) \text{ for } i = 1, \ldots, n;$$

- (bounded restart) from $\Delta \vdash^? q, H$ step to

  $$\Delta \vdash^? q_1, H * (q),$$

  provided for some H_1, H_2, H_3, it holds $H = H_1 * (q) * H_2 * (q_1) * H_3$, where each H_i may be empty.

EXAMPLE 18.

$$\begin{array}{rcl} & \vdash^? & ((c \to a) \to c) \to (c \to a) \to a,\ \emptyset \\ (c \to a) \to c & \vdash^? & (c \to a) \to a,\ \emptyset \\ (c \to a) \to c, c \to a & \vdash^? & a,\ \emptyset \\ (c \to a) \to c & \vdash^? & c,\ (a) \\ & \vdash^? & c \to a,\ (a, c) \\ c & \vdash^? & a,\ (a, c) \\ c & \vdash^? & c,\ (a, c) \\ & & \textit{success} \end{array}$$

The last step is by bounded restart, it is legal since c follows a in the history.

The locally linear computation with bounded restart is sound and complete for intuitionistic logic. However, before stating the theorem, we want to remark about the meaning of the atoms in the history. To make the things easy, let the query be $\Delta \vdash^? G, (p)$ and G be atomic. Suppose in addition that the query succeeds by a reduction step. Then G will be added to the history list just after p. Let the next goal be p, so that we can apply the bounded restart rule to the query $\Delta \vdash^? p, (p, G)$. The bounded restart step could be performed by reduction if we had the formula $G \to p$ in the database. Thus the original query is equivalent to the query $\Delta, G \to p \vdash^? G$. The general correspondence is expressed in the next theorem.

THEOREM 19 (Soundness and completeness of locally linear computation with bounded restart). *For the computation of Definition 17 we have:* $\Delta \vdash^? G, (p_1, \ldots, p_n)$ *succeeds iff* $\Delta, G \to p_n, p_n \to p_{n-1}, \ldots, p_2 \to p_1 \vdash G$ *in intuitionistic logic.*

3.5 Restart rule for classical logic

It is interesting to note that by adopting a variation of the bounded restart rule we can obtain a proof procedure for implicational classical logic. The variation is obtained by cancelling any restrictions and simply allowing us to ask any earlier atomic goal. We need not keep the history as a sequence, but only as a *set* of atomic goals. The rule becomes

DEFINITION 20 (Restart rule in the LL-computation). If $a \in H$, from $\Delta \vdash^? q, H$ step to $\Delta \vdash^? a, H \cup \{q\}$.

The formal definition of *locally linear computation with restart* is Definition 17 with the additional restart rule above in place of the bounded restart rule.

EXAMPLE 21.

$$\begin{array}{lll} & \vdash^? & (a \to b) \to a \ \vdash^? \ a, \ \emptyset \\ & \vdash^? & \ \vdash^? \ a \to b, \{a\} \\ a & \vdash^? & a \ \vdash^? \ b, \{a\} \\ a & \vdash^? & a, \{a, b\} \text{ by restart} \\ & & \textit{success}. \end{array}$$

The above query fails in the intuitionistic computation. Thus, this example shows that we are getting a logic which is stronger than intuitionistic logic. Namely, we are getting classical logic. This claim has to be properly proved, of course.

If we adopt the basic computation procedure for intuitionistic implication of Definition 4 rather than the LL-computation, we can restrict the restart rule to always choose the *initial goal* as the goal with which we restart. Thus, we do not need to keep the history, but only the initial goal and the rule becomes more deterministic. On the other hand, the price we pay is that we cannot throw out the formulas of database when they are used.

DEFINITION 22 (Simple computation with restart). The queries have the form

$$\Delta \ \vdash^? \ G, (G_0),$$

where G_0 is a goal. The computation rules are the same as in the basic computation procedure for intuitionistic implication of Definition 4 plus the following rule

(Restart) from $\Delta \ \vdash^? \ q, (G_0)$ step to $\Delta \ \vdash^? \ G_0, (G_0)$.

It is clear that the initial query of any derivation will have the form $\Gamma \ \vdash^? A, (A)$.

Given the underlying computation procedure of Definition 4, restarting from an arbitrary atomic goal in the history is equivalent to restart from the initial goal. This is expressed formally in the next proposition, where we let $\vdash^?_{RI}$ and $\vdash^?_{RA}$ be respectively the deduction procedure of Definition 22 and the deduction procedure of Definition 4 extended by the restart rule of Definition 20.

PROPOSITION 23. *For any database Δ and formula G, we have*

(1) $\Delta \ \vdash^?_{RA} \ G, \emptyset$ *succeeds iff (2)* $\Delta \ \vdash^?_{RI} \ G, (G)$ *succeeds.*

We show that the proof-procedure obtained by adding the rule of restart from the initial goal to the basic procedure for intuitionistic logic defined in Definition 4 is sound and complete with respect to classical provability.

The idea is to replace the effect of restart from the initial goal by adding a suitable set of formulas which depends on the initial goal. This set of formulas can be seen as representing the negation (or complement) of the goal.

DEFINITION 24. Let A be any formula. The complement of A, denoted by $Cop(A)$ is the following set of formulas:

$$Cop(A) = \{A \to p \mid p \text{ any atom of the language}\}$$

The set $Cop(A)$ represents the negation of A in our implicational language which does not contain neither $\neg$, nor $\bot$.

The crucial, although easy, fact is that we can replace any application of the restart rule by a reduction step using a formula in $Cop(A)$.

LEMMA 25. *(1)* $\Gamma \vdash^? A$ *succeeds by using restart rule iff (2)* $\Gamma \cup Cop(A) \vdash^? A$ *succeeds without using restart, that is by the intuitionistic procedure of Definition 4.*

Now we only have to show that $\Delta \vdash A$ in classical logic iff $\Delta \cup Cop(A) \vdash^? A$ succeeds by the procedure defined in Definition 4. To this purpose we need the following lemma, which shows that $Cop(A)$ works as the negation of A.

LEMMA 26. *For any database* Δ *and formulas* G *such that* $\Delta \supseteq Cop(G)$*, and for any goal* A*, we have*
if $\Delta \cup \{A\} \vdash^? G$ *and* $\Delta \cup Cop(A) \vdash^? G$ *succeed then also* $\Delta \vdash^? G$ *succeeds.*

THEOREM 27. *For any* Δ *and* A*, (a) is equivalent to (b) below:*

(a) $\Delta \vdash A$ *in classical logic,*

(b) $\Delta \cup Cop(A) \vdash^? A$ *succeeds by the intuitionistic procedure defined in Definition 4.*

Proof. ($\Leftarrow$) Show (b) implies (a).
Assume $\Delta \cup Cop(A) \vdash^? A$ succeeds Then by the soundness of the computation procedure we get that $\Delta \cup Cop(A) \vdash A$ in intuitionistic logic, and hence in classical logic. Since the *proof* is *finite* there is a finite set of the form $\{A \to p_i, \ldots, A \to p_n\}$ such that
(a1) $\Delta, A \to p_1, \ldots A \to p_n \vdash A$ (in intuitionistic logic).
We must also have that $\Delta \vdash A$, in classical logic, because if there were an assignment h making Δ true and A false, it would also make $A \to p_i$ all true, contradicting (a1).

The above concludes the proof that (b) implies (a).

($\Rightarrow$) Show that (a) implies (b).
We prove that if $\Delta \cup Cop(A) \ \vdash^? \ A$ does not succeed then $\Delta \not\vdash A$ in classical logic. Let $\Delta_0 = \Delta \cup Cop(A)$. We define a sequence of databases $\Delta_n, n = 1, 2 \ldots$ as follows:
Let $B_1, B_2, B_3, \ldots$ be an enumeration of all formulas of the language.
Assume Δ_{n-1} has been defined and assume that $\Delta_{n-1} \ \vdash^? \ A$ does not succeed. We define Δ_n:
If $\Delta_{n-1} \cup \{B_n\} \ \vdash^? \ A$ does not succeed, let $\Delta_n = \Delta_{n-1} \cup \{B_n\}$. Otherwise from Lemma 26 we must have:
$\Delta_{n-1} \cup$ Cop $(B_n) \ \vdash^? \ A$ does not succeed.
and so let $\Delta_n = \Delta_{n-1} \cup Cop(B_n)$.
Finally, let $\Delta' = \bigcup_n \Delta_n$
Clearly $\Delta' \ \vdash^? \ A$ does not succeed.

Define an assignment of truth values h on the atoms of the language by $h(p) = \mathbf{true}$ iff $\Delta' \ \vdash^? \ p$ succeeds . We now prove that

for any $B, h(B) = \mathbf{true}$ iff $\Delta' \ \vdash^? \ B$ succeeds,

by induction on B. For atoms this is the definition.
Let $B = C \to D$. We prove the two directions by simultaneous induction. Suppose $\Delta' \ \vdash^? \ C \to D$ succeeds. If $h(C) = \mathbf{false}$, then $h(C \to D) = \mathbf{true}$ and we are done. Thus, assume $h(C) = \mathbf{true}$. By the induction hypothesis, it follows that $\Delta' \ \vdash^? \ C$ succeeds. Since, by hypothesis we have that $\Delta', C \ \vdash^? \ D$ succeeds, by cut we obtain that $\Delta' \ \vdash^? \ D$ succeeds, and hence by the induction hypothesis $h(D) = \mathbf{true}$.

Conversely, if $\Delta' \ \vdash^? \ C \to D$ does not succeed, we show that $h(C \to D) = \mathbf{false}$. Let $Head(D) = q$, we get

(1) $\Delta' \ \vdash^? \ D$ does not succeed
(2) $\Delta', C \ \vdash^? \ q$ does not succeed.

Hence by the induction hypothesis on (1) we have $h(D) = \mathbf{false}$. We show that $\Delta' \ \vdash^? \ C$ must succeed. Suppose on the contrary that $\Delta' \ \vdash^? \ C$ does not succeed. Hence $C \notin \Delta'$. Let $B_n = C$ in the given enumeration. Since $B_n \notin \Delta_n$, by construction, it must be $Cop(C) \subseteq \Delta'$. In particular $C \to q \in \Delta'$, and hence $\Delta', C \ \vdash^? \ q$ succeeds, against (2).

We have shown that $\Delta' \ \vdash^? \ C$ succeeds, whence $h(C) = \mathbf{true}$, by the induction hypothesis. Since $h(C) = \mathbf{false}$, we obtain $h(C \to D) = \mathbf{false}$.

We can now complete the proof. Since $\Delta' \ \vdash^? \ A$ does not succeed, we get $h(A) = \mathbf{false}$. On the other hand, for any $B \in \Delta \cup Cop(A)$, $h(B) = \mathbf{true}$ (since $\Delta \cup Cop(A) \subseteq \Delta'$) and $h(A) = \mathbf{false}$. This means that $\Delta \cup Cop(A) \not\vdash A$ in classical logic. This complete the proof. ■
From the above theorem and Lemma 25 we immediately obtain the completeness of the proof procedure with restart from the initial goal.

THEOREM 28. $\Delta \vdash A$ *in classical logic iff* $\Delta \vdash^? A, (A)$ *succeeds using the restart rule from the initial goal, added to the procedure of Definition 4.*

Also the locally linear computation of Definition 17 with the restart rule from any previous goal is sound and complete. A proof is contained in [Gabbay and Olivetti, 2000].

THEOREM 29. *[Soundness and completeness of locally linear computation with restart]*
$\Delta \vdash^? G, H$ *succeeds iff* $\Delta \vdash G \vee \bigvee H$ *in classical logic.*

Observe that we cannot restrict the application of restart to the first atomic goal occurring in the computation, consider:

$$(p \to q) \to p, p \to r \vdash^? r, \emptyset,$$

it succeeds by the following computation:

$$\begin{array}{rcl} (p \to q) \to p, p \to r & \vdash^? & r,\ \emptyset, \\ (p \to q) \to p & \vdash^? & p, \{r\}, \\ & \vdash^? & p \to q,\ \{r, p\}, \\ p & \vdash^? & q,\ \{r, p\}, \\ p & \vdash^? & p,\ \{r, p\},\quad \text{restart from } p. \end{array}$$

However restarting from r, the first atomic goal would not help.

3.6 Termination and complexity

The proof systems based on locally linear computation are a good starting point for designing efficient automated-deduction procedures; on the one hand proof search is guided by the goal, on the other hand derivations have a smaller size since a formula that has to be reused does not create further branching. We now want to remark upon termination of the procedures.

The basic LL- procedure obviously terminates: since formulas are thrown out as soon as they are used in a reduction step, every branch of a given derivation eventually ends with a query which either immediately succeeds, or no further reduction step is possible from it. This was the motivation of the LL-procedure as an alternative to a loop-checking mechanism. Does the (bounded) restart rule preserve this property? As we have stated, it does not, in the sense that a silly kind of loop may be created by restart. Let us consider the following example, here we give the computation for intuitionistic logic, but the example works for the classical case as well:

$$
\begin{array}{rll}
a \to b, b \to a & \vdash^? & a, \emptyset \\
a \to b & \vdash^? & b, (a) \\
 & \vdash^? & a, (a, b) \\
 & \vdash^? & b, (a, b, a) \\
 & \vdash^? & a, (a, b, a, b) \\
 & \vdots &
\end{array}
$$

This is a loop created by restart. It is clear that continuing the derivation by restart does not help, as none of the new atomic goals match the head of any formula in the database.

In the case of classical logic, we can modify the restart rules as follows.

> From $\Delta \vdash^? q, H$ step to $\Delta \vdash^? q_1, H \cup \{q\}$,
> provided there exists a formula $C \in \Delta$, with $q_1 = Head(C)$,
> and $q \in H$.

It is obvious that this restriction preserves completeness.

In the case of intuitionistic logic, the situation is slightly more complex. The atom with which we finally restart must match the head of some formula of the database in order to make any progress. But this atom might only be reachable through a sequence of restart steps which goes further and further back in the history. To handle this situation, we require that the atom chosen for restart matches some head, but we 'collapse' several restart steps into a single one. In other words, we allow restart from a previous goal q which is accessible from the current one through a sequence of bounded restart steps.

Given a history $H = (q_1, q_2, \ldots, q_n)$ we define an auxiliary binary relation on atomic formulas $q \leq_H q'$ as follows:

1. either q coincides with q', or
2. q precedes q' in the list H,
3. or for some q'' one has $q \leq_H q''$ and $q'' \leq_H q'$.

(In other words, $q \leq_H q'$ iff q' is reachable from q by the reflexive-transitive closure of the binary precedence relation generated by the list H. The modified bounded restart rule for intuitionistic logic becomes: from

$$\Delta \vdash^? q, H,$$

step to

$\Delta \vdash^? q', H*(q)$, provided

1. there exists a formula $C \in \Delta$, with $q' = Head(C)$, $q' \neq q$ and
2. $q \leq_H q'$ holds.

It is easy to see that the above rule ensures the termination of the procedure preserving its completeness.

Moreover we can reformulate the rules for success and reduction by building the bounded restart rule into them, to this purpose we simply restate:

1. (br-success) $\Gamma \vdash^? q, H$ succeeds if there exists q' such that $q \leq_H q'$ and $q' \in \Gamma$;
2. (br-reduction) From $\Gamma, C_1 \to \ldots \to C_n \to q' \vdash^? q, H$ if $q \leq_H q'$ steps for $i = 1, \ldots, n$ to $\Gamma \vdash^? C_i, H*(q)$.

The proof procedure can be further refined to match the known complexity bound for intuitionistic logic, namely $O(n \log n)$ space. Observe that the history list may be kept linear in the length of the database+goal: only the leftmost and the rightmost occurrence of any atom in H are needed for determining $\leq_H$. Thus the history lenght is bounded by $2*k$ where k is the number of atoms occurring in the initial query. The length of each derivation branch is bounded by the length of the initial query and so is the length of each intermediate query. In serching a proof of a given query, we first apply the implication rule if the goal is an implication; if the goal is atomic we try first (br-success) and if it is not applicable we try (br-reduction).

The proof search space can be then described as a tree that contains AND branchings, corresponding to the (br-reduction steps) with multiple subgoals, and OR branchings corresponding to backtracking points, determined by alternative formulas which can be used in the (br-reduction) steps. The latter are branchings in the proof search space, not in the derivation tree. We assume that subgoals are examined and alternatives are scanned in a fixed manner (for instance from left to right).

To achieve a good space complexity bound, we do not store the whole derivation, we rather perform the proof search in a depth first manner expanding one query at a time. We only store one query at a time, the one which is going to be expanded by the rules. Moreover we keep a copy of the initial query. In addition we use a stack to keep track of the AND branchings and backtracking points, if any. We will not enter into the details of how to store the relevant information in the stack entries, it is described in [Gabbay *et al.*, 1999]. We only observe that: (1) since we can index formulas and subformulas of the initial query, each stack entry will not require more than $O(\log n)$ bits, being n is the length of the initial query. (2) we have a stack entry for each query occurring along a derivation branch. Thus the depth of the stack is bounded by n the length of the initial query. In the whole, an

algorithm to search a derivation will not need more than $O(n \log n)$ space, where n is the length of the initial query. This matches the known optimal upper-bound for intuitionistic logic.

THEOREM 30. *The procedure with bounded restart gives an $O(n \log n)$-space decision procedure for the implicational fragment of intuitionistic logic.*

The proof procedure based on bounded restart is almost deterministic except for one crucial point: the choice of the database formula to use in a reduction step. Here a sharp difference between classical and intuitionistic logic arises. In intuitionistic logic, the choice is critical: we could make the wrong choice and then have to backtrack to try an alternative formula. In the case of classical logic, backtracking is not necessary, that is, it does not matter which formula we choose to match an atomic goal in a reduction step.

LEMMA 31. *Let*

$$A = A_1 \to \ldots \to A_n \to q \text{ and } B = B_1 \to \ldots \to B_m \to q.$$

Then (a) is equivalent to (b):

(a) $\Delta, A \vdash^? B_i, H \cup \{q\}$ *succeeds for* $i = 1, \ldots m$;

(b) $\Delta, B \vdash^? A_i, H \cup \{q\}$ *succeeds for* $i = 1, \ldots, n$.

By the previous lemma we immediately have.

PROPOSITION 32. *In any computation of $\Delta \vdash^? q, H$ with restart, no backtracking is necessary. The atom q can match with the head of any $A_1 \to \ldots \to A_n \to q \in \Delta$ and success or failure does not depend on the choice of such a formula.*

The parallel property to Lemma 31, Proposition 32 clearly does not hold for the intuitionistic case. This difference gives an intuitive account of the difference of complexity between the intuitionistic and the classical case.[6]

3.7 Extending the language

Conjunction and negation

In this and the next section we extend the language to the full propositional language. We start by considering conjunction. The addition of conjunction to the propositional language does not change the proof system much. Every formula A with conjunctions is equivalent in intuitionistic logic to a conjunction of formulas $\bigwedge_i A_i$, where A_i contain no conjunctions. If we

[6] We recall that intuitionistic provability is PSPACE-complete [Statman, 1979], whereas classical provability is CoNP-complete, although the space requirements are the same.

agree to represent a conjunction of formulas as a *set*, the handling of conjunction is straightforward, we can transform every database and goal into sets of formulas without conjunction. Then we extend the proof procedure to sets of goals S:

From $\Delta \vdash^? S$ step to $\Delta \vdash^? A$ for every $A \in S$.

The computation rule for conjunction can be stated directly:
$\Delta \vdash^? A \wedge B$ succeeds iff $\Delta \vdash^? A$ succeeds and $\Delta \vdash^? B$ succeeds.

We now turn to *negation*. As we have seen, negation can be introduced in classical and intuitionistic logic by adding a constant symbol $\bot$ for falsity and defining the new connective $\neg A$ for negation as $A \to \bot$. We will adopt this definition. However, we have to modify the computation rules, because we have to allow for the special nature of $\bot$, namely that $\bot \vdash A$ holds for any A.

DEFINITION 33. [Computations for data and goal containing $\bot$ for intuitionistic and classical logic] The basic procedure is that one defined in 4, (plus the restart rule for classical logic), with the following modifications:

1. Modify (success) rule to read: $\Delta \vdash^? q$ immediately succeeds, if $q \in \Delta$ or $\bot \in \Delta$.

2. Modify (reduction rule) to read: from $\Delta \vdash^? q$ step, for $i = 1, \ldots, n$ to

   $$\Delta \vdash^? B_i$$

   if there is $C \in \Delta$ such that $Head(C) \in \{q, \bot\}$ and $Body(C) = (B_1, \ldots, B_n)$.

In Definition 33 we have actually defined two procedures. One is the computation without the restart rule for intuitionistic logic with $\bot$, and the other is the computation with the restart rule for classical logic. We have to show that the two procedures indeed correctly capture the intended fragment of the respective systems. This is easy to see. The effect of the axiom $\bot \vdash A$ is built into the computation via the modifications in 1. and 2. of Definition 33 and hence we know we are getting intuitionistic logic. To show that the restart rule yields classical logic, it is sufficient to show that the computation

$$(A \to \bot) \to \bot \vdash^? A$$

always succeeds with the restart rule. This can also be easily checked.

To complete the picture we show in the next proposition that the computation of $\Delta \vdash^? A$ with restart is the same as the computation of $\Delta \vdash^? (A \to \bot) \to A$ without restart. This means that the restart rule

(with original goal A) can be effectively implemented by adding $A \to \bot$ to the database and using the formula $A \to \bot$ and the $\bot$-rules to replace uses of the restart rule. The above considerations correspond to the known translation from classical logic to intuitionistic logic, namely:

$\Delta \vdash A$ in classical logic iff $\Delta \vdash \neg A \to A$ in intuitionistic logic.

The proof is similar to that one of Lemma 25, namely $Cop(G)$ is a way of representing $G \to \bot$ without using $\bot$.

PROPOSITION 34. *For any database Δ and goal G:*

$\Delta \vdash^? G$ *succeeds with restart iff* $\Delta \cup \{G \to \bot\} \vdash^? G$ *succeeds without restart.*

EXAMPLE 35. We check:

$(q \to \bot) \to \bot \vdash^? q, (q)$

$(q \to \bot) \to \bot \vdash^? q \to \bot, (q)$

$(q \to \bot) \to \bot, q \vdash^? \bot, (q).$

We cannot use the reduction rule here. So, we fail in intuitionistic logic. In classical logic we can use restart to obtain:

$(q \to \bot) \to \bot, q \vdash^? q, (q)$

and terminate successfully.

The locally linear computation with bounded restart (respectively restart) is complete for the $(\to, \bot, \wedge)$-fragment of intuitionistic (classical) logic. The termination and complexity analysis of the previous section applies also to this larger fragment.

Extension to the whole propositional intuitionistic logic

To obtain a goal-directed proof method for full intuitionistic propositional logic we must find a way of handling disjunctive data. The handling of disjunction is more difficult than the handling of conjunction and negation. Consider the formula $a \to (b \vee (c \to d))$. We cannot rewrite this formula in intuitionistic logic to anything of the form $B \to q$, where q is atomic (or $\bot$).

We therefore have to change our proof procedures to accommodate the general form of an intuitionistic formula with disjunction.

In classical logic disjunctions can be pulled to the outside of formulas using the following equivalences:

1. $(A \vee B \to C) \equiv (A \to C) \wedge (B \to C)$

2. $(C \to A \vee B) \equiv (C \to A) \vee (C \to B)$,

where $\equiv$ denotes logical equivalence in classical logic. 1. is valid in intuitionistic logic but 2. is not valid. We have seen at the beginning of the section, the axioms governing disjunction. In view of those axioms, it is not difficult to devise rules to handle disjunction:

R1: from $\Delta \vdash^? A \vee B$ step to $\Delta \vdash^? A$ or to $\Delta \vdash^? B$.

R2: from $\Delta, A \vee B \vdash^? C$ step to $\Delta, A \vdash^? C$ and to $\Delta, B \vdash^? C$.

We can try to incorporate the two rules for disjunction within a goal-directed proof procedure for full intuitionistic logic.

DEFINITION 36. Computation rules for full intuitionistic logic with disjunction.

1. The propositional language contains the connectives $\wedge, \vee, \to, \bot$. Formulas are defined inductively as usual.
2. We define the operation $\Delta + A$, for any formula $A = \bigwedge_i A_i$, as follows: $\Delta + A = \Delta \cup \{A_i\}$ provided A_i are not conjunctions.
3. The computation rules are as follows.

   (suc) $\Delta \vdash^? q$ succeeds if $q \in \Delta$ or $\bot \in \Delta$;

   (conj) from $\Delta \vdash^? A \wedge B$ step to $\Delta \vdash^? A$ and to $\Delta \vdash^? B$;

   (g-dis) from $\Delta \vdash^? A \vee B$ step to $\Delta \vdash^? A$ or to $\Delta \vdash^? B$;

   (imp) from $\Delta \vdash^? A \to B$ step to $\Delta + A \vdash^? B$;

   (red) from $\Delta \vdash^? G$ if G is an atom q or $G = A \vee B$, if $C \in \Delta$, with $C = A_1 \to \ldots A_n \to D$ (where D is not an implication) step to

   (a) $\Delta \vdash^? A_i$, for $i = 1, \ldots, n$, and to

   (b) $\Delta + D \vdash^? G$.

   (c-dis) from $\Delta, A \vee B \vdash^? C$ step to $\Delta + A \vdash^? C$ and to $\Delta + B \vdash^? C$.

The above rules give a sound and complete system for full intuitionistic logic. However the rules are far from satisfactory, in the sense that the goal-directness is lost. For instance, we must be allowed to perform a reduction step not only when the goal is atomic, but also when it is a disjunction, as in the following case

$A, A \to B \vee C \vdash^? B \vee C$.

Similarly, even if the goal is an atom q in the reduction case (red) we cannot require that D in the formula $A_1 \to \ldots A_n \to D$ is atomic and $D = q$. If there are disjunctions in the database, at every step we can choose to work on the goal or to split the database by (c-dis) rule. Moreover, if there are n disjunctions a systematic application of (c-dis) rule yields 2^n branches. All of this means that if we handle disjunction in the most obvious way we loose the goal-directedness of deduction and the computation becomes very inefficient.

The reason is that if *positive* disjunctions are allowed in a database Γ it is not true that

(dp) $\Gamma \vdash A \vee B$ implies $\Gamma \vdash A$ or $\Gamma \vdash B$.

This property, called *disjunction property*, holds when Γ does not contain positive disjunctions, but fails when it does contain them, as the example above shows. This means that the goal $A \vee B$ cannot be decomposed/reduced to A and to B. In other words, we cannot proceed in a goal-directed fashion. There are three ways to overcome the problem of disjunction. The simplest solution is to kill the problem at the root: do not allow positive occurrences of disjunction in the database. To prevent positive disjunctions means to restrict our consideration to so-called *Harrop formulas*. To introduce them, let us define the two types of formulas D and G by mutual induction as follows:

$$D := \; q \mid \bot \mid G \to D \mid D \wedge D$$
$$G := \; q \mid \bot \mid G \wedge G \mid G \vee G \mid D \to G.$$

A formula is *Harrop* if it is defined according to the D-clauses above. D-formulas are allowed as constituents of the database, whereas G formulas are allowed to be asked as goals. It is easy to extend the goal directed procedure to Harrop formulas. A database will be a set of D-formulas (which are not conjunctions themselves). We just add the rule R1 given above to handle disjunctive goals. This gives us a complete system, which can be optimized by adopting the diminishing resource approach and bounded restart.

Another solution, that we just mention, is to eliminate disjunction by adopting Statman's translation [Statman, 1979]: we can translate every pair database-goal (Γ, G) in a pair (Γ^*, G^*), such that Γ^*, G^* do not contain disjunction, but contain additional atoms, and it holds (in intuitionistic logic)

$\Gamma \vdash G$ iff $\Gamma^* \vdash G^*$.

We can then use the proof procedure without disjunction.

However, one can try to cope with the whole propositional intuitionistic logic without limitations, by using additional machinery. There are two difficulties to define a goal-directed procedure. Consider the query $\Gamma \;\; \vdash^?$

$A \vee B$. We can adopt the rule R1 by continuing for instance with $\Gamma \vdash^? A$, and remember that at a previous step we could have chosen the disjunct B, thus we must be able to go back to $\Gamma \vdash^? B$. The use of history and restart can accomplish the necessary book-keeping mechanism. However, we must take into account that the database may have changed in the meantime, and we must go back to the right database; notice that in general

$$(A \to B) \vee C \not\equiv A \to B \vee C \not\equiv B \vee (A \to C)$$

(although these equivalences hold in classical logic). One way to keep track of the dependency between the goal and database from which it is asked is to use labels. This solution is developed in [Gabbay and Olivetti, 2000], where a labelled goal-directed proof procedure for full intuitionistic logic is given. The labels are partially ordered and can be interpreted as possible worlds.

The other technical trick is to extend suitably the notion of 'Head'of a formula to formulas with positive disjunctions; this is necessary to define the reduction step. For instance, ignoring the labelling and restart mechanism, the query Γ, K, where $K = A \to (B \vee (C \to q)) \vdash^? q$, and q is an atom, would be reduced to the queries:

$$\Gamma, K \vdash^? A,$$
$$\Gamma, K, B \vdash^? q,$$
$$\Gamma, K, C \to q \vdash^? C.$$

and q would be recorded in the history.

We shall not give the details of the procedure which can be found in [Gabbay and Olivetti, 2000]. A similar, although much simpler, procedure can be given for classical logic. However, in classical logic the treatment of disjunctive data is not problematic, since on the one hand we can define disjunction using the other connectives (the $\to$ connective alone suffices as $A \vee B \equiv (A \to B) \to B$). On the other hand every formula can be rewritten as a set of clauses of the form:

$$p_1 \wedge \ldots \wedge p_n \to q_1 \vee \ldots \vee q_m$$

where $n \geq 0$, and $m > 0$, every p_i is an atom, and every q_j is an atom or is $\bot$. For data of this sort, a goal-directed procedure is easily designed, see [Gabbay and Olivetti, 2000; Nadathur, 1998; Loveland, 1991; Loveland, 1992].

3.8 Some history

A goal-directed proof system for a (first-order) fragment of intuitionistic and classical logic was first given by Gabbay [Gabbay and Reyle, 1984;

Gabbay, 1985] as a foundation of hypothetical logic programming. A similar system for intuitionistic logic was proposed in [McCarty, 1988a; McCarty, 1988b]. The restart rule was first proposed by Gabbay in his lecture notes in 1984 and the first theoretical results published in [Gabbay, 1985] and then 1985 in [Gabbay and Kriwaczek, 1991].[7] A similar idea to restart has been exploited by Loveland [1991; 1992], in order to extend conventional logic programming to non-Horn databases; in Loveland's proof procedure the restart rule is a way of implementing reasoning by case-analysis.

The concept of goal-directed computation can also be seen as a generalization of the notion of *uniform proof* as introduced in [Miller *et al.*, 1991]. A uniform proof system is called 'abstract logic programming' [Miller *et al.*, 1991]. The extension of the uniform proof paradigm to classical logic is recently discussed in [Harland, 1997] and [Nadathur, 1998]. The essence of a uniform-proof system is the same underlying the goal-directed paradigm: the proof-search is driven by the goal and the connectives can be interpreted directly as search instructions.

The locally linear computation with bounded restart was first presented by Gabbay [1991] and then further investigated in [Gabbay, 1992], where goal-directed procedures for classical and some intermediate logics are also presented. This refinment is strongly connected to the contraction-free sequent calculi for intuitionistic logic which have been proposed by many people: Dyckhoff [1992], Hudelmaier [1990], and Lincoln *et al.*, [1991]. To see the intuitive connection, let us consider the query:

$$(1)\ \Delta, (A \to p) \to q\ \vdash^?\ q, \emptyset$$

we can step by reduction to $\Delta\ \vdash^?\ A \to p, (q)$ and then to $\Delta, A\ \vdash^?\ p, (q)$, which, by the soundness corresponds to

$$(2)\ \Delta, A, p \to q\ \vdash^?\ q.$$

In all the mentioned calculi (1) can be reduced to (2) by a sharpened left-implication rule (here used backwards). This modified rule is the essential ingredient to obtain a contraction-free sequent calculus for **I**, at least for its implicational fragment. A formal connection with these contraction-free calculi has not been studied yet. It might turn out that LL-computations correspond to *uniform proofs* (in the sense of [Miller *et al.*, 1991]) within these calculi.

In [Gabbay and Olivetti, 2000] the goal-directed methods are extended to some *intermediate logics*, i.e. logics which are between intuitionistic and classical logics. In particular, it is given a proof procedure for the family of intermediate logics of Kripke models with finite height, and for Dummett-Gödel logic LC. These proof systems are obtained by adding suitable restart rules to the intuitionistic system.

[7] The lecture notes have evolved also into the book [Gabbay, 1998].

4 MODAL LOGICS OF STRICT IMPLICATION

In this section we examine how to extend the goal-directed proof methods to modal logics. We begin considering a minimal language which contains only *strict implication*, we will then extend it to the full language of modal logics. Strict implication, denoted by $A \Rightarrow B$ is read as 'necessarily A implies B'. The notion of necessity (and the dual notion of possibility) are the subject of modal logics. Strict implication can be regarded as a derived notion: $A \Rightarrow B = \Box(A \rightarrow B)$, where $\rightarrow$ denotes material implication and $\Box$ denotes modal necessity. However, strict implication can also be considered as a primitive notion, and has already been considered as such at the beginning of the century in many discussions about the paradoxes of material implication [Lewis, 1912; Lewis and Langford, 1932].

The extension of the goal-directed approach to strict implication and modal logics relies upon the *possible worlds* semantics of modal logics which is mainly due to Kripke.

The strict implication language $\mathcal{L}(\Rightarrow)$ contains all formulas built out from a denumerable set *Var* of propositional variables by applying the strict implication connective, that is, if $p \in Var$ then p is a formula of $\mathcal{L}(\Rightarrow)$, and if A and B are formulas of $\mathcal{L}(\Rightarrow)$, then so is $A \Rightarrow B$. Let us fix an atom p_0, we can define the constant $\top \equiv p_0 \Rightarrow p_0$ and let $\Box A \equiv \top \Rightarrow A$.

Semantics

We review the standard Kripke semantics for $\mathcal{L}(\Rightarrow)$.

A Kripke structure M for $\mathcal{L}(\Rightarrow)$ is a triple (W, R, V), where W is a non-empty set (whose elements are called possible worlds), R is a binary relation on W, and V is a mapping from W to sets of propositional variables of $\mathcal{L}$. Truth conditions for formulas (of $\mathcal{L}(\Rightarrow)$) are defined as follows:

- $M, w \models p$ iff $p \in V(w)$;
- $M, w \models A \Rightarrow B$ iff for all w' such that wRw' and $M, w' \models A$, it holds $M, w' \models B$.

We say that a formula A is *valid* in a structure M, denoted by $M \models A$, if $\forall w \in W,\ M, w \models A$. We say that a formula A is *valid* with respect to a given class of structures $\mathcal{K}$, iff it is valid in every structure $M \in \mathcal{K}$. We sometimes use the notation $\models_{\mathcal{K}} A$. Let us fix a class of structures $\mathcal{K}$. Given two formulas A and B, we can define the *consequence relation* $A \models_{\mathcal{K}} B$ as

$$\forall M = (W, R, V) \in \mathcal{K} \forall w \in W \text{ if } M, w \models_{\mathcal{K}} A \text{ then } M, w \models_{\mathcal{K}} B.$$

Different modal logics are obtained by considering classes of structures whose relation R satisfies some specific properties. The properties of the accessibility relations we consider are listed in Table 1.

Table 1. Standard properties of the accessilbility relation.

Reflexivity	$\forall x\ xRx$
Transitivity	$\forall x \forall y \forall z\ xRy \wedge yRz \rightarrow xRz$
Symmetry	$\forall x \forall y\ xRy \rightarrow yRx$
Euclidean	$\forall x \forall y \forall z\ xRy \wedge xRz \rightarrow yRz$

Table 2. Some standard modal logics.

Name	Reflexivity	Transitivity	Symmetry	Euclidean
K				
KT	*			
K4		*		
S4	*	*		
K5				*
K45		*		*
KB			*	
KTB	*		*	
S5	*	*	*	*

We will take into consideration strict implication $\Rightarrow$ as defined in systems **K**, **KT**,[8] **K4**, **S4**, **K5**, **K45**, **KB**, **KBT**, and **S5**.[9]

Properties of accessibility relation R in Kripke frames, corresponding to these systems are shown in Table 2.

Letting **S** be one of the modal systems above, we use the notation $\models_{\mathbf{S}} A$ (and $A \models_{\mathbf{S}} B$) to denote validity in (and the consequence relation determined by) the class of structures corresponding to **S**.

4.1 Proof systems

In this section we present proof methods for all modal systems mentioned above. We regard a database as a set of labelled formulas $x_i : A_i$ equipped by a relation α giving connections between labels. The labels obviously represent worlds. Thus, $x_i : A_i$ means that A_i holds at x_i. The goal of a query is asked with respect to a world. The form of databases and goals determine the notion of consequence relation

$$\{x_1 : A_1, \ldots, x_n : A_n\}, \alpha \vdash x : A$$

[8] We use the acronym **KT** rather than the more common **T**, as the latter is also the name of a subrelevance logic we will meet in Section 5.

[9] We do not consider here systems containing $D : \Box A \rightarrow \Diamond A$, which correspond to the *seriality* of the accessibility relation, i.e. $\forall x \exists y\ xRy$ in Kripke frames. The reason is that seriality cannot be expressed in the language of strict implication alone; moreover, it cannot be expressed in any modal language, unless $\neg$ or $\Diamond$ is allowed.

whose intended meaning is that if A_i holds at x_i (for $i = 1, \ldots, n$) and the x_i are connected as α prescribes, then A must hold at x.

For different logics α will be required to satisfy different properties such as reflexivity, transitivity, etc., depending on the properties of the accessibility relation of the system under consideration.

DEFINITION 37. Let us fix a denumerable alphabet $\mathcal{A} = \{x_1, \ldots, x_i, \ldots\}$ of labels. A *database* is a finite graph of formulas labelled by $\mathcal{A}$. We denote a database as a pair (Δ, α), where Δ is a finite set of labelled formulas $\Delta = \{x_1 : A_1, \ldots, x_n : A_n\}$ and $\alpha = \{(x_1, x_1'), \ldots, (x_m, x_m')\}$ is a set of links. Let $Lab(E)$ denote the set of labels $x \in \mathcal{A}$ occurring in E, and assume that (i) $Lab(\Delta) = Lab(\alpha)$, and (ii) if $x : A \in \Delta, x : B \in \Delta$, then $A = B$.[10]

A *trivial database* has the form $(\{x_0 : A\}, \emptyset)$.

The *expansion* of a database (Γ, α) by $y : C$ at x, with $x \in Lab(\Gamma)$, $y \notin Lab(\Gamma)$ is defined as follows:

$$(\Gamma, \alpha) \oplus_x (y : C) = (\Delta \cup \{y : C\}, \alpha \cup \{(x, y)\}).$$

DEFINITION 38. A query Q is an expression of the form:

$$Q = (\Delta, \alpha) \vdash^? x : G, H$$

where (Δ, α) is a database, $x \in Lab(\Delta)$, G is a formula, and H, *the history*, is a set of pairs

$$H = \{(x_1, q_1), \ldots, (x_m, q_m)\}$$

where x_i are labels and q_i are atoms. We will often omit the parentheses around the two components of a database and write $Q = \Delta, \alpha \vdash^? x : G, H$. A query from a trivial database $\{x_0 : A\}$ will be written simply as:

$$x_0 : A \vdash^? x_0 : B, H,$$

and if $A = \top$, we sometimes just write $\vdash^? x_0 : B, H$.

DEFINITION 39. Let α be a set of links, we introduce a family of relation symbols $A^{\mathbf{S}}_\alpha(x, y)$, where $x, y \in Lab(\alpha)$. We consider the following conditions:

(**K**) $(x, y) \in \alpha \Rightarrow A^{\mathbf{S}}_\alpha(x, y)$,
(**T**) $x = y \Rightarrow A^{\mathbf{S}}_\alpha(x, y)$,
(**4**) $\exists z(A^{\mathbf{S}}_\alpha(x, z) \wedge A^{\mathbf{S}}_\alpha(z, y)) \Rightarrow A^{\mathbf{S}}_\alpha(x, y)$,
(**5**) $\exists z(A^{\mathbf{S}}_\alpha(z, x) \wedge A^{\mathbf{S}}_\alpha(z, y)) \Rightarrow A^{\mathbf{S}}_\alpha(x, y)$,
(**B**) $A^{\mathbf{S}}_\alpha(x, y) \Rightarrow A^{\mathbf{S}}_\alpha(y, x)$.

[10] We will drop this condition in Section 4.3 when we extend the language by allowing conjunction.

For $\mathbf{K} \in \mathbf{S} \subseteq \{\mathbf{K}, \mathbf{T}, \mathbf{4}, \mathbf{5}, \mathbf{B}\}$, we let $A^{\mathbf{S}}$ be the least relation satisfying all conditions in $\mathbf{S}$. Thus, for instance, $A^{\mathbf{K45}}$ is the least relation such that:

$$\begin{aligned} A^{\mathbf{K45}}_{\alpha}(x,y) \quad \Leftrightarrow \quad & (x,y) \in \alpha \vee \\ & \vee \exists z(A^{\mathbf{K45}}_{\alpha}(x,z) \wedge A^{\mathbf{K45}}_{\alpha}(z,y)) \vee \\ & \vee \exists z(A^{\mathbf{K45}}_{\alpha}(z,x) \wedge A^{\mathbf{K45}}_{\alpha}(z,y)). \end{aligned}$$

We will use the standard abbreviations (i.e. $A^{\mathbf{S5}} = A^{\mathbf{KT5}} = A^{\mathbf{KT45}}$).

DEFINITION 40 (Modal Deduction Rules). For each modal system $\mathbf{S}$, the corresponding proof system, denoted by $\mathrm{P}(\mathbf{S})$, comprises the following rules parametrized to predicates $A^{\mathbf{S}}$:

- (success) $\Delta, \alpha \ \vdash^{?} \ x : q, H$ immediately succeeds if q is an atom and $x : q \in \Delta$.
- (implication) From $\Delta, \alpha \ \vdash^{?} \ x : A \Rightarrow B, H$, step to

  $$(\Delta, \alpha) \oplus_x (y : A) \ \vdash^{?} \ y : B, H,$$

  where $y \notin \mathit{Lab}(\Delta) \cup \mathit{Lab}(H)$.
- (reduction) If $y : C \in \Delta$, with $C = B_1 \Rightarrow B_2 \Rightarrow \ldots \Rightarrow B_k \Rightarrow q$, with q atomic, then from

  $$\Delta, \alpha \ \vdash^{?} \ x : q, H$$

  step to

  $$\begin{aligned} & \Delta, \alpha \ \vdash^{?} \ u_1 : B_1, H \cup \{(x,q)\}, \\ & \vdots, \\ & \Delta, \alpha \ \vdash^{?} \ u_k : B_k, H \cup \{(x,q)\}, \end{aligned}$$

  for some $u_0, \ldots, u_k \in \mathit{Lab}(\alpha)$, with $u_0 = y$, $u_k = x$, such that

  for $i = 0, \ldots, k-1$, $A^{\mathbf{S}}_{\alpha}(u_i, u_{i+1})$ holds.
- (restart) If $(y, r) \in H$, then, from $\Delta, \alpha \ \vdash^{?} \ x : q, H$, with q atomic, step to

  $$\Delta, \alpha \ \vdash^{?} \ y : r, H \cup \{(x,q)\}.$$

Restricted restart

Similar to the case of classical logic, in any deduction of a query Q of the form $\Delta, \alpha \ \vdash^{?} \ x : G, \emptyset$, the *restart rule* can be restricted to the choice of the pair (y, r), such that r is the uppermost atomic goal occurred in the deduction and y is the label associated to r (that is, the query in which r appears contains $\ldots \ \vdash^{?} \ y : r$). Hence, if the initial query is $Q = \ \Delta, \alpha \ \vdash^{?}$

$x : G, \emptyset$ and G is an atom q, such a pair is (x, q), if G has the form $B_1 \Rightarrow \ldots \Rightarrow B_k \Rightarrow r$, then the first pair is obtained by repeatedly applying the implication rule until we reach the query $\ldots \vdash^? x_k : r$, with $x_k \notin Lab(\Delta)$. With this restriction, we do not need to keep track of the history any more, but only of the first pair. An equivalent formulation is to allow restart from the initial goal (and its relative label) even if it is implicational, but the re-evaluation of an implication causes a redundant increase of the database, that is why we prefer the above formulation.

PROPOSITION 41. *If* $\Delta, \alpha \vdash^? x : G, \emptyset$ *succeeds then it succeeds by a derivation in which every application of restart is* restricted restart.

We have omitted the reference to the specific proof system P(**S**), since of the previous claim does not depend on the specific properties of the predicates $A^{\mathbf{S}}$ involved in the definition of a proof system P(**S**). We will omit the reference to P(**S**) whenever it is not necessary.

We show some examples of the proof procedure.

EXAMPLE 42. In Figure 4 we show a derivation of

$$((p \Rightarrow p) \Rightarrow a \Rightarrow b) \Rightarrow (b \Rightarrow c) \Rightarrow a \Rightarrow c.$$

in P(**K**). By Proposition 41, we only record the first pair for restart, which, however, is not used in the derivation. As usual in each node we only show the additional data, if any. Thus the database in each node is given by the collection of the formulas from the root to that node. Here is an explanation of the steps: in step (2) $\alpha = \{(x_0, x_1)\}$; in step (3) $\alpha = \{(x_0, x_1), (x_1, x_2)\}$; in step (4) $\alpha = \{(x_0, x_1), (x_1, x_2), (x_2, x_3)\}$; since $A^{\mathbf{K}}_\alpha(x_2, x_3)$, by reduction w.r.t. $x_2 : b \Rightarrow c$ we get (5); since $A^{\mathbf{K}}_\alpha(x_1, x_2)$ and $A^{\mathbf{K}}_\alpha(x_2, x_3)$, by reduction w.r.t. $x_1 : (p \Rightarrow p) \Rightarrow a \Rightarrow b$ we get (6) and (8). the latter immediately succeeds as $x_3 : a \in \Delta$; from (6) we step to (7) which immediately succeeds.

EXAMPLE 43. In Figure 5 we show a derivation of

$$((((a \Rightarrow a) \Rightarrow p) \Rightarrow q) \Rightarrow p) \Rightarrow p$$

in P(**KBT**), we use restricted restart according to Proposition 41. In step (2), $\alpha = \{(x_0, x_1)\}$. Step (3) is obtained by reduction w.r.t. $x_1 : (((a \Rightarrow a) \Rightarrow p) \Rightarrow q) \Rightarrow p$, as $A^{\mathbf{KBT}}_\alpha(x_1, x_1)$. In step (4) $\alpha = \{(x_0, x_1), (x_1, x_2)\}$; step (5) is obtained by restart; step (6) by reduction w.r.t. $x_2 : (a \Rightarrow a) \Rightarrow p$, as $A^{\mathbf{KBT}}_\alpha(x_2, x_1)$; in step (7) $\alpha = \{(x_0, x_1), (x_1, x_2), (x_1, x_3)\}$ and the query immediately succeeds.

In order to prove soundness and completeness, we need to give a formal meaning to queries, i.e. to define when a query is valid. We first introduce a notion of *realization* of a database in a model to give a semantic meaning to databases.

DEFINITION 44 (Realization and validity). Let $A^{\mathbf{S}}$ be an accessibility predicate, given a database (Γ, α) and a Kripke model $M = (W, R, V)$,

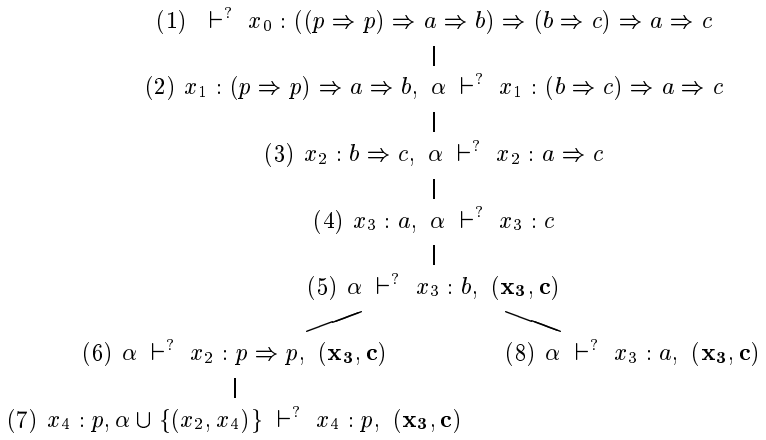


Figure 4. Derivation for Example 42.

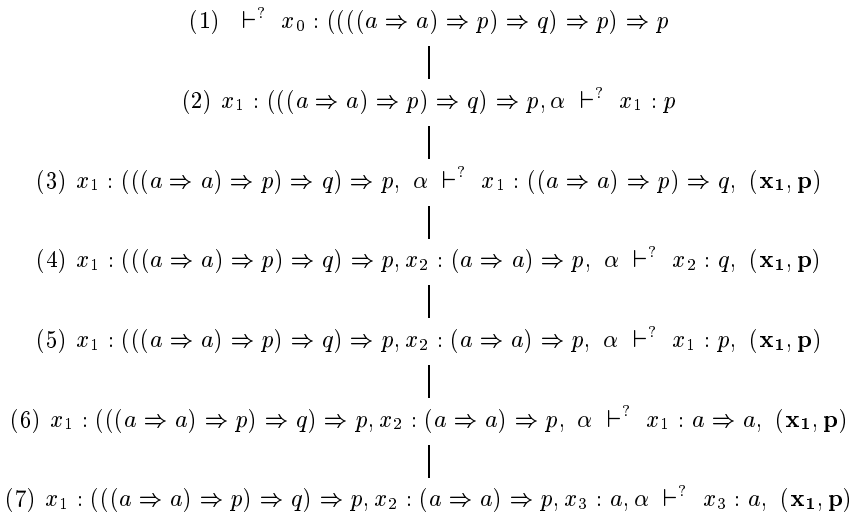


Figure 5. Derivation for Example 43.

a mapping $f : Lab(\Gamma) \to W$ is called a *realization* of (Γ, α) in M with respect to $A^{\mathbf{S}}$, if the following hold:

1. $A^{\mathbf{S}}_{\alpha}(x, y)$ implies $f(x)Rf(y)$;
2. if $x : A \in \Gamma$, then $M, f(x) \models A$.

We say that a query $Q = \Gamma\ , \alpha \ \vdash^{?} \ x : G, H$ is *valid* in $\mathbf{S}$ if for every $\mathbf{S}$-model M and every realization f of (Γ, α), we have either $M, f(x) \models G$, or for some $(y, r) \in H$, $M, f(y) \models r$.

The soundness of the proof procedure can be proved easily by induction on the length of the computation.

THEOREM 45 (Soundness). *Let* $Q = \ \Gamma, \alpha \ \vdash^{?} \ x : G, H$ *succeed in the proof system P(*$\mathbf{S}$*), then it is valid in* $\mathbf{S}$.

COROLLARY 46. *If* $x_0 : A \ \vdash^{?} \ x_0 : B, \emptyset$ *succeeds in P(*$\mathbf{S}$*) , then* $A \models_{\mathbf{S}} B$ *holds. In particular, if* $\vdash^{?} \ x_0 : A, \emptyset$ *succeeds in P(*$\mathbf{S}$*), then A is valid in* $\mathbf{S}$.

To prove completeness we proceed in a similar way to what we did for intuitionistic logic. First we show that cut is admissible. Then we prove completeness by a sort of canonical model construction. The cut rule states the following: let $x : A \in \Gamma$, then if (1) $\Gamma \vdash y : B$ and (2) $\Delta \vdash z : A$ succeed, we can 'replace' $x : A$ by Δ in Γ and get a successful query from (1). Since there are labels and accessibility predicates, we must be careful. There are two points to clarify. First, we need to define the involved notion of substitution. Furthermore the proof systems $\mathrm{P}(\mathbf{S})$ depend uniformly on predicate $A^{\mathbf{S}}$, and we expect that the admissibility of cut depends on the properties of predicate $A^{\mathbf{S}}$. It turns out that the admissibility of cut (stated in Theorem 48) holds for every proof system $\mathrm{P}(\mathbf{S})$, such that $A^{\mathbf{S}}$ satisfies the following conditions:

- (i) $A^{\mathbf{S}}$ is closed under substitution of labels;
- (ii) $A^{\mathbf{S}}_{\alpha}(x, y)$ implies $A^{\mathbf{S}}_{\alpha \cup \beta}(x, y)$;
- (iii) $A^{\mathbf{S}}_{\alpha}(u, v)$ implies $\forall x\ y\ (A^{\mathbf{S}}_{\alpha \cup \{(u,v)\}}(x, y) \leftrightarrow A^{\mathbf{S}}_{\alpha}(x, y))$.

These conditions ensure the following properties.

PROPOSITION 47.

(a) If $\Gamma, \alpha \ \vdash^{?} \ x : C, H$ *succeeds then also* $\Gamma[u/v], \alpha[u/v] \ \vdash^{?} \ x[u/v] : C, H[u/v]$ *succeeds.*

(b) If $A^{\mathbf{S}}_{\alpha}(x, y)$ *and* $\Gamma, \alpha \cup \{(x, y)\} \ \vdash^{?} \ u : G, H$ *succeed, then also* $\Gamma, \alpha \ \vdash^{?} \ u : G, H$ *succeeds.*

We say that two databases (Γ, α), (Δ, β) are *compatible for substitution*,[11] if

> for every $x \in Lab(\Gamma) \cap Lab(\Delta)$, for all formulas C, $x : C \in \Gamma \Leftrightarrow x : C \in \Delta$.

If (Γ, α) and (Δ, β) are compatible for substitution, $x : A \in \Gamma$, and $y \in Lab(\Delta)$, we denote by

$$(\Gamma, \alpha)[x : A/\Delta, \beta, y] = (\Gamma - \{x : A\} \cup \Delta, \alpha[x/y] \cup \beta).$$

the database which results by replacing $x : A$ in (Γ, α) by (Δ, β) at point y.

At this point we can state precisely the result about cut.

THEOREM 48 (Admissibility of cut). *Let predicate $A^{\mathbf{S}}$ satisfy the conditions (i), (ii), (iii) above. If the following queries succeed in the proof system P(**S**) :*

1. $\Gamma[x : A] \ \vdash^? \ u : B, H_1$
2. $\Delta, \beta \ \vdash^? \ y : A, H_2$.

and (Γ, α) and (Δ, β) are compatible for substitution, then also

3. $(\Gamma, \alpha)[x : A/\Delta, \beta, y] \ \vdash^? \ u[x/y] : B, H_1[x/y] \cup H_2$ *succeeds in P(**S**) .*

The proof proceeds similarly to the one of Theorem 10 and is given in [Gabbay and Olivetti, 2000]. From the theorem we immediately have the following two corollaries.

COROLLARY 49. *Under the same conditions as above, if $x : A \ \vdash^? \ x : B$ succeeds and $x : B \ \vdash^? \ x : C$ succeeds then also $x : A \ \vdash^? \ x : C$ succeeds.*

COROLLARY 50. *If $\mathbf{K} \in \mathbf{S} \subseteq \{\mathbf{K}, \mathbf{4}, \mathbf{5}, \mathbf{B}, \mathbf{T}\}$, then in the proof system P(**S**) cut is admissible.*

As we have said, we can prove the completeness by a sort of canonical model construction, which is less constructive of the one of Theorem 9. The following properties will be used in the completeness proof.

PROPOSITION 51.

- *(Identity) If $x : A \in \Gamma$, then $\Gamma, \alpha \ \vdash^? \ x : A, H$ succeeds.*
- *(Monotony) If $Q = \ \Gamma, \alpha \ \vdash^? \ x : C, H$ succeeds and $\Gamma \subseteq \Delta$, $\alpha \subseteq \beta$, $H \subseteq H'$, then also $\Delta, \beta \ \vdash^? \ x : C, H'$ succeeds.*

[11] This condition is not necessary if we allow the occurrence of several formulas with the same label in a database, as we will do in Section 4.3 when we add conjunction.

THEOREM 52 (Completeness). *Given a query* $Q = \Gamma, \alpha \ \vdash^? \ x : A, H$, *if* Q *is* **S***-valid then* Q *succeeds in the proof system P(***S***).*

Proof. By contraposition, we prove that if $Q = \Gamma, \alpha \ \vdash^? \ x : A, H$ does not succeed in one proof system P(**S**), then there is an **S**-model M and a realization f of (Γ, α), such that $M, f(x) \not\models A$ and for any $(y, r) \in H$, $M, f(y) \not\models r$.

We construct an **S**-model by extending the database, through the evaluation of all possible formulas at every world (each represented by one label) of the database. Since such evaluation may lead, for implication formulas, to create new worlds, we must carry on the evaluation process on these new worlds. Therefore, in the construction we consider an enumeration of pairs (x_i, A_i), where x_i is a label and A_i is a formula.

Assume $\Gamma, \alpha \ \vdash^? \ x : A, H$ fails in P(**S**). We let $\mathcal{A}$ be a denumerable alphabet of labels and $\mathcal{L}$ be the underlying propositional language. Let (x_i, A_i), for $i \in \omega$ be an enumeration of pairs of $\mathcal{A} \times \mathcal{L}$, starting with the pair (x, A) and containing infinitely many repetitions, that is

$$(x_0, A_0) = (x, A),$$
$$\forall y \in \mathcal{A}, \forall F \in \mathcal{L}, \forall n \, \exists m > n \ (y, F) = (x_m, A_m).$$

Given such enumeration we define (i) a sequence of databases (Γ_n, α_n), (ii) a sequence of histories H_n, (iii) a new enumeration of pairs (y_n, B_n), as follows:

- (step 0) Let $(\Gamma_0, \alpha_0) = (\Gamma, \alpha)$, $H_0 = H$, $(y_0, B_0) = (x, A)$.
- (step n+1) Given (y_n, B_n), if $y_n \in \mathit{Lab}(\Gamma_n)$ and $\Gamma_n, \alpha_n \ \vdash^? \ y_n : B_n, H_n$ fails then proceed according to (a) else to (b).

(a) if B_n if atomic, then we set

$$H_{n+1} = H_n \cup \{(y_n, B_n)\},$$
$$(\Gamma_{n+1}, \alpha_{n+1}) = (\Gamma_n, \alpha_n),$$
$$(y_{n+1}, B_{n+1}) = (x_{k+1}, A_{k+1}),$$
$$\text{where } k = \max_{t \leq n} \exists s_{\leq n} (y_s, B_s) = (x_t, A_t),$$

else let $B_n = C \Rightarrow D$, then we set

$$H_{n+1} = H_n,$$
$$(\Gamma_{n+1}, \alpha_{n+1}) = (\Gamma_n, \alpha_n) \oplus_{y_n} (x_m : C),$$
$$(y_{n+1}, B_{n+1}) = (x_m, D),$$
$$\text{where } x_m = \min\{x_t \in \mathcal{A} \mid x_t \notin \mathit{Lab}(\Gamma_n) \cup \mathit{Lab}(H_n)\}.$$

(b) We set

$$H_{n+1} = H_n,$$
$$(\Gamma_{n+1}, \alpha_{n+1}) = (\Gamma_n, \alpha_n),$$
$$(y_{n+1}, B_{n+1}) = (x_{k+1}, A_{k+1}),$$
$$\text{where } k = \max\{t \leq n \mid \exists s_{\leq n}(y_s, B_s) = (x_t, A_t)\},$$ ■

The proof of completeness is made of several lemmas.

LEMMA 53. $\forall k\ \exists n \geq k\ (x_k, A_k) = (y_n, B_n)$.

Proof. By induction on k. If $k = 0$, the claim holds by definition. Let $(x_k, A_k) = (y_n, B_n)$.

(i) if $y_n \notin Lab(\Gamma_n)$, or $\Gamma_n, \alpha_n \vdash^? y_n : B_n, H_n$ succeeds, or B_n is atomic, then $(x_{k+1}, A_{k+1}) = (y_{n+1}, B_{n+1})$.

(ii) Otherwise, let $B_n = C_1 \Rightarrow \ldots \Rightarrow C_t \Rightarrow r$, $(t > 0)$, then $(x_{k+1}, A_{k+1}) = (y_{n+t+1}, B_{n+t+1})$. ■

LEMMA 54. *For all $n \geq 0$, if $\Gamma_n, \alpha_n \vdash^? y_n : B_n, H_n$ fails, then:*

$$\forall m \geq n\ \Gamma_m, \alpha_m \vdash^? y_n : B_n, H_m \textit{ fails.}$$

Proof. By induction on $cp(B_n) = c$. if $c = 0$, that is B_n is an atom, say q, then we proceed by induction on $m \geq n + 1$.

- $(m = n + 1)$ we have $\Gamma_n, \alpha_n \vdash^? y_n : q, H_n$ fails, then also $\Gamma_n, \alpha_n \vdash^? y_n : q, H_n \cup \{(y_n, q)\}$ fails, whence, by construction,

  $$\Gamma_{n+1}, \alpha_{n+1} \vdash^? y_n : q, H_{n+1} \text{ fails.}$$

- $(m > n + 1)$ Suppose we have proved the claim up to $m \geq n + 1$, and suppose by way of contradiction that $\Gamma_m, \alpha_m \vdash^? y_n : q, H_m$ fails, but

  (i) $\Gamma_{m+1}, \alpha_{m+1} \vdash^? y_n : q, H_{m+1}$ succeeds.

  At step m, (y_m, B_m) is considered; it must be $y_m \in Lab(\Gamma_m)$ and

  (ii) $\Gamma_m, \alpha_m \vdash^? y_m : B_m, H_m$ fails.

  We have two cases, according to the form of B_m. If B_m is an atom r, as $(y_n, q) \in H_m$, from query (ii) by restart we can step to

  $$\Gamma_m, \alpha_m \vdash^? y_n : q, H_m \cup \{(y_m, r)\},$$

that is the same as $\Gamma_{m+1}, \alpha_{m+1} \vdash^? y_n : q, H_{m+1}$, which succeeds and we get a contradiction. If $B_m = C_1 \Rightarrow \ldots \Rightarrow C_k \Rightarrow r$, with $k > 0$, then from query (ii) we step in k steps to $\Gamma_{m+k}, \alpha_{m+k} \vdash^? y_{m+k} : r, H_{m+k}$, where $(\Gamma_{m+k}, \alpha_{m+k}) = (\Gamma_m, \alpha_m) \oplus_{y_m} (y_{m+1} : C_1) \oplus_{y_{m+1}} \ldots \oplus_{y_{m+k-1}} (y_{m+k} : C_k)$ and $H_{m+k} = H_m$; then, by restart, since $(y_n, q) \in H_{m+k}$, we step to

(iii) $\Gamma_{m+k}, \alpha_{m+k} \vdash^? y_n : q, H_{m+k} \cup \{(y_{m+k}, r)\}$.

Since query (i) succeeds, by monotony we have that also query (iii) succeeds, whence query (ii) succeeds, contradicting the hypothesis.

Let $cp(B_n) = c > 0$, that is $B_n = C \Rightarrow D$. By hypothesis $\Gamma_n, \alpha_n \vdash^? y_n : C \Rightarrow D, H_n$, fails. Then by construction and by the computation rules $\Gamma_{n+1}, \alpha_{n+1} \vdash^? y_{n+1} : D, H_{n+1}$, fails, and hence, by the induction hypothesis, $\forall m \geq n+1$,

$\Gamma_m, \alpha_m \vdash^? y_{n+1} : D, H_m$, fails.

Suppose by way of contradiction that for some $m \geq n+1$, $\Gamma_m, \alpha_m \vdash^? y_n : C \Rightarrow D, H_m$, succeeds. This implies that, for some $z \notin Lab(\Gamma_m) \cup Lab(H_m)$,

(1) $(\Gamma_m, \alpha_m) \oplus_{y_n} (z : C) \vdash^? z : D, H_m$, succeeds.

Since $y_{n+1} : C \in \Gamma_{n+1} \subseteq \Gamma_m$, $\alpha_{n+1} \subseteq \alpha_m$, $H_{n+1} \subseteq H_m$, by monotony, we get

(2) $\Gamma_m, \alpha_m \vdash^? y_{n+1} : C, H_m$, succeeds.

The databases involved in queries (1) and (2) are clearly compatible for substitution, hence by cut we obtain that $\Gamma_m, \alpha_m \vdash^? y_{n+1} : D, H_m$ succeeds, and we have a contradiction. ■

LEMMA 55.

(i) $\forall m, \Gamma_m, \alpha_m \vdash^? x : A, H_m$ *fails, whence*

(ii) $\forall m$, *if* $(y, r) \in H_m$, *then* $\Gamma_m, \alpha_m \vdash^? y : r, H_m$ *fails.*

Proof. Left to the reader. ■

LEMMA 56. *If* $B_n = C \Rightarrow D$ *and* $\Gamma_n, \alpha_n \vdash^? y_n : C \Rightarrow D, H_n$ *fails, then there is a* $y \in \mathcal{A}$, *such that for* $k \leq n$, $y \notin \mathrm{Lab}(\Gamma_k)$ *and* $\forall m > n$: *(i)* $(y_n, y) \in \alpha_m$, *(ii)* $\Gamma_m, \alpha_m \vdash^? y : C, H_m$ *succeeds, (iii)* $\Gamma_m, \alpha_m \vdash^? y : D, H_m$ *fails.*

Proof. By construction, we can take $y = y_{n+1}$, the new point created at step $n+1$, so that (i), (ii), (iii) hold for $m = n+1$. In particular

(*) $\Gamma_{n+1}, \alpha_{n+1} \ \vdash^? \ y_{n+1} : D, H_{n+1}$ fails.

Since the (Γ, α_m) are not decreasing (w.r.t. inclusion), we immediately have that (i) and (ii) also hold for every $m > n+1$. By construction, we know that $B_{n+1} = D$, whence by (*) and Lemma 54, (iii) also holds for every $m > n+1$. ■

Construction of the Canonical model
We define an **S**-model as follows $M = (W, R, V)$, such that

- $W = \bigcup_n Lab(\Gamma_n)$;
- $xRy \equiv \exists n A^{\mathbf{S}}_{\alpha_n}(x, y)$,
- $V(x) = \{q \mid \exists n \ x \in Lab(\Gamma_n) \ \wedge \ \Gamma_n, \alpha_n \ \vdash^? \ x : q, H_n \text{ succeeds}\}$.

LEMMA 57. *The relation R as defined above has the same properties of $A^{\mathbf{S}}$, e.g. if* **S**=**S4**, *that is $A^{\mathbf{S}}$ is transitive and reflexive, then so is R and the same happens in all other cases.*

Proof. Left to the reader. ■

LEMMA 58. *for all $x \in W$ and formulas B,*

$$M, x \models B \ \Leftrightarrow \ \exists n \ x \in \mathrm{Lab}(\Gamma_n) \wedge \Gamma_n, \alpha_n \ \vdash^? \ x : B, H_n \ \textit{succeeds.}$$

Proof. We prove both directions by mutual induction on $cp(B)$. If B is an atom then the claim holds by definition. Thus, assume $B = C \Rightarrow D$.

($\Leftarrow$) Suppose for some m $\Gamma_m, \alpha_m \ \vdash^? \ x : C \Rightarrow D, H_m$ succeeds. Let xRy and $M, y \models C$, for some y. By definition of R, we have that for some n_1, $A^{\mathbf{S}}_{\alpha_{n_1}}(x, y)$ holds. Moreover, by the induction hypothesis, for some n_2, $\Gamma_{n_2}, \alpha_{n_2} \ \vdash^? \ y : C, H_{n_2}$ succeeds. Let $k = \max\{n_1, n_2, m\}$, then we have

1. $\Gamma_k, \alpha_k \ \vdash^? \ x : C \Rightarrow D, H_k$ succeeds,
2. $\Gamma_k, \alpha_k \ \vdash^? \ y : C, H_k$ succeeds,
3. $A^{\mathbf{S}}_{\alpha_k}(x, y)$.

So that from 1. we also have:

1′. $(\Gamma_k, \alpha_k) \oplus_x (z : C) \ \vdash^? \ z : D, H_k$ succeeds, (with $z \notin Lab(\Gamma_k) \cup Lab(H_k)$).

We can cut 1′. and 2., and obtain:

$$\Gamma_k, \alpha_k \cup \{(x, y)\} \ \vdash^? \ y : D, H_k \text{ succeeds.}$$

Hence, by 3. and Proposition 47(b) we get $\Gamma_k, \alpha_k \vdash^? y : D, H_k$ succeeds, and by the induction hypothesis, $M, y \models D$,

($\Rightarrow$) Suppose by way of contradiction that $M, x \models C \Rightarrow D$, but for all n if $x \in Lab(\Gamma_n)$, then $\Gamma_n, \alpha_n \vdash^? x : C \Rightarrow D, H_n$ fails. Let $x \in Lab(\Gamma_n)$, then there are $m \geq k > n$, such that $(x, C \Rightarrow D) = (x_k, A_k) = (y_m, B_m)$ is considered at step $m + 1$, so that we have:

$$\Gamma_m, \alpha_m \vdash^? y_m : C \Rightarrow D, H_m \text{ fails.}$$

By Lemma 56, there is a $y \in \mathcal{A}$, such that (a) for $t \leq m$, $y \notin Lab(\Gamma_t)$ and (b): $\forall m' > m$ (i) $(y_n, y) \in \alpha_{m'}$, (ii) $\Gamma_{m'}, \alpha_{m'} \vdash^? y : C, H_{m'}$ succeeds, (iii) $\Gamma_{m'}, \alpha_{m'} \vdash^? y : D, H_{m'}$ fails.
By (i) we have xRy holds, by (ii) and the induction hypothesis, we have $M, y \models C$. By (a) and (iii), we get: $\forall n$ if $y \in Lab(\Gamma_n)$, then $\Gamma_n, \alpha_n \vdash^? y : D, H_n$ fails. Hence, by the induction hypothesis, we have $M, y \not\models D$, and we get a contradiction. ■

Proof of The Completeness Theorem, 52. We are now able to conclude the proof of the completeness theorem. Let $f(z) = z$, for every $z \in Lab(\Gamma_0)$, where $(\Gamma_0, \alpha_0) = (\Gamma, \alpha)$ is the original database. It is easy to see that f is a realization of (Γ, α) in M: if $A^{\mathbf{S}}_{\alpha}(u, v)$ then $A^{\mathbf{S}}_{\alpha_0}(u, v)$, hence $f(u)Rf(v)$.
If $u : C \in \Gamma = \Gamma_0$, then by identity and the previous lemma we have $M, f(u) \models C$. On the other hand, by Lemma 55, and the previous lemma we have $M, f(x) \not\models A$ and $M, f(y) \not\models r$ for every $(y, r) \in H$. This concludes the proof. ■

By the previous theorem we immediately have the corollary.

COROLLARY 59. *If $A \models_{\mathbf{S}} B$ holds, then $A \vdash^? x_0 : B, \emptyset$, succeeds in P(**S**). In particular, if A is valid in the modal system **S**, then $\vdash^? x_0 : A, \emptyset$, succeeds in P(**S**) .*

4.2 Simplification for specific systems

In this section we show that for most of the modal logics we have considered, the use of labelled databases is not necessary and we can simplify either the structure of databases, or the deduction rules.

If we want to check the validity of a formula A, we evaluate A from a trivial database $\vdash^? x_0 : A, \emptyset$. Restricting our attention to computations from trivial databases, we observe that we can only generate databases which have the form of trees.

DEFINITION 60. A database (Δ, α) is called a *tree-database* if the set of links α forms a tree.

Let (Δ, α) be a tree database and $x \in Lab(\Delta)$, we define the subdatabase $Path(\Delta, \alpha, x)$ as the *list* of labelled formulas lying on the path from the root

of α, say x_0, up to x, that is: $Path(\Delta, \alpha, x) = (\Delta', \alpha')$, where:

$$\begin{aligned} \alpha' &= \{(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n) \mid x_n = x \\ &\quad \text{and for } i = 1, \ldots, n, (x_{i-1}, x_i) \in \alpha\} \\ \Delta' &= \{y : A \in \Delta \mid y \in Lab(\alpha')\}. \end{aligned}$$

PROPOSITION 61. *If a query Q occurs in any derivation from a trivial database, then $Q = \Delta, \alpha \vdash^? z : B, H$, where (Δ, α) is a tree-database.*

From now on we restrict our consideration to tree-databases.

Simplification for **K**, **K4**, **S4**, **KT**: *Databases as Lists*

For systems **K**, **K4**, **S4**, **KT** the proof procedure can be simplified in the sense that: (i) the databases are lists of formulas, (ii) the restart rule is not needed. The key fact is expressed by the following theorem.

THEOREM 62. *If $\Delta, \alpha \vdash^? x : A, \emptyset$ succeeds, then $Path(\Delta, \alpha, x) \vdash^? x : A, \emptyset$ succeeds without using restart.*

Intuitively, only the formulas laying on the path from the root to $x : A$ can be used in a proof of $\Delta, \alpha \vdash^? x : A, \emptyset$. The reason why restart is not needed is related: a restart step, say a restart from $x : q$, is useful only if we can take advantage of formulas at worlds created after the first call of $x : q$ by means of the evaluation of an implicational goal. But these new worlds (being new) do not lay on the path from the root to $x : q$, thus they can be ignored and so can the restart step.

By virtue of this theorem we can reformulate the proof system for logics from **K** to **S4** as follows. A database is simply a list of formulas $A_1, \ldots, A_n$, which stands for the labelled database $(\{x_1 : A_1, \ldots, x_n : A_n\}, \alpha)$, where $\alpha = \{(x_1, x_2), \ldots (x_{n-1}, x_n)\}$. A query has the form:

$$A_1, \ldots, A_n \vdash^? B$$

which represents $\{x_1 : A_1, \ldots, x_n : A_n\}, \alpha \vdash^? x_n : B$. The history has been omitted since restart is not needed. Letting $\Delta = A_1, \ldots, A_n$, we reformulate the predicates $A^{\mathbf{S}}$ as relations between formulas within a database $A^{\mathbf{S}}(\Delta, A_i, A_j)$, in particular we can define:

$$\begin{aligned} A^{\mathbf{K}}(\Delta, A_i, A_j) &\equiv i + 1 = j \\ A^{\mathbf{KT}}(\Delta, A_i, A_j) &\equiv i = j \vee i + 1 = j \\ A^{\mathbf{K4}}(\Delta, A_i, A_j) &\equiv i < j \\ A^{\mathbf{S4}}(\Delta, A_i, A_j) &\equiv i \leq j \end{aligned}$$

The rules become:

- (success) $\Delta \vdash^? q$ succeeds if $\Delta = A_1, \ldots, A_n$, and $A_n = q$;

- (implication) from $\Delta \vdash^? A \Rightarrow B$ step to $\Delta, A \vdash^? B$;
- (reduction) from $\Delta \vdash^? q$ step to $\Delta_i \vdash^? D_i$, for $i = 1, \ldots, k$, if there is a formula $A_j = D_1 \Rightarrow \ldots \Rightarrow D_k \Rightarrow q \in \Delta$, and there are integers $j = j_0 \leq j_1 \leq \ldots \leq j_k = n$, such that

$$i = 1, \ldots, k,\ A^{\mathbf{S}}(\Delta, A_{j_{i-1}}, A_{j_i}) \text{ holds and } \Delta_i = A_1, \ldots, A_{j_i}.$$

EXAMPLE 63. We show that $((b \Rightarrow a) \Rightarrow b) \Rightarrow c \Rightarrow (b \Rightarrow a) \Rightarrow a$ is a theorem of **S4**.

$$\begin{array}{rcl}
 & \vdash^? & ((b \Rightarrow a) \Rightarrow b) \Rightarrow c \Rightarrow (b \Rightarrow a) \Rightarrow a \\
(b \Rightarrow a) \Rightarrow b & \vdash^? & c \Rightarrow (b \Rightarrow a) \Rightarrow a \\
(b \Rightarrow a) \Rightarrow b, c & \vdash^? & (b \Rightarrow a) \Rightarrow a \\
(b \Rightarrow a) \Rightarrow b, c, b \Rightarrow a & \vdash^? & a \quad \text{reduction w.r.t. } b \Rightarrow a \ (1) \\
(b \Rightarrow a) \Rightarrow b, c, b \Rightarrow a & \vdash^? & b \quad \text{reduction w.r.t. } (b \Rightarrow a) \Rightarrow b \ (2) \\
(b \Rightarrow a) \Rightarrow b, c, b \Rightarrow a & \vdash^? & b \Rightarrow a \\
(b \Rightarrow a) \Rightarrow b, c, b \Rightarrow a, b & \vdash^? & a \quad \text{reduction w.r.t. } b \Rightarrow a \\
(b \Rightarrow a) \Rightarrow b, c, b \Rightarrow a, b & \vdash^? & b.
\end{array}$$

This formula fails in both **KT** and **K4**, and therefore also fails in **K**: reduction at step (1) is allowed in **KT** but not in **K4**; on the contrary, reduction at step (2) is allowed in **K4** but not in **KT**.

Simplification for **K5**, **K45**, **S5**: *Databases as Clusters*

We can also give an unlabelled formulation of logics **K5**, **K45**, **S5**. The simplification is allowed by the fact that we can define explicitly the accessibility relation.

PROPOSITION 64. *Let* $Q = \Delta, \alpha \vdash^? x : G, H$ *be any query which occurs in a P(***K5***) deduction from a trivial database* $x_0 : A \vdash^? x_0 : B, H_0$. *Let* $R_\alpha^{\mathbf{K5}}(x, y)$ *be defined as follows:*

$$\begin{array}{rcl}
R_\alpha^{\mathbf{K5}}(x, y) & \equiv & (x = x_0 \wedge (x_0, y) \in \alpha) \\
 & & \vee\ (\{x, y\} \subseteq \mathrm{Lab}(\alpha) \wedge x \neq x_0 \wedge y \neq x_0).
\end{array}$$

Then we have $R_\alpha^{\mathbf{K5}}(x, y) \equiv A_\alpha^{\mathbf{K5}}(x, y)$.

COROLLARY 65. *Under the same conditions as the last proposition, we have:*

$$R_\alpha^{\mathbf{K5}}(x_0, x) \text{ and } R_\alpha^{\mathbf{K5}}(x_0, y) \text{ implies } x = y.$$

PROPOSITION 66. *Let* $Q = \Delta, \alpha \vdash^? x : G, H$ *be any query which occurs in a P(***K45***) deduction from a trivial database* $x_0 : A \vdash^? x_0 : B, H_0$. *Let* $R_\alpha^{\mathbf{K45}}(x, y)$ *be defined as follows:*

$$R_\alpha^{\mathbf{K45}}(x,y) \equiv \{x,y\} \subseteq \mathrm{Lab}(\alpha) \wedge y \neq x_0.$$

Then we have $R_\alpha^{\mathbf{K45}}(x,y) \equiv A_\alpha^{\mathbf{K45}}(x,y)$.

PROPOSITION 67. *Let* $Q = \Delta, \alpha \vdash^? x : G, H$ *be any query which occurs in a P(***S5***) deduction from a trivial database* $x_0 : A \vdash^? x_0 : B, H_0$. *Let* $R_\alpha^{\mathbf{S5}}(x,y)$ *be defined as follows:*

$$R_\alpha^{\mathbf{S5}}(x,y) \equiv \{x,y\} \subseteq \mathrm{Lab}(\alpha).$$

Then we have $R_\alpha^{\mathbf{S5}}(x,y) \equiv A_\alpha^{\mathbf{S5}}(x,y)$.

From the previous propositions we can reformulate the proof systems for **K5**, **K45** and **S5** without making use of labels. For **K5** the picture is as follows: either a database contains just one point x_0, or there is an initial point x_0 which is connected to another point x_1, and any point excluding x_0 is connected to any other. In the case of **K45**, x_0 is connected also to any point other than itself. Thus, in order to get a concrete structure without labels we must keep distinct the initial world from all the others, and we must indicate what is the current world, that is the world in which the goal formula is evaluated. In case of **K5** we must also identify the (only) world to which the initial world is connected. We are thus led to consider the following structure.

A non-empty database has the form:

$$\Delta = B_0 \mid \mid \quad \text{or } \Delta = B_0 \mid B_1, \ldots, B_n \mid B_i, \text{ where } 1 \leq i \leq n,$$

and $B_0, B_1, \ldots, B_n$ are formulas. We also define

$$Actual(\Delta) = \begin{cases} B_0 \text{ if } \Delta = B_0 \mid \mid, \\ B_i \text{ if } \Delta = B_0 \mid B_1, \ldots, B_n \mid B_i. \end{cases}$$

This rather odd structure is forced by the fact that in **K5** and **K45** we have reflexivity in all worlds, except in the initial one and therefore, in contrast to all other systems, we have considered so far, the success of

$$\vdash^? x_0 : A \Rightarrow B, \text{ which means that } A \Rightarrow B \text{ is valid,}$$

does not imply the success of

$$x_0 : A \vdash^? x_0 : B, \text{ which means that } A \to B \text{ is valid (material implication).}^{12}$$

The addition operation is defined as follows:

$$\Delta \oplus A = \begin{cases} B_0 \mid B_1, \ldots, B_n, A \mid A & \text{if } \Delta = B_0 \mid B_1, \ldots, B_n \mid B_i \\ B_0 \mid A \mid A & \text{if } \Delta = B_0 \mid \mid \\ \top \mid A \mid A & \text{if } \Delta = \emptyset \end{cases}$$

[12] In these two systems the validity of $\Box C$ does not imply the validity of C, as it holds for all the other systems considered in this section.

A query has the form

$$\Delta \ \vdash^? \ G, H, \text{ where } H = \{(A_1, q_1), \ldots, (A_k, q_k)\}, \text{ with } A_j \in \Delta.$$

DEFINITION 68 (Deduction Rules for **K5** and **K45**).
Given $\Delta = \ B_0 \mid B_1, \ldots, B_n \mid B$, let

$$\begin{aligned} A^{\mathbf{K5}}(\Delta, X, Y) \quad &\equiv \quad (X = B_0 \ \wedge \ Y = B_1) \\ &\quad\ \vee \ (X = B_i \wedge Y = B_j \ \wedge \ i, j > 0) \text{ and} \\ A^{\mathbf{K45}}(\Delta, X, Y) \quad &\equiv \quad (X = B_i \wedge Y = B_j \text{ with } j > 0) \end{aligned}$$

- (success) $\Delta \ \vdash^? \ q, H$ succeeds if $\mathit{Actual}(\Delta) = q$.
- (implication) From $\Delta \ \vdash^? \ A \Rightarrow B, H$ step to $\Delta \oplus A \ \vdash^? \ B, H$.
- (reduction) if $\Delta = \ B_0 \mid B_1, \ldots, B_n \mid B$ and $C = D_1 \Rightarrow \ldots \Rightarrow D_k \Rightarrow q \in \Delta$, from $\Delta \ \vdash^? \ G, H$ step to

  $$B_0 \mid B_1, \ldots, B_n \mid C_i \ \vdash^? \ D_i, H \cup \{(B, q)\} \text{ for } i = 1, \ldots, k,$$

  for some $C_0, \ldots, C_k \ \in \ \Delta$, such that $C_0 \ = \ C$, $C_k \ = \ B_n$, and $A^{\mathbf{K5}}(\Delta, C_{i-1}, C_i)$ (respectively $A^{\mathbf{K45}}(\Delta, C_{i-1}, C_i)$) holds.
- (restart) If $\Delta = \ B_0 \mid B_1, \ldots, B_n \mid B_i$ and $(B_j, r) \in H$, with $j > 0$, then from $\Delta \ \vdash^? \ q, H$, step to

  $$B_0 \mid B_1, \ldots, B_n \mid B_j \ \vdash^? \ r, H \cup \{(B_i, q)\},$$

According to the above discussion, we observe that the check of the validity of $\models A \Rightarrow B$, corresponds to the query

$$\emptyset \ \vdash^? \ A \Rightarrow B, \emptyset,$$

which (by the implication rule) is reduced to the query

$$\top \mid A \mid A \ \vdash^? \ B, \emptyset.$$

This is different from checking the validity of $A \rightarrow B$ ($\rightarrow$ is the material implication), which corresponds to the query

$$A \mid \ \mid \vdash^? \ B, \emptyset.$$

The success of the former query does not imply the success of the latter. For instance in **K5**,

$$\not\models (\top \Rightarrow p) \rightarrow p \text{ and indeed } \top \Rightarrow p \mid \ \mid \vdash^? \ p, \emptyset \text{ fails.}$$

On the other hand we have

$\models (\top \Rightarrow p) \Rightarrow p$ and indeed $\top \mid \top \Rightarrow p \mid \top \Rightarrow p \ \vdash^? \ p, \emptyset$ succeeds.

The reformulation of the proof system for **S5** is similar, but simpler. In the case of **S5**, there is no need to keep the first formula/world apart from the others. Thus, we may simply define a non-empty database as a pair $\Delta = (S, A)$, where S is a set of formulas and $A \in S$. If $\Delta = (S, A)$, we let

$$Actual(\Delta) = A \text{ and } \Delta \oplus B = (S \cup \{B\}, B).$$

For $\Delta = \emptyset$, we define $\emptyset \oplus A = (\{A\}, A)$. With these definitions the rules are similar to those of **K5** and **K45**, with the following simplifications:

- (reduction) if $\Delta = (S, B)$ and $C = D_1 \Rightarrow \ldots \Rightarrow D_k \Rightarrow q \in \Delta$, then from $\Delta \ \vdash^? \ G, H$ step to

  $(S, C_i) \ \vdash^? \ D_i, H \cup \{(B, q)\}$, where for $i = 1, \ldots, k$ $C_i \in \Delta$ and $C_k = B$.

- (restart) If $\Delta = (S, B)$ and $(C, r) \in H$, then from $(S, B) \ \vdash^? \ q, H$, step to

  $(S, C) \ \vdash^? \ r, H \cup \{(B, q)\}$,

EXAMPLE 69. In Figure 6 we show a derivation of the following formula in **S5**

$$((a \Rightarrow b) \Rightarrow c) \Rightarrow (a \Rightarrow d \Rightarrow c) \Rightarrow (d \Rightarrow c).$$

In the derivation we make use of restricted restart, according to Proposition 41. A brief explanation of the derivation: step (5) is obtained by reduction w.r.t. $(a \Rightarrow b) \Rightarrow c$, step (7) by restart, steps (8) and (9) by reduction w.r.t. $a \Rightarrow d \Rightarrow c$, and they both succeed immediately.

4.3 Extending the language

In this section we extend the proof procedures to broader fragments. We first consider a simple extension allowing conjunction. To handle conjunction in the labelled formulation, we simply drop the condition that a label x may be attached to only one formula, thus formulas with the same label can be thought as logically conjuncted. In the unlabelled formulation, for those systems enjoying such a formulation, the general principle is to deal with *sets* of formulas, instead of *single formulas*. A database will be a structured collection of *sets* of formulas, rather than a collection of formulas. The structure is always the same, but the constituents are now sets. Thus, in the cases of **K**, **KT**, **K4** and **S4**, databases will be lists of *sets* of formulas,

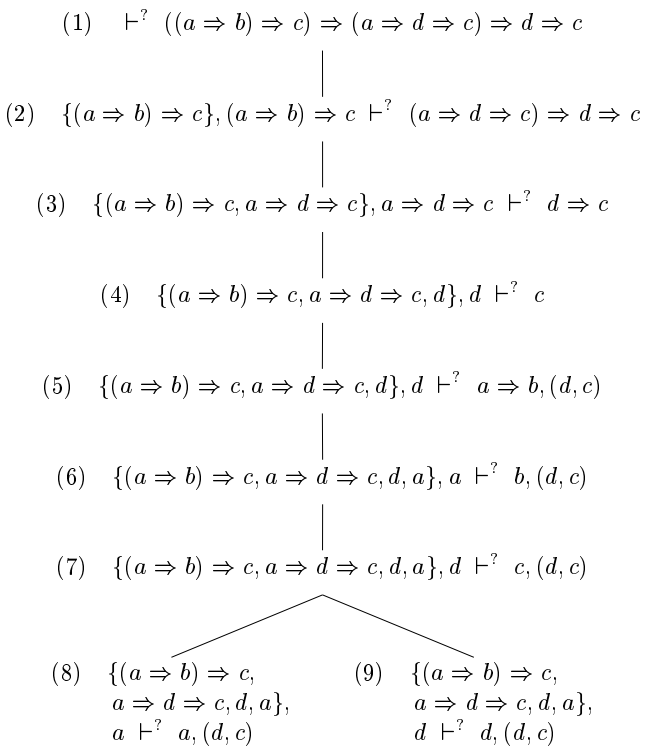


Figure 6. Derivation for Example 69.

whereas in the cases of **K5**, **K45** and **S5**, they will be clusters of *sets* of formulas.

A conjunction of formulas is interpreted as a set, so that queries may contain *sets* of goal formulas.

A formula A of language $\mathcal{L}(\wedge, \Rightarrow)$ is in *normal form* if it is an atom or has form:

$$\bigwedge_i [S_1^i \Rightarrow \ldots \Rightarrow S_{n_i}^i \Rightarrow q_i]$$

where S_j^i are conjunctions of formulas in *normal form*. In all modal logics considered in this section (formulated $\mathcal{L}(\wedge, \Rightarrow)$) it holds that every formula has an equivalent one in normal form.

We simplify the notation for NF formulas and replace conjunctions with sets. For example the NF of $(b \Rightarrow (c \wedge d)) \Rightarrow (e \wedge f)) \wedge ((g \wedge h) \Rightarrow (k \wedge u))$ is the set containing the following formulas:

$$\{b \Rightarrow c, b \Rightarrow d\} \Rightarrow e,\ \{b \Rightarrow c, b \Rightarrow d\} \Rightarrow f,\ \{g, h\} \Rightarrow k,\ \{g, h\} \Rightarrow u.$$

For the deduction procedures all we have to do is to handle sets of formulas. We define, for $x \in Lab(\Gamma) \cup Lab(H)$, $y \notin Lab(\Gamma) \cup Lab(H)$ and finite set of formulas $S = \{D_1, \ldots, D_t\}$,

$$(\Gamma, \alpha) \oplus_x y : S = (\Delta \cup \{y : D_1, \ldots, y : D_t\}, \alpha \cup \{(x, y)\}),$$

then we change the *(implication) rule* in the obvious way:

$$\text{from } \Delta, \alpha \vdash^? x : S \Rightarrow B, H,$$

step to

$$(\Delta, \alpha) \oplus_x y : S \vdash^? y : B, H,$$

where S is a set of formulas in NF and $y \notin Lab(\Gamma)$, and we add a rule for proving sets of formulas:

$$\text{from } (\Delta, \alpha) \vdash^? x : \{B_1, \ldots B_k\}, H$$

step to

$$(\Delta, \alpha) \vdash^? x : B_i, H \text{ for } i = 1, \ldots, k.$$

Regarding the simplified formulations without labels, the structural restrictions in the rules (reduction and success) are applied to the sets of formulas, which are now the constituents of databases, considered as units; the history H, when needed, becomes a set of pairs (S_i, A_i), where S_i is a set and A_i is a formula. The property of restricted restart still holds for this formulation.

We can extend further extend the $\mathcal{L}(\Rightarrow, \wedge)$-fragment in two directions. In one direction, we can define a modal analogue of Harrop formulas for intuitionistic logic that we have introduced in Section 3.7. This extension is relevant for logic programming applications [Giordano *et al.*, 1992; Giordano and Martelli, 1994]. In the other direction, we can define a proof system for the whole propositional modal language via a translation into an implicative normal form.

Modal Harrop formulas

We can define a modal analogue of Harrop formulas by allowing disjunction of goals and *local clauses* of the form

$$G \to q,$$

where $\to$ denotes ordinary (material) implication. We call them 'local', since $x : G \to q$ can be used only in world x to reduce the goal $x : q$ and it is not usable/visible in any other world. It is 'private' to x, whereas the 'global' clause $G \Rightarrow q$ can be used to reduce q in any world y accessible from x. It is not a case that modalities have been used to implement visibility rules and structuring mechanisms in logic programming [Giordano *et al.*, 1992;

Giordano and Martelli, 1994]. In order to define a modal Harrop fragment we distinguish D-formulas, which are the constituents of databases, and G-formulas which can occur as goals. The former are further distinct in *modal* D-formulas (MD) and *local* D-formulas (LD).

$$LD := G \to q,$$
$$MD := \top \mid q \mid G \Rightarrow MD,$$
$$D := LD \mid MD,$$
$$CD := D \mid CD \wedge CD;$$
$$G := \top \mid q \mid G \wedge G \mid G \vee G \mid CD \Rightarrow G.$$

We also use $\Box G$ and $\Box D$ as syntactic sugar for $\top \Rightarrow G$ and $\top \Rightarrow D$. Notice that atoms are both LD- and MD-formulas (as $\top \to q \equiv q$); moreover, any non-atomic MD-formula can be written as $G_1 \Rightarrow \ldots \Rightarrow G_k \Rightarrow q$. Finally, CD formulas are just conjunction of D-formulas.

For D- and G-formulas as defined above we can easily extend the proof procedure. We give it in the most general formulation for labelled databases. It is clear that one can derive an unlabelled formulation for systems which allow it, as explained in the previous section. In the labelled formulation, queries have the form

$$\Delta, \alpha \ \vdash^? \ x : G, H$$

where Δ is a set of D-formulas, G is a G-formula, and $H = \{(x_1, G_1), \ldots, (x_k, G_k)\}$, where G_i are G-formulas. The **additional** rules are:

- (true) $\Delta, \alpha \ \vdash^? \ x : \top, H$ immediately succeeds.
- (local-reduction) From $\Delta, \alpha \ \vdash^? \ x : q, H$ step to

  $$\Delta, \alpha \ \vdash^? \ x : G, H \cup \{(x : q)\}$$

  if $x : G \to q \in \Delta$.
- (and) From $\Delta, \alpha \ \vdash^? \ x : G_1 \wedge G_2, H$ step to

  $$\Delta, \alpha \ \vdash^? \ x : G_1, H \text{ and } \Delta, \alpha \ \vdash^? \ x : G_2, H.$$
- (or) From $\Delta, \alpha \ \vdash^? \ x : G_1 \vee G_2, H$ step to

  $$\Delta, \alpha \ \vdash^? \ x : G_1, H \cup \{(x, G_2)\} \text{ or to } \Delta, \alpha \ \vdash^? \ x : G_2, H \cup \{(x, G_1)\}.$$

EXAMPLE 70. Let Δ be the following database

$$x_0 : [(\Box p \Rightarrow s) \wedge b] \to q,$$
$$x_0 : ([(p \Rightarrow q) \wedge \Box a] \Rightarrow r) \to q,$$
$$x_0 : a \to b.$$

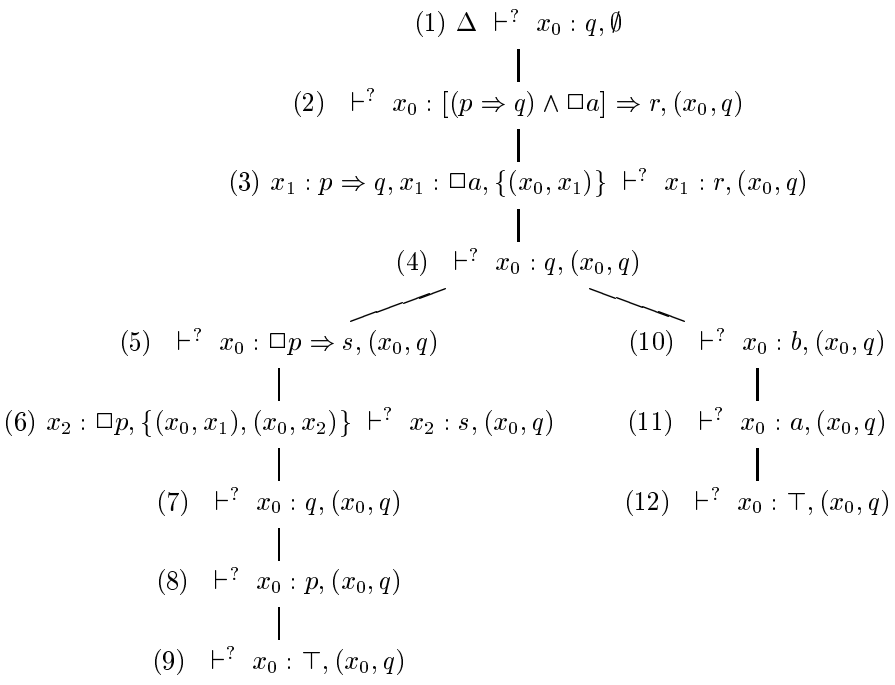


Figure 7. Derivation for Example 70.

We show that $\Delta, \emptyset \vdash^? x_0 : q, \emptyset$ succeeds in the proof system for **KB** and this shows that the formula

$\bigwedge \Delta \to q$ is valid in **KB**.

A derivation is shown in Figure 7. The property of restricted restart still holds for this fragment, thus we do not need to record the entire history, but only the first pair (x_0, q). At each step we only show the additional data introduced in that step. A quick explanation of the steps: step (2) is obtained by local reduction w.r.t. $x_0 : ([(p \Rightarrow q) \wedge \Box a] \Rightarrow r) \to q$, step (4) by restart, steps (5) and (10) by local reduction w.r.t. $x_0 : [(\Box p \Rightarrow s) \wedge b] \to q$, step (7) by restart, step (8) by reduction w.r.t. $x_1 : p \Rightarrow q$ since letting $\alpha = \{(x_0, x_1), (x_0, x_2)\}$ $A_\alpha^{\mathbf{KB}}(x_1, x_0)$ holds; step (9) is obtained by reduction w.r.t. $x_2 : \Box p$ $(= \top \Rightarrow p)$ since $R_\alpha^{\mathbf{KB}}(x_2, x_0)$ holds, step (11) by local reduction w.r.t. $x_0 : a \to b$, step (12) by reduction w.r.t. $x_1 : \Box a$ $(= \top \Rightarrow a)$ since $R_\alpha^{\mathbf{KB}}(x_1, x_0)$ holds.

The soundness and completeness results can be extended to this fragment.

THEOREM 71. *$\Delta \vdash^? G, H$ succeeds in P(**S**) if and only if it is valid.*

Extension to the whole propositional language

We can easily extend the procedure to the whole propositional modal language: we just consider the computation procedure for classical logic and we combine it with the modal procedure for $\mathcal{L}(\Rightarrow, \wedge)$. To minimize the work, we can introduce a simple normal form on the set of connectives $(\Rightarrow, \rightarrow, \wedge, \top, \bot)$. It is obvious that this set forms a complete base for modal logic. The normal form is an immediate extension of the normal form for $\Rightarrow, \wedge$.

PROPOSITION 72. *Every modal formula over the language* $(\rightarrow, \neg, \Diamond, \Box)$ *is equivalent to a set (conjunction) of NF-formulas of the form*

$$S_0 \rightarrow (S_1 \Rightarrow (S_2 \Rightarrow \ldots \Rightarrow (S_n \Rightarrow q) \ldots)$$

where q *is an atom,* $\top$*, or* $\bot$*,* $n \geq 0$*, and each* S_i *is a conjunction (set) of NF-formulas.*

As usual we omit parentheses, so that the above will be written as

$$S_0 \rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots \Rightarrow S_n \Rightarrow q.$$

In practice we will replace the conjunction by the set notation as we wish. When we need it, we distinguish two types of NF-formulas, (i) those with non-empty S_0, which are written as above, and (ii) those with empty S_0, which are simplified to $S_1 \Rightarrow S_2 \Rightarrow \ldots \Rightarrow S_n \Rightarrow q$. For a quick case analysis, we can also say that type (i) formulas have the form $S \rightarrow D$, and type (ii) have the form $S \Rightarrow D$, where D is always of type (ii).

For instance, the NF-form of $p \rightarrow \Diamond r$ is

$$(p \wedge (r \Rightarrow \bot)) \rightarrow \bot, \text{ or equivalently } \{p, r \Rightarrow \bot\} \rightarrow \bot.$$

This formula has the structure $S_0 \rightarrow q$, where $S_0 = \{p, r \Rightarrow \bot\}$ and $q = \bot$. The NF-form of $\Box(\Diamond a \rightarrow \Diamond(b \wedge c))$ is given by $((a \Rightarrow \bot) \rightarrow \bot) \Rightarrow ((b \wedge c) \Rightarrow \bot) \Rightarrow \bot$. We give below the rules for queries of the form

$$\Gamma, \alpha \ \vdash^? \ x : G, H,$$

where Γ is a labelled set of NF-formulas, G is a NF-formula, α is a set of links (as usual), H is a set of pairs $\{(x_1, q_1), \ldots (x_k, q_k)\}$, where q_i is an atom.

DEFINITION 73 (Deduction rules for whole modal logics). For each modal system $\mathbf{S}$, the corresponding proof system, denoted by $\mathrm{P}(\mathbf{S})$, comprises the following rules, parametrized to predicates $A^{\mathbf{S}}$.

- (success) $\Delta, \alpha \ \vdash^? \ x : q, H$ immediately succeeds if q is an atom and $x : q \in \Delta$.
- (strict implication) From $\Delta, \alpha \ \vdash^? \ x : S \Rightarrow D, H$ step to

$$(\Delta, \alpha) \oplus_x (y : S) \ \vdash^? \ y : D, H,$$

where $y \notin Lab(\Delta) \cup Lab(H)$.

- (implication) From $\Delta, \alpha \ \vdash^? \ x : S \to D, H$ step to

$$\Delta \cup \{x : A \mid A \in S\}, \alpha \ \vdash^? \ x : D, H.$$

- (reduction) If $y : C \in \Delta$, with $C = S_0 \to S_1 \Rightarrow S_2 \Rightarrow \ldots \Rightarrow S_k \Rightarrow q$, with q atomic, then from

$$\Delta, \alpha \ \vdash^? \ x : q, H$$

step to

$$\begin{array}{l} \Delta, \alpha \ \vdash^? \ u_0 : S_0, H' \\ \Delta, \alpha \ \vdash^? \ u_1 : S_1, H' \\ \vdots \\ \Delta, \alpha \ \vdash^? \ u_k : S_k, H' \end{array}$$

where $H' = H$ if $q = \bot$, and $H' = H \cup \{(x, q)\}$ otherwise, for some $u_0, \ldots, u_k \in Lab(\alpha)$, such that $u_0 = y$, $u_k = x$, and

$$A^{\mathbf{S}}_\alpha(u_i, u_{i+1}) \text{ holds, for } i = 0, \ldots, k-1.$$

- (restart) If $(y, r) \in H$, then, from $\Delta, \alpha \ \vdash^? \ x : q, H$, with q atomic, step to

$$\Delta, \alpha \ \vdash^? \ y : r, H \cup \{(x, q)\}.$$

- (falsity) From $\Delta, \alpha \ \vdash^? \ x : q, H$, if $y \in Lab(\Gamma)$ step to

$$\Delta, \alpha \ \vdash^? \ y : \bot, H \cup \{(x : q)\}.$$

- (conjunction) From $(\Delta, \alpha) \ \vdash^? \ x : \{B_1, \ldots B_k\}, H$ step to

$$(\Delta, \alpha) \ \vdash^? \ x : B_i, H \text{ for } i = 1, \ldots, k.$$

If the number of subgoals is 0, i.e. $k = 0$, the above reduction rule becomes the rule for *local clauses* of the previous section. On the other hand if $S_0 = \emptyset$, then the query with goal S_0 is omitted and we have the rule of Section 4.1.

The proof procedure is sound and complete, as asserted in the next theorem, and the completeness proof is just a minor extension of the one of Theorem 52.

THEOREM 74. *$\Gamma, \gamma \ \vdash^? \ x : G, H$ succeeds if and only if it is valid.*

EXAMPLE 75. In **K5** we have $\Diamond p \to \Box\Diamond p$. This is translated as $((p \Rightarrow \bot) \to \bot) \to ((p \Rightarrow \bot) \Rightarrow \bot)$. Below we show a derivation. Some

explanation and remarks: at step (1) we can only apply the rule for falsity, or reduction w.r.t. $y : p \Rightarrow \bot$, since $A_\alpha^{\mathbf{K5}}(x_0, y)$ implies $A_\alpha^{\mathbf{K5}}(y, y)$. We apply the rule for falsity. The reduction at step (2) is legitimate as $A_\alpha^{\mathbf{K5}}(x_0, y)$ and $A_\alpha^{\mathbf{K5}}(x_0, z)$ implies $A_\alpha^{\mathbf{K5}}(y.z)$.

$$
\begin{array}{lll}
 & \vdash^? & x_0 : ((p \Rightarrow \bot) \to \bot) \to ((p \Rightarrow \bot) \Rightarrow \bot \\
x_0 : (p \Rightarrow \bot) \to \bot & \vdash^? & x_0 : (p \Rightarrow \bot) \Rightarrow \bot \\
y : p \Rightarrow \bot,\ \alpha = \{(x_0, y)\} & \vdash^? & y : \bot \qquad (1) \\
 & \vdash^? & x_0 : \bot \quad \text{rule for } \bot \\
 & \vdash^? & x_0 : p \Rightarrow \bot \\
z : p,\ \alpha = \{((x_0, y), (x_0, z)\} & \vdash^? & z : \bot \qquad (2) \\
 & \vdash^? & z : p \\
 & & \text{success}
\end{array}
$$

This proof procedure is actually a minor extension of the one based on strict-implication/conjunction. The rule for falsity may be source of non-determinism, as it can be applied to any label y. Further investigation should clarify to what extent this rule is needed and if it is possible to restrict its applications to special cases. Another point which deserve investigation is *termination*. The proof procedure we have described may not terminate. The two standard techniques to ensure termination, loop-checking and diminishing-resources, could be possibly applied in this context. Again further investigation is needed to clarify this point, taking into account the kwnon results (see [Viganò, 1999; Heudering *et al.*, 1996]).

4.4 Some history

Many authors have developed analytic proof methods for modal logics, (see the fundamental book by Fitting [1983], and Goré [1999] for a recent and comprehensive survey).

The use of goal-directed methods in modal logic has not been fully explored. The most relevant work in this area is the one by Giordano, Martelli and colleagues [Giordano *et al.*, 1992; Giordano and Martelli, 1994; Baldoni *et al.*, 1998] who have developed goal-directed methods for fragments of first-order (multi-)modal logics. Their work is motivated by several purposes: introducing scoping constructs (such as blocks and modules) in logic programming, representing epistemic and inheritance reasoning. In particular in [Giordano and Martelli, 1994] a family of first-order logic programming languages is defined, based on the modal logic **S4** with the aim of representing a variety of scoping mechanisms. If we restrict our consideration to the propositional level, their languages are strongly related to the one defined in Section 4.3 in the case the underlying logic is **S4**. The largest (propositional) fragment of **S4** they consider, called L_4, is very close

to the one defined in the previous Section for modal-Harrop formulas, although neither one of the two is contained in the other. The proof procedure they give for L_4 (at the propositional level) is essentially the same as the unlabelled version of P(**S4**) for modal Harrop formulas.

Abadi and Manna [1989] have defined an extension of PROLOG, called *TEMPLOG* based on a fragment of first-order temporal logic. Their language contains the modalities $\Diamond$, $\Box$, and the temporal operator $\bigcirc$ (next). They introduce a notion of temporal Horn clause whose constituents are atoms B possibly prefixed by an arbitrary sequence of next, i.e. $B \equiv \bigcirc^k A$ (with $k \geq 0$). The modality $\Box$ is allowed in front of clauses (permanent clauses) and clause-heads, whereas the modality $\Diamond$ is allowed in front of goals. The restricted format of the rules allows one to define an efficient and simple goal-directed procedure without the need of any syntactic structuring or labelling. An alternative, although related, extension based on temporal logic has been studied in [Gabbay, 1987].

Farinãs in [1986] describes MOLOG a (multi)-modal extension of PROLOG. His proposal is more a general framework than a specific language, in the sense that the language can support different modalities governed by different logics. The underlying idea is to extend classical resolution by special rules of the following pattern: let B, B' be modal atoms (i.e. atomic formulas possibly prefixed by modal operators), then if $G \rightarrow B$ is a clause and $B' \wedge C_1 \wedge \ldots \wedge C_k$ is the current goal, and

$$(*) \models_{\mathbf{S}} B \equiv B' \text{ holds}$$

then the goal can be reduced to $G \wedge C_1 \wedge \ldots \wedge C_k$. It is clear that the effectiveness of the method depends on how difficult it is to check (*); in case of conventional logic programming the (*) test is reduced to unification. The proposed framework is exemplified in [Farinãs, 1986] by defining a multimodal language based on **S5** with necessity operators such as $Knows(a)$. In this case one can define a simple matching predicate for the test in (*), and hence an effective resolution rule.

In general, we can distinguish two paradigms in proof systems for modal logics: on the one hand we have *implicit* calculi in which each proof configuration contains a set of formulas implicitly representing a single possible world; the modal rules encodes the shifts of world by manipulating sets of formulas and formulas therein. On the other hand we have explicit methods in which the possible world structure is explicitly represented using labels and relations among them; the rules can create new worlds, or move formulas around them. In between there are 'intermediate' proof methods which add some semantic structure to flat sequents, but they do not explicitly represent a Kripke model [Masini, 1992; Wansing, 1994; Goré, 1999].

The use of labels to represent worlds for modal logics is rather old and goes back to Kripke himself. In the seminal work [Fitting, 1983] formulas

are labelled by strings of atomic labels (world prefixes), which represent paths of accessible worlds. The rules for modalities are the same for every system: for instance if a branch contains $\sigma : \Box A$, and σ' is accessible from σ, then one can add $\sigma' : A$ to the same branch. For each system, there are some specific accessibility conditions on prefixes which constraint the propagation of modal formulas. This approach has been recently improved by Massacci [1994]. Basin, Mattews and Viganò have developed a proof-theory for modal logics making use of labels and an explicit accessibility relation [Basin *et al.*, 1997a; Basin *et al.*, 1999; Viganò, 1999]. A related approach was presented in [Gabbay, 1996] and [Russo, 1996]. These authors have developed both sequent and natural deduction systems for several modal logics which are completely uniform.

If we forget the goal-directed feature the proof methods presented in this section clearly belongs to the 'explicit'-calculi tradition in their labelled version, and to the 'intermediate'-calculi tradition calculi in their unlabelled version. The sequence of (sets of) formulas represent a sequence of possible worlds. It is not a case that the unlabelled version of **K** is strongly related to the two-dimensional sequent calculus by Masini [1992].

5 SUBSTRUCTURAL LOGICS

5.1 Introduction

In this section we consider substructural logics. The denomination *substructural logics* comes from sequent calculi terminology. In sequent calculi, there are rules which introduce logical operators and rules which modify the structure of sequents. The latter are called the *structural rules*. In case of classical and intuitionistic logic these rules are *contraction*, *weakening* and *exchange*. Substructural logics restrict or allow a finer control on structural rules. More generally, substructural logics restricts *the use* of formulas in a deduction. The restrictions may require either that every formula of the database must be used, or that it cannot be used more than once, or that it must be used according to a given ordering of database formulas.

We present the systems of substructural logics, restricted to their implicational fragment, by means of a Hilbert axiomatization and a possible-world semantics.

DEFINITION 76 (Axiomatization of implication). We consider the following list of axioms:

(id) $A \to A$;

(h1) $(B \to C) \to (A \to B) \to A \to C$;

(h2) $(A \to B) \to (B \to C) \to A \to C$;

(h3) $(A \to A \to B) \to A \to B$;

(h4) $(A \to B) \to ((A \to B) \to C) \to C$;

(h5) $A \to (A \to B) \to B$;

(h6) $A \to B \to B$.

Together with the following rules:

$$\frac{A \to B \qquad A}{B}\,(MP)$$

$$\frac{A \to B}{(B \to C) \to A \to C}\,(\textit{Suff}).$$

Each system is axiomatized by taking the closure under modus ponens (MP) and under substitution of the combinations of axioms/rules of Table 3.

Table 3. Axioms for substructural implication.

Logic	Axioms
FL	(id), (h1), (*Suff*)
T-W	(id), (h1), (h2)
T	(id), (h1), (h2), (h3)
E-W	(id), (h1), (h2), (h4)
E	(id), (h1), (h2), (h3), (h4)
L	(id), (h1), (h2), (h5)
R	(id), (h1), (h2), (h3), (h5)
BCK	(id), (h1), (h2), (h5), (h6)
I	(id), (h1), (h2), (h3), (h5), (h6)

In the above axiomatization, we have not worried about the minimality and independence of the group of axioms for each system. For some systems the corresponding list of axioms given above is redundant, but it quickly shows some inclusion relations among the systems. We just remark that in presence of (h4), (h2) can be obtained by (h1). Moreover, (h4) is a weakening of (h5). The rule of (*Suff*) is clearly obtainable from (h2) and (MP). To have a complete picture we have included also intuitionistic logic **I**, although the axiomatization above is highly redundant (see Section 3.1).

We give a brief explanation of the names of the systems and how they are known in the literature.[13] **R** is the most important system of *relevant*

[13] The names **R**, **E**, **T**, **BCK**, etc. in this section refer mainly to *the implicational fragment* of the logical systems known in the literature [Anderson and Belnap, 1975]

logic and it is axiomatized by dropping the irrelevant axiom (h6) from the axiomatization of intuitionistic implication.

The system **E** combines relevance and necessity. The implication of **E** can be read at the same time as relevant implication and strict implication. Moreover, we can define

$$\Box A =_{def} (A \to A) \to A,$$

and **E** interprets $\Box$ the same as **S4**. **E** is axiomatized by restricting the exchange axiom (h5) to implicational formulas (the axiom (h4)).

The weaker **T** stems from a concern about the use of the two hypotheses in an inference by Modus Ponens: the restriction is that the minor A must not be derived 'before' the ticket $A \to B$. This is clarified by the Fitch-style natural deduction of **T**, for which we refer to [Anderson and Belnap, 1975].

BCK is the system which result from intuitionistic implicational logic by dropping contraction. **L** rejects both weakening and contraction and it is the implicational fragment of linear logic [Girard, 1987] (also commonly known as **BCI** logic).

We will also consider contractionless versions of **E** and **T**, , namely **E-W**and **T-W** respectively.

The weakest system we consider is **FL**,[14] which is related to the *right implicational* fragment of Lambek calculus. This system rejects all substructural rules.

We will mainly concentrate on the implicational fragment of the systems mentioned. In Section 5.3, we will extend the proof systems to a fragment similar to Harrop-formulas. For the fragment considered, all the logics studied in this section are subsystems of intuitionistic logic. However, this is no longer true for the fragment comprising an involutive negation, which can be added (and has been added) to each system. In this section we do not consider the treatment of negation. We refer the reader to [Anderson and Belnap, 1975; Anderson *et al.*, 1992] for an extensive discussion.

In Figure 5.1 we show the inclusion relation of the systems we consider in this section.

We give a corresponding semantics for this set of systems. The semantics we refer is a simplification of the one proposed in [Fine, 1974], [Anderson *et al.*, 1992] and elaborated more recently by Došen [1988; 1989].[15]

with the corresponding names. The implicational fragments are usually denoted with the subscript $\to$. Thus, what we call **R** is denoted in the literature by $\mathbf{R}_{\to}$ and so forth; since we are mainly concerned with the implicational systems we have preferred to minimize the notation, stating explicitly when we make exception to this convention.

[14] The denomination of the system is taken from [Ono, 1998; Ono, 1993].

[15] Dealing only with the implicational fragment, we have simplified Fine semantics: we do not have *prime* or maximal elements.

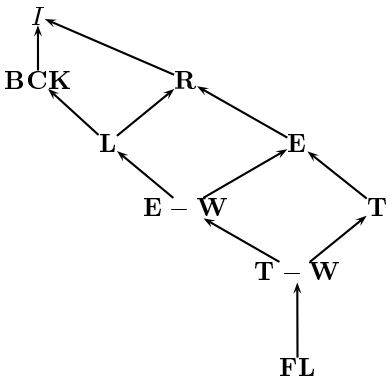


Figure 8. Lattice of Substructural Logics.

DEFINITION 77. Let us fix a language $\mathcal{L}$, a Fine **S**-structure[16] M is a tuple of the form:

$$M = (W, \leq, \circ, 0, V),$$

where W is a non empty set, $\circ$ is a binary operation on W, $0 \in W$, $\leq$ is a partial order relation on W, V is a function of type $W \to Pow(Var)$. In all structures the following properties are assumed to hold:

$0 \circ a = a$,
$a \leq b$ implies $a \circ c \leq b \circ c$,
$a \leq b$ implies $V(a) \subseteq V(b)$.

For each system **S**, a **S**-structure satisfies a subset of the following conditions, as specified in Table 4

(a1) $a \circ (b \circ c) \leq (a \circ b) \circ c$;

(a2) $a \circ (b \circ c) \leq (b \circ a) \circ c$;

(a3) $(a \circ b) \circ b \leq a \circ b$;

(a4) $a \circ 0 \leq a$;

(a5) $a \circ b \leq b \circ a$;

(a6) $0 \leq a$.

Truth conditions for $a \in W$, we define

- $M, a \models p$ if $p \in V(a)$;

[16] We just write **S**-structure if there is no risk of confusion.

- $M, a \models A \to B$ if

$$\forall b \in W\,(M, b \models A \;\Rightarrow\; M, a \circ b \models B).$$

We say that A is *valid* in M (denoted by $M \models A$) if $M, 0 \models A$. We say that A is **S**- valid, denoted by
$\models^{Fine}_{\mathbf{S}} A$ if A is valid in every **S**-structure.

Table 4. Algebraic conditions of Fine Semantics.

Logic	(a1)	(a2)	(a3)	(a4)	(a5)	(a6)
FL	*					
T-W	*	*				
T	*	*	*			
E-W	*	*		*		
E	*	*	*	*		
L	*	*		*	*	
R	*	*	*	*	*	
BCK	*	*		*	*	*
I	*	*	*	*	*	*

We have included again intuitionistic logic **I** in Table 4 to show its proper place within this framework. Again this list of semantical conditions is deliberately redundant in order to show quickly the inclusion relation among the systems. The axiomatization given above is sound and complete with respect to this semantics. In particular each axiom (h**i**) corresponds to the semantical condition (a**i**).

THEOREM 78 (Anderson *et al.*, 1992, Fine, 1974, Došen, 1989). $\models_{\mathbf{S}} A$ *if and only if* A *is derivable in the corresponding axiom system of Definition 76.*

We assume that $\circ$ associates to the left, so we write

$$a \circ b \circ c = (a \circ b) \circ c.$$

5.2 Proof systems

We develop proof methods for the implicational logics: **R**, **BCK**, **E**, **T**, **E-W**, **T-W**, **FL**. As we have seen in the section about modal logics, we can control the use of formulas by labelling data and putting constraints on the labels. In this specific context by labelling data, we are able to record whether they have been used or not and to express the additional conditions needed for each specific system. Formulas are labelled with atomic labels x, y, z. Intuitively these labels can be read as representing at the same time *resources* and *positions* within a database.

DEFINITION 79. Let us fix a denumerable alphabet $\mathcal{A} = \{x_1, \ldots, x_i, \ldots\}$ of labels. We assume that labels are totally ordered as shown in the enumeration, $\mathbf{v_0}$ is the first label. A *database* is a finite set of labelled formulas $\Delta = \{x_1 : A_1, \ldots, x_n : A_n\}$. We assume that

if $x : A \in \Delta$ and $x : B \in \Delta$, then $A = B$.[17]

We use the notation $Lab(E)$ for the set of labels occurring in an expression E, and we finally assume that $\mathbf{v_0} \notin Lab(\Delta)$. Label $\mathbf{v_0}$ will be used for queries from the empty database.

DEFINITION 80. A query Q is an expression of the form:

$$\Delta, \delta \ \vdash^? \ x : G$$

where Δ is a database, δ is a finite set of labels not containing $\mathbf{v_0}$; moreover if $x \neq \mathbf{v_0}$ then $x \in Lab(\Delta)$, and G is a formula.

A query from the empty database has the form:

$$\vdash^? \ \mathbf{v_0} : G.$$

Let $\max(\delta)$ denote the maximum label in δ according to the enumeration of the labels. By convention, we stipulate that if $\delta = \emptyset$, then $\max(\delta) = \mathbf{v_0}$. The set of labels δ may be thought as denoting the set of resources that are available to prove the goal. Label x in front of the goal has a double role as a 'position' in the database from which the goal is asked, and as available resource.

The rules for success and reduction are parametrized to some conditions $Succ^{\mathbf{S}}$ and $Red^{\mathbf{S}}$ that will be defined below.

- (success) $\Delta, \delta \ \vdash^? \ x : q$; succeeds if $x : q \in \Delta$ and $Succ^{\mathbf{S}}(\delta, x)$.
- (implication) from $\Delta, \delta \ \vdash^? \ x : C \to G$ step to

  $$\Delta \cup \{y : C\}, \delta \cup \{y\} \ \vdash^? \ y : G,$$

  where $y > \max(Lab(\Delta))$, (whence $y \notin Lab(\Delta)$);
- (reduction) from

  $$\Delta, \delta \ \vdash^? \ x : q,$$

  if there is some $z : C \in \Delta$, with $C = \ A_1 \to \ldots \to A_k \to q$, and there are δ_i, and x_i for $i = 0, \ldots, k$ such that:

  1. $\delta_0 = \{z\}$, $x_0 = z$,

[17] This restriction will be lifted in Section 5.3 where conjunction is introduced in the language.

2. $\bigcup_{i=0}^{k} \delta_i = \delta$,
3. $Red^{\mathbf{S}}(\delta_0, \ldots, \delta_k, x_0, \ldots, x_k; x)$

then for $i = 1, \ldots k$, we step to

$$\Delta, \delta_i, \vdash^{?} x_i : A_i.$$

The conditions for success are either (s1) or (s2) according to each system:

(s1) $Succ^{\mathbf{S}}(\delta, x) \equiv x \in \delta$,

(s2) $Succ^{\mathbf{S}}(\delta, x) \equiv \delta = \{x\}$.

The conditions $Red^{\mathbf{S}}$ are obtained as combination of the following clauses:

(r0) $x_k = x$;

(r1) for $i, j = 0, \ldots, k$, $\delta_i \cap \delta_j = \emptyset$;

(r2) for $i = 1, \ldots, k$, $x_{i-1} \leq x_i$ and $\max(\delta_i) \leq x_i$;

(r3) for $i = 1, \ldots, k$, $x_{i-1} \leq x_i$ and $\max(\delta_i) = x_i$;

(r4) for $i = 1, \ldots, k$, $x_{i-1} < x_i$, $\max(\delta_{i-1}) = x_{i-1} < \min(\delta_i)$ and $\max(\delta_k) = x_k$.

The conditions $Red^{\mathbf{S}}$ are then defined according to Table 5.

Table 5. Restrictions on reduction and success.

Condition	(r0)	(r1)	(r2)	(r3)	(r4)	(Success)
FL	*				*	(s2)
T-W	*	*		*		(s2)
T	*			*		(s2)
E-W	*	*	*			(s2)
E	*		*			(s2)
L		*				(s2)
R						(s2)
BCK		*				(s1)

Notice that

$(r4) \Rightarrow (r3) \Rightarrow (r2)$, and
$(r4) \Rightarrow (r1)$.

We give a quick explanation of the conditions $Succ^{\mathbf{S}}$ and $Red^{\mathbf{S}}$. We recall that the component δ represents the set of available resources which must/can be used in a derivation.

For the success rule, in all cases but **BCK**, we have that we can succeed if $x : q$ is in the database, x is the only resource left, and q is asked from position x; in the case of **BCK** x must be among the available resources, but we do not require that x is the only one left.

The conditions for the reduction rule can be explained intuitively as follows: resources δ are split in several δ_i, for $i = 1, \ldots, k$ and each part δ_i must be used in a derivation of a subgoal A_i.

In the case of logics *without contraction* we cannot use a resource twice, therefore by restriction (r1), the δ_is must be disjointed and z, the label of the formula we are using in the reduction step, is no longer available.

Restriction (r2) imposes that successive subgoals are to be proved from successive positions in the database: only positions $y \geq x$ are 'accessible' from x; moreover each x_i must be accessible from resources in δ_i. Notice that the last subgoal A_k must be proved from x, the position from which the atomic goal q is asked.

Restriction (r3) is similar to (r4), but it further requires that the position x_i is among the available resources δ_i.

Restriction (r4) forces the goal A_i to be proved by using successive disjointed *segments* δ_i of δ. Moreover, z which labels the formula used in the reduction step must be the first (or least) resource among the available ones.

It is not difficult to see that intuitionistic (implicational) logic is obtained by considering success condition (s1) and no other constraint. More interestingly, we can see that **S4**-strict implication is given by considering success condition (s1) and restrictions (r0) and (r2) on reduction. We leave the reader to check that the above formulation coincides with the database-as-list formulation of **S4** we have seen in the previous section. We can therefore consider **S4** as a substructural logic obtained by imposing a restriction on the weakening and the exchange rules. On the other hand, the relation between **S4** and **E** should be apparent: the only difference is the condition on the success rule which controls the weakening restriction.

We can prove that each system is complete with respect to the its axiomatization by a syntactic proof. To this aim, we need to show that every axiom/rule is derivable, and the sets of derivable formulas is closed under substitution and Modus Ponens. The former property is proved by induction on the length of a derivation. The latter property is as usual a straightforward consequence of cut admissibility. This property is proved similarly to Theorem 10, although the details of the proof are more complex, because of the various restrictions on the reduction rule (see [Gabbay and Olivetti, 2000] pages 181–191, Theorem 5.19).

PROPOSITION 81 (Substitution). *If* $Q = \Gamma, \gamma \vdash^? x : A$ *succeeds, then also* $Q' = \Gamma[q/B], \gamma \vdash^? x : A[q/B]$ *succeeds.*

PROPOSITION 82 (Modus Ponens). *If* $\vdash^? \mathbf{v_0} : A \to B$ *and* $\vdash^? \mathbf{v_0} : A$ *succeed then also* $\vdash^? \mathbf{v_0} : B$ *succeeds.*

PROPOSITION 83 (Identity). *If* $x : A \in \Gamma$ *and* $Succ^{\mathbf{S}}(\gamma, x)$ *then* $\Gamma, \gamma \vdash^? x : A$ *succeeds.*

THEOREM 84 (Completeness). *For every system* **S**, *if* A *is a theorem of* **S**, *then* $\vdash \mathbf{v_0} : A$ *succeeds in the corresponding proof system for* **S**.

Proof. By Propositions 81, 82, we only need to show a derivation of an arbitrary atomic instance of each axiom in the relative proof system. In the case of reduction, the condition $\gamma = \bigcup \gamma_i$, will not be explicitly shown, as its truth will be apparent by the choice of γ_i. We assume that the truth of the condition for the success rule is evident and we do not mention it. At each step we only show the current goal, the available resources and the *new* data introduced in the database, if any. Moreover, we justify the queries obtained by a reduction step by writing the relation $Red^{\mathbf{S}}(\gamma_0, \ldots, \gamma_n, x_0, \ldots, x_n; x)$ (for suitable γ_i, x_i) under them; the database formula used in the reduction step is identified by γ_0.

(id) In all systems:

$$\vdash^? \mathbf{v_0} : a \to a$$

we step to

$$u : a, \{u\} \vdash^? u : a,$$

which immediately succeeds in all systems.

(h1) In all systems:

$$\vdash^? \mathbf{v_0} : (b \to c) \to (a \to b) \to a \to c.$$

three steps of the implication rule leads to:

$$\begin{aligned} x_1 : b \to c,\ x_2 : a \to b,\ x_3 : a,\ \{x_1, x_2, x_3\} &\vdash^? x_3 : c \\ \{x_2, x_3\} &\vdash^? x_3 : b \\ Red^{\mathbf{S}}(\{x_1\}, \{x_2, x_3\}, x_1, x_3; x_3) & \\ \{x_3\} &\vdash^? x_3 : a \\ Red^{\mathbf{S}}(\{x_2\}, \{x_3\}, x_2, x_3; x_3). & \end{aligned}$$

(h2) In all systems, but **FL**:

$$\vdash^? \ \mathbf{v_0} : (a \to b) \to (b \to c) \to a \to c.$$

three steps of the implication rule leads to:

$$\begin{aligned} x_1 : a \to b,\ x_2 : b \to c,\ x_3 : a,\ \{x_1, x_2, x_3\} \quad &\vdash^? \quad x_3 : c \\ (*)\ \{x_1, x_3\} \quad &\vdash^? \quad x_3 : b, \\ Red^{\mathbf{S}}(\{x_2\}, \{x_1, x_3\}, x_2, x_3; x_3) & \\ \{x_3\} \quad &\vdash^? \quad x_3 : a, \\ Red^{\mathbf{S}}(\{x_1\}, \{x_3\}, x_1, x_3; x_3). & \end{aligned}$$

the step (*) is allowed in all systems, but those with (r4), namely **FL**.

(h3) In all systems, but those with (r1) or (r4):

$$\vdash^? \ \mathbf{v_0} : (a \to a \to b) \to a \to b.$$

Two steps of the implication rule leads to:

$$x_1 : a \to a \to b,\ x_2 : a,\ \{x_1, x_2\} \ \vdash^? \ x_2 : b.$$

By reduction we step to:

$$\{x_2\} \ \vdash^? \ x_2 : a \quad \text{and} \quad \{x_2\} \ \vdash^? \ x_2 : a$$

since $Red^{\mathbf{S}}(\{x_1\}, \{x_2\}, \{x_2\}, x_1, x_2, x_2; x_2)$ holds in all systems without (r1) and (r4).

(h4) In all systems, but those with (r3) or (r4):

$$\vdash^? \ \mathbf{v_0} : (a \to b) \to ((a \to b) \to c) \to c.$$

two steps of the implication rule leads to:

$$\begin{aligned} x_1 : a \to b,\ x_2 : (a \to b) \to c,\ \{x_1, x_2\} \quad &\vdash^? \quad x_2 : c \\ (*)\ \{x_1\} \quad &\vdash^? \quad x_2 : a \to b \\ Red^{\mathbf{S}}(\{x_2\}, \{x_1\}, x_2, x_2; x_2) & \\ x_3 : a,\ \{x_1, x_3\} \quad &\vdash^? \quad x_3 : b \\ \{x_3\} \quad &\vdash^? \quad x_3 : a \\ Red^{\mathbf{S}}(\{x_1\}, \{x_3\}, x_1, x_3; x_3) & \end{aligned}$$

The step (*) is allowed by (r2), but not by (r3) or (r4) since $\max(\{x_1\}) = x_1 < x_2$.

(h5) In **L**,**R**,**BCK**:

$$\vdash^? \ \mathbf{v_0} : a \to (a \to b) \to b.$$

two steps of implication rule leads to:

$$\begin{array}{rll} x_1 : a,\ x_2 : a \to b,\ \{x_1, x_2\} & \vdash^? & x_2 : b \\ \{x_1\} & \vdash^? & x_1 : a \end{array}$$

$Red^{\mathbf{S}}(\{x_2\}, \{x_1\}, x_2, x_1; x_2)$.

(h6) In **BCK** we have:

$$\vdash^? \ \mathbf{v_0} : a \to b \to b.$$

two steps by implication rule leads to:

$$x_1 : a,\ x_2 : b,\ \{x_1, x_2\} \ \vdash^? \ x_2 : b$$

which succeeds by the success condition of **BCK**. This formula does not succeed in any other system.

(*Suff*) We prove the admissibility of (*Suff*) rule in **FL**. Let $\vdash^? \ \mathbf{v_0} : A \to B$ succeed. Then for any formula C, we have to show that

$$\vdash^? \ \mathbf{v_0} : (B \to C) \to A \to C \text{ succeeds.}$$

Let $C = C_1 \to \ldots \to C_n \to q$. Starting from

$$\vdash^? \ \mathbf{v_0} : (B \to C_1 \to \ldots \to C_n \to q) \to A \to C_1 \to \ldots \to C_n \to q$$

by the implication rule, we step to $\Delta, \{x_1, \ldots, x_{n+2}\} \vdash x_{n+2} : q$, where

$$\begin{array}{rcl} \Delta & = & \{x_1 : B \to C_1 \to \ldots \to C_n \to q,\ x_2 : A, \\ & & x_3 : C_1, \ldots, x_{n+2} : C_n\}. \end{array}$$

From the above query we step by reduction to:

$$\begin{array}{l} Q' = \ \Delta, \{x_2\} \ \vdash^? \ x_2 : B \text{ and} \\ Q_i = \ \Delta, \{x_{i+2}\} \ \vdash^? \ x_{i+2} : C_i \text{ for } i = 1, \ldots, n. \end{array}$$

since the conditions for reduction are satisfied. By hypothesis, $\vdash^? \ \mathbf{v_0} : A \to B$ succeeds, which implies, by the implicational rule, that $x_2 : A, \{x_2\} \ \vdash^? \ x_2 : B$ succeeds, but then Q' succeeds by monotony. Queries Q_i succeed by Proposition 83. ■

We can prove the soundness semantically. To this purpose we need to interpret databases and queries in the semantics. As usual, we introduce the notion of realization of a database and then of validity of a query.

DEFINITION 85 (Realization). Given a database Γ, and a set of labels γ, an **S**-*realization* of (Γ, γ) in an **S**-structure $M = (W, \circ, \leq, 0, V)$, is a mapping $\rho : \mathcal{A} \to W$ such that:

1. $\rho(\mathbf{v_0}) = 0$;
2. if $y : B \in \Gamma$ then $M, \rho(y) \models B$.

In order to define the notion of validity of a query, we need to introduce some further notation. Given an **S**-realization ρ, γ and x, we define

$$\rho(\gamma) = 0 \text{ if } \gamma = \emptyset,$$
$$\rho(\gamma) = \rho(x_1) \circ \ldots \circ \rho(x_n) \text{ if } \gamma = \{x_1, \ldots, x_n\}, \text{ where } x_1 < \ldots < x_n$$
$$\rho(< \gamma, x >) = \rho(\gamma) \text{ if } x \in \gamma,$$
$$\rho(< \gamma, x >) = \rho(\gamma) \circ 0 \text{ if } x \notin \gamma.$$

DEFINITION 86 (Valid query). Let $Q = \Gamma, \gamma \vdash^? x : A$, we say that Q is **S**-*valid* if for every **S**-structure M, for every realization ρ of Γ in M, we have

$$M, \rho(< \gamma, x >) \models A.$$

According to the definition above, the **S**-validity of the query $\vdash^? \mathbf{v_0} : A$ means that the formula A is **S**-valid (i.e. $\models^{Fine}_{\mathbf{S}} A$).

THEOREM 87. *If $Q = \Gamma, \gamma \vdash^? x : A$ succeeds in the proof system for* **S** *then Q is* **S**-*valid. In particular, if $\mathbf{v_0} : A$ succeeds in the proof system for* **S**, *then $\models^{Fine}_{\mathbf{S}} A$.*

The proof can be done by induction on the length of derivations, by suitably relating the constraints of the reduction rule to the algebraic semantic conditions.

5.3 Extending the language

In this section we show how we can extend the language by some other connectives. We allow extensional conjunction ($\wedge$), disjunction ($\vee$), and intensional conjunction or *tensor* ($\otimes$). The distinction between $\wedge$ and $\otimes$ is typical of substructural logics and it comes from the rejection of some structural rule: $\wedge$ is the usual lattice-inf connective, $\otimes$ is close to a residual operator with respect to $\rightarrow$. In relevant logic literature $\otimes$ is often called

fusion or *cotenability* and denoted by $\circ$.[18] The addition of the above connectives presents some semantic options. The most important one is whether *distribution* (dist) of $\wedge$ and $\vee$ is assumed or not. The list of axioms/rules below characterizes distributive substructural logics.

DEFINITION 88 (Axioms for $\wedge, \otimes, \vee$).

1. $A \wedge B \to A$,
2. $A \wedge B \to B$,
3. $(C \to A) \wedge (C \to B) \to (C \to A \wedge B)$,
4. $A \to A \vee B$,
5. $B \to A \vee B$,
6. $(A \to C) \wedge (B \to C) \to (A \vee B \to C)$
7. $\dfrac{A \quad B}{A \wedge B}$
8. $\dfrac{A \to B \to C}{A \otimes B \to C}$
9. $\dfrac{A \otimes B \to C}{A \to B \to C}$

(e-$\wedge$) For **E** and **E-W** only

$$\Box A \wedge \Box B \to \Box(A \wedge B)$$

where $\Box C =_{def} (C \to C) \to C$.

(dist) $A \wedge (B \vee C) \to (A \wedge B) \vee C$.

As we have said, the addition of distribution (dist) is a semantic choice, which may be argued. However, for the fragment of the language we consider in this section it does not really matter whether distribution is assumed or not. This fragment roughly corresponds to the Harrop fragment of the section of modal logics (see Section 4.3); since we do not allow positive occurrences of disjunction, the presence of distribution is immaterial.[19] We have included distribution to have a complete axiomatization with respect to the semantics we adopt.

[18] We follow here the terminology and notation of linear logic [Girard, 1987].

[19] In the fragment we consider we trivially have (for any **S**) $\Gamma, \alpha \vdash^? x : A \wedge (B \vee C)$ implies $\Gamma, \alpha \vdash^? x : A \vee (B \wedge C)$, where Γ is a set of D-formulas and A, B, C are G-formulas (see below).

As in the case of modal logics (see Section 4.3), we distinguish D-formulas, which are the constituents of databases, and G-formulas which can be asked as goals.

DEFINITION 89. Let D-formulas and G-formulas be defined as follows:

$$D := q \mid G \to D,$$
$$CD := D \mid CD \wedge CD,$$
$$G := q \mid G \wedge G \mid G \vee G \mid G \otimes G \mid CD \to G.$$

A database Δ is a finite set of *labelled* of D-formulas.

A database corresponds to a $\otimes$-composition of conjunctions of D-formulas. Formulas with the same label are thought as $\wedge$-conjuncted. Every D-formula has the form

$$G_1 \to \ldots \to G_k \to q,$$

In the systems **R**, **L**, **BCK**, we have the theorems

$$(A \to B \to C) \to (A \otimes B \to C) \quad \text{and} \quad (A \otimes B \to C) \to (A \to B \to C)$$

Thus, in these systems we can simplify the syntax of (non atomic) D-formulas to $G \to q$ rather than $G \to D$. This simplification is not allowed in the other systems where we only have the weaker relation

$$\vdash (A \to B \to C) \;\Leftrightarrow\; \vdash A \otimes B \to C.$$

The extent of Definition 89 is shown by the following proposition.

PROPOSITION 90. *Every formula on $(\wedge, \vee, \to, \otimes)$ without*

- *positive*[20] *(negative) occurrences of $\otimes$ and $\vee$ and*
- *occurrences of $\otimes$ within a negative (positive) occurrence of $\wedge$*

is equivalent to a $\wedge$-conjunction of D-formulas (G-formulas).

The reason we have put the restriction on nested occurrences of $\otimes$ within $\wedge$ is that, on the one hand, we want to keep the simple labelling mechanism we have used for the implicational fragment, and on the other we want to identify a common fragment for all systems to which the computation rules are easily extended. The labelling mechanism no longer works if we relax

[20] Positive and negative occurrences are defined as follows: A occurs positively in A; if $B\#C$ occurs positively (negatively) in A (where $\# \in \{\wedge, \vee, \otimes\}$), then B and C occur positively (negatively) in A; if $B \to C$ occurs positively (negatively) in A, then B occurs negatively (positively) in A and C occurs positively (negatively) in n A. We say that a connective $\#$ has a positive (negative) occurrence in a formula A if there is a formula $B\#C$ which occurs positively (negatively) in A.

this restriction. For instance, how could we handle $A \wedge (B \otimes C)$ as a D-formula? We should add $x : A$ and $x : B \otimes C$ in the database. The formula $x : B \otimes C$ cannot be decomposed, unless we use complex labels: intuitively we should split x into some y and z, add $y : B$ and $z : C$, and remember that x, y, z are connected (in terms of Fine semantics the connection would be expressed as $x = y \circ z$).[21]

The computation rules can be extended to this fragment without great effort.

DEFINITION 91 (Proof system for the extended language). We give the rules for queries of the form

$$\Delta, \delta \ \vdash^? \ x : G,$$

where Δ is a set of D-formulas and G is a G-formula.

- (success) $\Delta, \delta \ \vdash^? \ x : q$ succeeds if $x; q \in \Delta$ and $Succ^{\mathbf{S}}(\delta, x)$.
- (implication) from $\Delta, \delta \ \vdash^? \ x : CD \to G$
  if $CD = D_1 \wedge \ldots \wedge D_n$, we step to

  $$\Delta \cup \{y : D_1, \ldots, y : D_n\}, \delta \cup \{y\} \ \vdash^? \ y : G$$

  where $y > \max(Lab(\Delta))$, (hence $y \notin Lab(\Delta)$).
- (reduction) from $\Delta, \delta \ \vdash^? \ x : q$
  if there is $z : G_1 \to \ldots \to G_k \to q \in \Delta$ and there are δ_i, and x_i for $i = 0, \ldots, k$ such that:
  1. $\delta_0 = \{z\}$, $x_0 = z$,
  2. $\bigcup_{i=0}^{k} \delta_i = \delta$,
  3. $Red^{\mathbf{S}}(\delta_0, \ldots, \delta_k, x_0, \ldots, x_k; x)$,

  then for $i = 1, \ldots k$, we step to

  $$\Delta, \delta_i, \ \vdash^? \ x_i : G_i.$$

- (conjunction) from $\Delta, \delta \ \vdash^? \ x : G_1 \wedge G_2$ step to

  $$\Delta, \delta \ \vdash^? \ x : G_1 \text{ and } \Delta, \delta \ \vdash^? \ x : G_2.$$

- (disjunction) from $\Delta, \delta \ \vdash^? \ x : G_1 \vee G_2$ step to

  $$\Delta, \delta \ \vdash^? \ x : G_i \text{ for } i = 1 \text{ or } i = 2.$$

[21] In some logics, such as **L**, we do not need this restriction since we have the following property: $\Gamma, A \wedge B \vdash C$ implies $\Gamma, A \vdash C$ or $\Gamma, B \vdash C$. Thus, we can avoid introducing extensional conjunctions into the database, and instead introduce only one of the two conjuncts (at choice). This approach is followed by Harland and Pym [1991]. However the above property does not hold for **R** and other logics.

- (tensor) from $\Delta, \delta \ \vdash^? \ x : G_1 \otimes G_2$
  if there are δ_1, δ_2, x_1 and x_2 such that

  1. $\delta = \delta_1 \cup \delta_2$,
  2. $Red^{\mathbf{S}}(\delta_1, \delta_2, x_1, x_2; x)$,

  step to

  $$\Delta, \delta_1 \ \vdash^? \ x_1 : G_1 \text{ and } \Delta, \delta_2 \ \vdash^? \ x_2 : G_2.$$

An easy extension of the method is the addition of the truth constants $\mathbf{t}$, and $\top$ which are governed by the following axioms/rules

$A \to \top$,
$\vdash \mathbf{t} \to A$ iff $\vdash A$.

Plus the axiom of $(\mathbf{t} \to A) \to A$ for $\mathbf{E}$ and $\mathbf{E}$-$\mathbf{W}$. We can think of $\mathbf{t}$ as defined by propositional quantification

$$\mathbf{t} =_{def} \forall p(p \to p).$$

Equivalently, given any formula A, we can assume that $\mathbf{t}$ is the conjunction of all $p \to p$ such that the atom p occurs in A. Basing on this definition, it is not difficult to handle $\mathbf{t}$ in the goal-directed way and we leave to the reader to work out the rules. The treatment of $\top$ is straightforward.

EXAMPLE 92. Let Δ be the following database:

$x_1 : e \wedge g \to d$,
$x_2 : (c \to d) \otimes (a \vee b) \to p$,
$x_3 : c \to e$,
$x_3 : c \to g$,
$x_4 : (c \to g) \to b$.

In Figure 9, we show a successful derivation of

$$\Delta, \{x_1, x_2, x_3, x_4\} \ \vdash^? \ x_4 : p$$

in relevant logic $\mathbf{E}$ (and stronger systems). We leave to the reader to justify the steps according to the rules. The success of this query corresponds to the validity of the following formula in $\mathbf{E}$:

$$\big[(e \wedge g \to d) \otimes ((c \to d) \otimes (a \vee b) \to p) \otimes ((c \to e) \wedge \\ \wedge (c \to g)) \otimes ((c \to g) \to b)\big] \to p.$$

We can extend the soundness and completeness result to this larger fragment. We first extend Definition 77 by giving the truth conditions for the additional connectives.

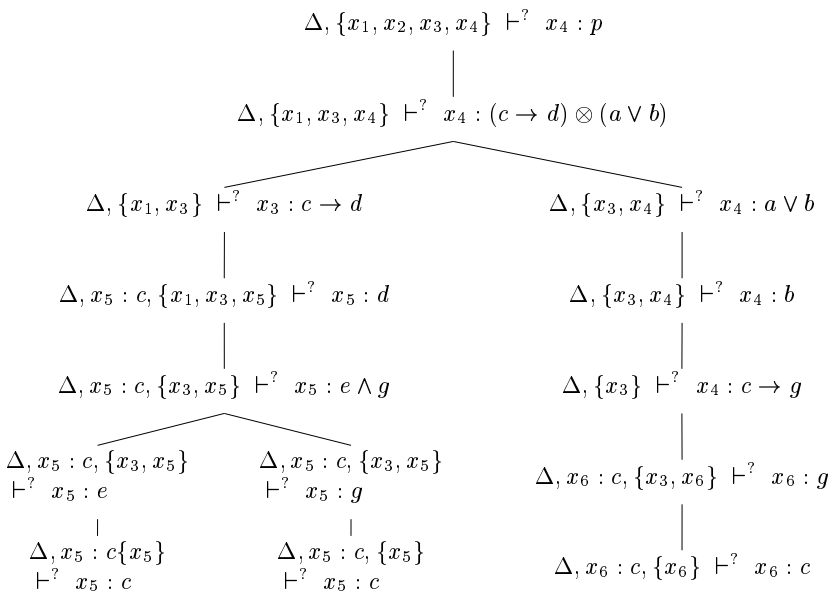


Figure 9. Derivation for Example 92.

DEFINITION 93. Let $M = (W, \circ, 0, V)$ be a **S**-structure, let $a \in W$ we stipulate:

$M, a \models A \wedge B$ iff $M, a \models A$ and $M, a \models B$,

$M, a \models A \vee B$ iff $M, a \models A$ or $M, a \models B$,

$M, a \models A \otimes B$ iff there are $b, c \in W$, s.t. $b \circ c \leq a$ and $M, a \models A$ and $M, a \models B$.

It is straightforward to extend Theorem 87 obtaining the soundness of the proof procedure with respect to this semantics. The completeness can be proved by a canonical model construction. The details are given in [Gabbay and Olivetti, 2000], (see Section 6.1). Putting the two results together we obtain:

THEOREM 94. *Let* $\Gamma, \gamma \vdash^? x : G$ *be a query with* $\gamma = \{x_1, \ldots, x_k\}$ *(ordered as shown), and let* $S_i = \{A \mid x_i : A \in \Gamma\}$, $i = 1, \ldots, k$. *The following are equivalent:*

1. $\models_{\mathbf{S}}^{Fine} (\bigwedge S_1 \otimes \ldots \otimes \bigwedge S_k) \to G$
2. $\Gamma, \gamma \vdash^? x : G$ *succeeds in the system* **S**.

5.4 Eliminating the labels

As in the case of modal logics, it turns out that for most of the systems (at least if we consider the implicational fragment), labels are not needed.

PROPOSITION 95.

- *If* $\Gamma, \gamma \vdash^? x : B$ *succeeds,* $u : A \in \Gamma$ *but* $u \notin \gamma$*, then* $\Gamma - \{u : A\}, \gamma \vdash^? x : B$ *succeeds.*
- *For* **R**, **L**, **BCK**, *if* $\Gamma, \gamma \vdash^? x : B$ *succeeds, then for every* $y \in \gamma$, $\Gamma, \gamma \vdash^? y : B$ *succeeds.*
- *For* **T**,**T-W**,**FL**, *if* $\Gamma, \gamma \vdash^? x : B$ *succeeds then it must be* $x = \max(\gamma)$.

Because of the previous proposition, the formulas which can contribute to the proof are those that are listed in γ. Since every copy of a formula gets a different label, labels and 'usable' formulas are into a 1-1 correspondence. Moreover in all cases, but **E** and **E-W**, the label x in front of the goal is either irrelevant or it is determined by γ. Putting these facts together, we can reformulate the proof systems for all logics, but **E** and **E-W** without using labels. The restrictions on reduction can be expressed directly in terms of the sequence of database formulas. In other words, formulas and labels become the same things. As an example we give the reformulation of **L** and **FL**. In case of **L**, it is easily seen that the order of formulas does not matter (it can be proved that permuting the database does not affect the success of a query), thus the database can be thought as a multiset of formulas, and the rules become as follows:

1. (success) $\Delta \vdash^? q$ succeeds if $\Delta = q$,
2. (implication) from $\Delta \vdash^? A \to B$ step to $\Delta, A \vdash^? B$,
3. (reduction) from $\Delta, C \to \ldots \to C_n \to q \vdash^? q$ step to $\Delta_i \vdash^? C_i$, where $\Delta = \sqcup_i \Delta_i$, ($\sqcup$ denotes multiset union).

In other words we obtain what we have called the *linear computation* in Section 3.3, Definition 13.

In case of **FL**, the order of the formulas is significant. Thus the rules for success and implication are the same, but in case of implication the formula A is added to the *end* of Δ. The rule of reduction becomes:

> (**FL**-reduction) from $C \to \ldots \to C_n \to q, \Delta \vdash^? q$ step to $\Delta_i \vdash^? C_i$, where $\Delta = \Delta_1, \ldots, \Delta_n$,

in this case "," denotes concatenation. The reformulation without labels for **R**, **T**, and **T-W**, **BCK** is similar and left to the reader.

5.5 Some history

The use of labels to deal with substructural logics is not a novelty: it has been used for instance by Prawitz [1965], by Anderson and Belnap [1975], to develop a natural-deduction formulation of most relevant logics. The goal-directed proof systems we have presented are similar to their natural deduction systems in this respect: there are not explicit structural rules. The structural rules are *internalized* as restrictions in the logical rules.

Bollen has developed a goal-directed procedure for a fragment of first-order **R** [Bollen, 1991]). His method avoids splitting derivations in several branches by maintaining a global proof-state $\Delta \vdash^? [A_1, \ldots, A_n]$, where all $A_1, \ldots A_n$ have to be proved (they can be thought as linked by $\otimes$). If we want to keep all subgoals together, we must take care that different subgoals A_i may happen to be evaluated in different contexts. For instance in

$$C \vdash^? D \to E, F \to G$$

according to the implication rule, we must evaluate E from $\{C, D\}$, and G from $\{C, F\}$. Bollen accommodates this kind of context-dependency by indexing subgoals with a number which refers to the part of the database that can be used to prove it. Furthermore, a list of numbers is maintained to remember the usage; in a successful derivation the usage list must contain the numbers of all formulas of the database.

Many people have developed goal-directed procedures for fragments of linear logic leading to the definition of logic programming languages based on linear logic . This follows the tradition of the *uniform proof paradigm* proposed by Miller [Miller *et al.*, 1991]. We notice, in passing, that implicational **R** can be encoded in linear logic by defining $A \to B$ by $A \multimap !A \multimap B$, where ! is the exponential operator which enables the contraction of the formula to which is applied. The various proposals differ in the choice of the language fragment. Much emphasis is given to the treatment of the exponential !, as it is needed for defining logic programming languages of some utility: in most applications, we need permanent resources or data, (i.e. data that are not 'consumed' along a deduction); permanent data can be represented by using the ! operator.

Some proposals, such as [Harland and Pym, 1991] and [Andreoli and Pareschi, 1991; Andreoli, 1992] take as basis the multi-consequent (or classical) version of linear logic. Moreover, mixed systems have been studied [Hodas and Miller, 1994] and [Hodas, 1993] which combine different logics into a single language: linear implication, intuitionistic implication and more.[22]

[22] In [Hodas, 1993], Hodas has proposed a language called O which combines intuitionistic, linear, affine and relevant implication. The idea is to partition the 'context', i.e. the database in several (multi) sets of data corresponding to the different handling of the data according to each implicational logic.

O'Hearn and Pym [1999] have recently proposed an interesting development of linear logic called the *logic of bunched implications*. Their system has strong semantical and categorical motivations. The logic of bunched implications combines a multiplicative implication, namely the one of linear logic and an additive (or intuitionistic) one. The proof contexts, that is the antecedents of a sequent, are structures built by two operators: ',' corresponds to the multiplicative conjunction $\otimes$ and ';' corresponds to the additive conjunction $\wedge$. These structures are called *bunches*. Similar structures have been used to obtain calculi for distributive relevant logics [Dunn, 1986]. The authors define also an interesting extension to the first-order case by introducing intensional quantifiers. Moreover they develop a goal-directed proof procedure for a Harrop-fragment of this logic and show significant applications to logic programming.

In [Gabbay and Olivetti, 2000], the goal-directed systems are extended to **RM0** where a suitable restart rule takes care of the mingle rule. Moreover, it is shown how the goal-directed method for **R** can be turned into a decision procedure (for the pure implicational part) by adding a suitable loop-checking mechanism. We notice that for contractionless logics, the goal-directed proof-methods of this section can be the base of decision procedures.

To implement goal-directed proof systems for substructural logics, one can import solutions and techniques developed in the context of linear logic programming. For instance, in the reduction and in the $\otimes$-rule we have to guess a partion of the goal label. A similar problem has been discussed in the linear logic programming community [Hodas and Miller, 1994; Harland and Pym, 1997] where one has to guess a split of the sequent (only the antecedent in the intuitionistic version, both the antecedent and consequent in the 'classical' version). A number of solutions have been provided, most of them based on a *lazy* computation of the split parts. Perhaps the most general way to handle this problem has been addressed in [Harland and Pym, 1997] where it is proposed to represent the sequent split by means of Boolean constraints expressed by labels attached to the formulas. Different strategies of searching the partitions correspond to different strategies of solving the constraints (lazy, eager and mixed).

6 DEVELOPMENTS, APPLICATIONS AND MECHANISMS

The goal-directed proof methods presented in this chapter may be useful for several purposes, beyond the mere deductive task. The deductive procedures can be easily extended in order to compute further information, or to support other logical tasks, such as abduction. We show two examples: the computation of interpolants in implicational linear logic, and the computation of abductive explanations in intuitionistic logic. Both of them are

simple extensions of the goal-directed procedures that we have seen in the previous sections. We conclude by a discussion about the extension of the goal-directed methods to the first order case.

6.1 Interpolation for linear implication

By interpolation we mean the following property. Let Q_1 and Q_2 be two sets of atoms; let $\mathcal{L}_1$ be the language generated by Q_1 and $\mathcal{L}_2$ by Q_2; finally let A, B be formulas with $A \in \mathcal{L}_1$ and $B \in \mathcal{L}_2$. If $A \vdash B$, then there is a formula $C \in \mathcal{L}_1 \cap \mathcal{L}_2$, such that $A \vdash C$ and $C \vdash B$. The formula C is called an *interpolant* of A and B. Here, we consider the interpolation property for implicational linear logic, i.e. $\vdash$ denotes provability in (implicational) linear logic, and A, B contain only implication. We show how the procedure of the previous section can be modified to compute an interpolant. Since the computation is defined for sequents of the form $\Gamma \vdash A$, where Γ is a database (a multiset of formulas in this case), we need to generalize the notion of interpolation to provability statements of this form. As a particular case we will have the usual interpolation property for pair of formulas. To generalize interpolation to database we need first to generalize the provability relation to databases. Let $\Delta = A_1, \ldots, A_n$, we define:

> $\Gamma \vdash \Delta$ iff there are $\Delta_1, \ldots, \Delta_n$, such that $\Delta = \Delta_1 \sqcup \ldots \sqcup \Delta_n$ and $\Delta_i \vdash A_i$ for $i = 1, \ldots, n$.

The formulas A_i can be thought as conjuncted by $\otimes$. Moreover, since $A_i \to \ldots \to A_n \to B$, is equivalent to $A_1 \otimes \ldots \otimes A_n \to B$, and $\otimes$ is associative and commutative, we abbreviate the above formula with $\Delta \to B$, where Δ is the multiset $A_1, \ldots, A_n$. Let $\Gamma \in \mathcal{L}_1$ and $A \in \mathcal{L}_2$, suppose that $\Gamma \vdash A$ holds; an *interpolant* for $\Gamma \vdash A$ is a database $\Pi \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $\Gamma \vdash \Pi$ and $\Pi \vdash A$. Oberve that if $\mid \Gamma \mid = 1$, it must be also $\mid \Pi \mid = 1$, and we have the ordinary notion of interpolant.

We give a recursive procedure to calculate an interpolant for $\Gamma \vdash A$. We do this by defining a recursive predicate $Inter(\Gamma; \Delta; A; \Pi)$ which is true whenever: $\Gamma \in \mathcal{L}_1$, $\Delta, A \in \mathcal{L}_2$, $\Gamma, \Delta \vdash A$ holds, and Π is an interpolant for $\Gamma \vdash \Delta \to A$.

- (Succ 1) $Inter(q; \emptyset; q; q)$.
- (Succ 2) $Inter(\emptyset; q; q; \emptyset)$.
- (Imp) $Inter(\Gamma; \Delta; A \to B, \Pi)$ if $Inter(\Gamma; \Delta, A; B, \Pi)$.
- (Red 1) $Inter(\Gamma, C_1 \to \ldots C_n \to q; \Delta; q; \Pi)$, if there are Γ_i, Δ_i, Π_i for $i = 1, \ldots, n$ such that
  1. $\Gamma = \sqcup_i \Gamma_i$, $\Delta = \sqcup_i \Delta_i$,

2. $Inter(\Delta_i; \Gamma_i; C_i; \Pi_i)$ for $i = 1, \ldots, n$,
3. $\Pi = \Pi_1 \sqcup \ldots \sqcup \Pi_n \to q$.

- (Red 2) $Inter(\Gamma; \Delta, C_1 \to \ldots C_n \to q; q; \Pi)$, if there are Γ_i, Δ_i, Π_i for $i = 1, \ldots, n$ such that

1. $\Gamma = \sqcup_i \Gamma_i$, $\Delta = \sqcup_i \Delta_i$, $\Pi = \sqcup_i \Pi_i$,
2. $Inter(\Gamma_i; \Delta_i; C_i; \Pi_i)$ for $i = 1, \ldots, n$.

If we want to find an interpolant for $\Gamma \vdash A$, the initial call is $Inter(\Gamma; \emptyset; A; \Pi)$, where Π is computed along the derivation. One can observe that the predicate $Inter$ is defined by following the definition of the goal-directed proof procedure. (Succ 1) and (Succ 2) apply to the case of immediate success, respectively when the atom is in $\mathcal{L}_1$, and when it is in $\mathcal{L}_2$. Analogously, (Red 1) deals with the case the atomic goal unifies with the head of a formula in $\mathcal{L}_1$ and (Red 2) when it unifies with the head of a formula in $\mathcal{L}_2$. It can be proved that if $\Gamma \vdash A$ the algorithm computes an interpolant. Moreover all interpolants can be obtained by selecting different formulas in the reduction steps. Here is a non trivial example.

EXAMPLE 96. Let

$$A = ((f \to e) \to ((a \to b) \to c) \to (a \to q) \to (q \to b) \to f \to b) \to g,$$

and

$$B = (e \to c \to d) \to (d \to a) \to (a \to b) \to g.$$

One can check that $A \vdash B$ holds; we compute an interpolant, by calling $Inter(A; \emptyset; B; \Pi)$. Let us abbreviate the antecedent of A by A'. Here are the steps, we underline the formula used in a reduction step:

(1) $Inter(\underline{A}; \emptyset;\ B;\ \Pi)$

(2) $Inter(A;\ e \to c \to d, d \to a, a \to b;\ g;\ \Pi)$

(3) $Inter(e \to c \to d, d \to a, a \to b;\ \emptyset;\ A', \Pi_1) \quad \Pi = \Pi_1 \to g$

(4) $Inter(e \to c \to d, d \to a, a \to b;\ f \to e, (a \to b) \to c, a \to q, \underline{q \to b}, f;\ b, \Pi_1)$

(5) $Inter(e \to c \to d, d \to a, a \to b;\ f \to e, (a \to b) \to c, \underline{a \to q}, f;\ q, \Pi_1)$

(6) $Inter(e \to c \to d, \underline{d \to a}, a \to b;\ f \to e, (a \to b) \to c, f;\ a, \Pi_1)$

(7) $Inter(f \to e, (a \to b) \to c, f;\ \underline{e \to c \to d}, a \to b;\ ;\ d, \Pi_2)$
$\Pi_1 = \Pi_2 \to a$

Now we have a split with $\Pi_2 = \Pi_3, \Pi_4$ and

(8) $Inter(\underline{f \to e, f};\ \emptyset;\ e;\ \Pi_3)$ and (9) $Inter(\underline{(a \to b) \to c};\ a \to b;\ ;\ c, \Pi_4)$.

(8) gives

(10) $Inter(\emptyset;\ f;\ f;\ \Pi_5) \quad \Pi_3 = \Pi_5 \to e$

whence $\Pi_5 = \emptyset$. (9) gives

(11) $Inter(a \to b;\ \emptyset;\ a \to b;\ \Pi_6) \quad \Pi_4 = \Pi_6 \to c$

(12) $Inter(\underline{a \to b};\ a;\ b;\ \Pi_6)$

(13) $Inter(a;\ \emptyset;\ a;\ \Pi_7) \quad \Pi_6 = \Pi_7 \to b$

Thus $\Pi_7 = a$ and we have:

$\Pi_6 = a \to b$,

$\Pi_3 = e$,

$\Pi_4 = (a \to b) \to c$,

$\Pi_2 = e, (a \to b) \to c$,

$\Pi_1 = e \to ((a \to b) \to c) \to a$,

$\Pi = (e \to ((a \to b) \to c) \to a) \to g$.

Π is evidently in the common language of A and B; we leave to the reader to check that $A \vdash \Pi$ and $\Pi \vdash B$.

It is likely that similar procedures based on the goal-directed computation can be devised for other logics admitting a goal directed presentation. However, further investigation is needed to see to what extent this approach works for other cases. For instance, the step in (Red 1) is justified by the structural *exchange* law which holds for linear logic. For non-commutative logics the suitable generalization of the interpolation property, which is the base of the inductive procedure, might be more difficult to find.

6.2 Abduction for intuitionistic implication

Abduction is an important kind of inference for many applications. It has been widely and deeply studied in logic programming context (see [Eshghi, 1989] for a seminal work). Abductive inference can be described as follows: given a set of data Γ and a formula A such that $\Gamma \not\vdash A$, find a set of formulas Π such that $\Gamma, \Pi \vdash A$. Usually, there are some further requirements on the possible Π's, such as *minimality*. The set Π is called an abductive solution for $\Gamma \vdash A$. We define below a metapredicate $Abduce(\Gamma;\ A;\ \Pi)$ whose meaning, is that Π is an abductive solution for $\Gamma \vdash A$. The predicate is defined by induction on the goal-directed derivation. Given a formula C and a database Γ, we write $C \to \Gamma$ to denote the set $\{C \to D \mid D \in \Gamma\}$.

1. $Abduce(\Gamma;\ q;\ \Pi)$ if $q \in \Gamma$ and $\Pi = \emptyset$.
2. $Abduce(\Gamma;\ q;\ \Pi)$ if $\forall C \in \Gamma\ q \neq Head(C)$ and $\Pi = \{q\}$.
3. $Abduce(\Gamma;\ A \to B;\ \Pi)$ if $Abduce(\Gamma, A;\ B;\ \Pi')$ and $\Pi = A \to \Pi'$.
4. $Abduce(\Gamma;\ q;\ \Pi)$ if there is $C_1 \to \ldots \to C_n \to q \in \Gamma$ and $\Pi_1, \ldots, \Pi_n$ such that
   (a) $Abduce(\Gamma; C_i, \Pi_i)$ for $i = 1, \ldots, n$.
   (b) $\Pi = \bigcup_i \Pi_i$.

It can be shown that if $Abduce(\Gamma;\ q;\ \Pi)$ holds then $\Gamma, \Pi \vdash A$.

EXAMPLE 97. Let $\Gamma = (a \to c) \to b, (d \to b) \to s \to q, (p \to q) \to t \to r$. We compute $Abduce(\Gamma;\ r;\ \Pi)$. We have

(1) $Abduce(\Gamma;\ r;\ \Pi)$ reduces to

(2) $Abduce(\Gamma;\ p \to q;\ \Pi_1)$ and (3) $Abduce(\Gamma;\ t;\ \Pi_2)$ with $\Pi = \Pi_1 \cup \Pi_2$; (3) gives $\Pi_2 = \{t\}$. (2) is reduced as follows

(4) $Abduce(\Gamma, p;\ q;\ \Pi_3)$ and $\Pi_1 = p \to \Pi_3$

(5) $Abduce(\Gamma, p;\ d \to b;\ \Pi_4)$ and (6) $Abduce(\Gamma, p;\ s;\ \Pi_5)$ with $\Pi_3 = \Pi_4 \cup \Pi_5$; (6) gives $\Pi_5 = \{s\}$. (5) is reduced as follows

(7) $Abduce(\Gamma, p, d;\ b;\ \Pi_6)$ and $\Pi_4 = b \to \Pi_6$

(8) $Abduce(\Gamma, p, d;\ a \to c;\ \Pi_6)$

(9) $Abduce(\Gamma, p, d, a;\ c;\ \Pi_7)$ and $\Pi_6 = a \to \Pi_7$
(9) gives $\Pi_7 = \{c\}$.

Thus

$\Pi_6 = \{a \to c\}$

$\Pi_4 = \{b \to a \to c\}$

$\Pi_3 = \{b \to a \to c, s\}$

$\Pi_1 = \{p \to b \to a \to c, p \to s\}$

$\Pi = \{p \to b \to a \to c, p \to s, t\}$.

One can easily check that $\Gamma, \Pi \vdash r$.

The abductive proof procedure we have exemplified is a sort of metapredicate defined from the goal-directed computation. It can be used to generate possible abductive solutions, that might be then compared and processed further. Of course variants and refinments of the abductive procedures are possible for specific purposes.

6.3 First-order extension

We conclude this chapter by some remarks on the major extension of these methods: the one to the first-order level. The extension is not straightforward. In general, in most non-classical logics there are several options in the interpretation of quantifiers and terms according to the intended semantics (typically, constant domain, increasing domains etc.); moreover, one may adopt either rigid, or non-rigid interpretation of terms. Sometimes the interpretation of quantifiers is not entirely clear, as it happens in substructural logics. However, even when quantifiers are well understood as in intuitionistic and modal logics, a goal-directed treatment of the full language creates troubles analogously to the treatment of disjunction. In particular the treatment of positive occurrences of the existential quantifier is problematic. In classical logic such a problem does not arise as one can always transform the pair (Database, Goal) into a set of universal sentences using Skolem transformation. A similar method cannot be applied to most non-classical logics, where one cannot skolemize the data before the computation starts.

As a difference with the theorem proving view, in the goal-directed approach (following the line of logic programming) one would like to define proof procedures which compute answer-substitutions. This means that the outcome of a successful computation of

$$\Delta \vdash^{?} G[X],$$

where $G[X]$ stands for $\exists X G[X]$ is not only 'yes', but it is (a most general) substitution X/t, such that $\Delta \vdash G[X/t]$ holds. In [Gabbay and Olivetti, 2000] it is presented a proof procedure for $(\rightarrow, \forall)$ fragment of intuitionistic logic. The procedure is in the style of a logic programming and uses unification to compute answer-substitutions. The proof-procedure checks the simultaneous success of a set of pairs (Database, Goals); this structure is enforced by the presence of shared free variables occurring in different databases arising during the computation. The approach of 'Run-time skolemization' is used to handle universally quantified goals.[23] That is to say the process of eliminating universal quantifiers on the goal by Skolem functions is carried on in parallel with goal reduction. The 'Run-time Skolemisation' approach has been presented in [Gabbay, 1992; Gabbay and Reyle, 1993] and adopted in N-Prolog [Gabbay and Reyle, 1993], a hypothetical extension of Prolog based on intuitionistic logic. A similar idea for classical logic is embodied in the free-variable tableaux, see [Fitting, 1990] and [Hähnle and Schmitt, 1994] for an improved rule. The use of Skolem functions and normal form for intuitionistic proof-search has been studied by many authors, we just mention: [Shankar, 1992], [Pym and Wallen, 1990], and [Sahlin *et al.*, 1992].

[23] These would be existentially quantified formulas in classical logic as checking $\Delta \vdash \forall x A$ is equivalent to check $\Delta \cup \{\exists x \neg A\}$ for inconsistency.

It is likely that in order to develop the first-order extensions for others non-classical logics one could take advantage of the labelled proof system. One would like to represent the semantic options on the interpretation of quantifiers by tinkering with the unification and the Skolemisation mechanism. The constraints on the unification and Skolemisation might perhaps be expressed in terms of the labels associated with the formulas and their dependencies. At present the extension of the goal-directed methods to the main families of non-classical logics along these lines is not at hand and it is a major topic of future investigation.

Dov M. Gabbay
King's College London, UK.

Nicola Olivetti
Università di Torino, Italy.

BIBLIOGRAPHY

[Abadi and Manna, 1989] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, **8**, 277–295, 1989.

[Andreoli, 1992] J. M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal Logic and Computation*, **2**, 297–347, 1992.

[Andreoli and Pareschi, 1991] J. M. Andreoli and R. Pareschi. Linear objects: logical processes with built in inheritance. *New Generation Computing*, **9**, 445–474, 1991.

[Anderson and Belnap, 1975] A. R. Anderson and N. D. Belnap. *Entailment, The Logic of Relevance and Necessity*, Vol. 1. Princeton University Press, New Jersey, 1975.

[Anderson *et al.*, 1992] A. R. Anderson, N. D. Belnap and J. M. Dunn. *Entailment, The Logic of Relevance and Necessity*, Vol. 2. Princeton University Press, New Jersey, 1992.

[Baldoni *et al.*, 1998] M. Baldoni, L. Giordano and A. Martelli. A modal extension of logic programming, modularity, beliefs and hypothetical reasoning. *Journal of Logic and Computation*, **8**, 597–635, 1998.

[Basin *et al.*, 1997a] D. Basin, S. Matthews and L. Viganò. Labelled propositional modal logics: theory and practice. *Journal of Logic and Computation*, **7**, 685–717, 1997.

[Basin *et al.*, 1997b] D. Basin, S. Matthews and L. Viganò. A new method for bounding the complexity of modal logics. In *Proceedings of the Fifth Gödel Colloquim*, pp. 89–102, LNCS 1289, Springer-Verlag, 1997.

[Basin *et al.*, 1999] D. Basin, S. Matthews and L. Viganò. Natural deduction for non-classical logics. *Studia Logica*, **60**, 119–160, 1998.

[Bollen, 1991] A. W. Bollen. Relevant logic programming, *Journal of Automated Reasoning*, **7**, 563–585, 1991.

[Došen, 1988] K. Došen. Sequent systems and grupoid models I. *Studia Logica*, **47**, 353–385, 1988.

[Došen, 1989] K. Došen. Sequent systems and grupoid models II. *Studia Logica*, **48**, 41–65, 1989.

[Dummett, 2001] M. Dummett. *Elements of Intuitionism*, Oxford University Press, (First edn. 1977), second edn., 2001.

[Dunn, 1986] J. M. Dunn. Relevance logic and entailment. In *Handbook of Philosophical Logic*, vol III, D. Gabbay and F. Guenthner, eds. pp. 117–224. D. Reidel, Dordercht, 1986.

[Dyckhoff, 1992] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic, *Journal of Symbolic Logic*, **57**, 795–807, 1992.

[Eshghi, 1989] K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In *Proceedings of the 6th ICLP*, pp. 234–254, Lisbon, 1989.

[Farinãs, 1986] L. Farinãs del Cerro. MOLOG: a system that extends Prolog with modal logic. *New Generation Computing*, **4**, 35–50, 1986.

[Fine, 1974] K. Fine. Models for entailment, *Journal of Philosophical Logic*, **3**, 347–372, 1974.

[Fitting, 1983] M. Fitting. *Proof methods for Modal and Intuitionistic Logic*, vol 169 of Synthese library, D. Reidel, Dorderecht, 1983.

[Fitting, 1990] M. Fitting. *First-order Logic and Automated Theorem Proving.* Springer-Verlag, 1990.

[Gabbay, 1981] D. M. Gabbay. *Semantical Investigations in Heyting's Intuitionistic Logic.* D. Reidel, Dordrecht, 1981.

[Gabbay, 1985] D. M. Gabbay. N-Prolog Part 2. *Journal of Logic Programming*, 251–283, 1985.

[Gabbay, 1987] D. M. Gabbay. Modal and temporal logic programming. In *Temporal Logics and their Applications*, A. Galton, ed. pp/ 197–237. Academic Press, 1987.

[Gabbay, 1991] D. M. Gabbay. Algorithmic proof with diminishing resources. In *Proceedings of CSL'90*, pp. 156–173. LNCS vol 533, Springer-Verlag, 1991.

[Gabbay, 1992] D. M. Gabbay. Elements of algorithmic proof. In *Handbook of Logic in Theoretical Computer Science*, vol 2. S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, eds., pp. 307–408, Oxford University Press, 1992.

[Gabbay, 1996] D. M. Gabbay, *Labelled Deductive Systems* (vol I), Oxford Logic Guides, Oxford University Press, 1996.

[Gabbay, 1998] D. M. Gabbay. *Elementary Logic*, Prentice Hall, 1998.

[Gabbay and Kriwaczek, 1991] D. M. Gabbay and F. Kriwaczek. A family of goal-directed theorem-provers based on conjunction and implication. *Journal of Automated Reasoning*, **7**, 511–536, 1991.

[Gabbay and Olivetti, 1997] D. M. Gabbay and N. Olivetti. Algorithmic proof methods and cut elimination for implicational logics - part I, modal implication. *Studia Logica*, **61**, 237–280, 1998.

[Gabbay and Olivetti, 1998] D. Gabbay and N. Olivetti. Goal-directed proof-procedures for intermediate logics. In *Proc. of LD'98 First International Workshop on Labelled Deduction*, Freiburg, 1998.

[Gabbay *et al.*, 1999] D. M. Gabbay, N. Olivetti and S. Vorobyov. Goal-directed proof method is optimal for intuitionistic propositional logic. Manuscript, 1999.

[Gabbay and Olivetti, 2000] D.M. Gabbay and N. Olivetti. *Goal-directed proof theory*, Kluwer Academic Publishers, Applied Logic Series, Dordrecht, 2000.

[Gabbay and Reyle, 1984] D. M. Gabbay and U. Reyle. N-Prolog: an Extension of Prolog with hypothetical implications, I. *Journal of Logic Programming*, **4**, 319–355, 1984.

[Gabbay and Reyle, 1993] D. M. Gabbay and U. Reyle. Computation with run time Skolemisation (N-Prolog, Part 3). *Journal of Applied Non Classical Logics*, **3**, 93–128, 1993.

[Gallier, 1987] J. H. Gallier. *Logic for Computer Science*, John Wiley, New York, 1987.

[Galmiche and Pym, 1999] D. Galmiche and D. Pym. Proof-Search in type-theoretic languages. To appear in *Theoretical Computer Science*, 1999.

[Giordano *et al.*, 1992] L. Giordano, A. Martelli and G. F. Rossi. Extending Horn clause logic with implication goals. *Theoretical Computer Science*, **95**, 43–74, 1992.

[Giordano and Martelli, 1994] L. Giordano and A. Martelli. Structuring logic programs: a modal approach. *Journal of Logic Programming*, **21**, 59–94, 1994.

[Girard, 1987] J. Y. Girard. Linear logic. *Theoretical Computer Science* **50**, 1–101, 1987.

[Goré, 1999] R. Goré. Tableaux methods for modal and temporal logics. In *Handbook of Tableau Methods*, M. D'Agostino *et al.*, eds. Kluwer Academic Publishers, 1999.

[Hähnle and Schmitt, 1994] R. Hähnle and P. H. Schmitt. The liberalized δ-rule in free-variables semantic tableaux. *Journal Automated Reasoning*, **13**, 211–221, 1994.

[Harland and Pym, 1991] J. Harland and D. Pym. The uniform proof-theoretic foundation of linear logic programming. In *Proceedings of the 1991 International Logic Programming Symposium*, San Diego, pp. 304–318, 1991.

[Harland, 1997] J. Harland. On Goal-Directed Provability in Classical Logic, *Computer Languages*, pp. 23:2-4:161-178, 1997.

[Harland and Pym, 1997] J. Harland and D. Pym. Resource-distribution by Boolean constraints. In *Proceedings of CADE 1997*, pp. 222–236, Springer Verlag, 1997.

[Heudering *et al.*, 1996] A. Heudering, M. Seyfried and H. Zimmerman. Efficient loop-check for backward proof search in some non-classical propositional logics. In (P. Miglioli et al. eds.) *Tableaux 96*, pp. 210–225. LNCS 1071, Springer Verlag, 1996.

[Hodas, 1993] J. Hodas. *Logic programming in intuitionistic linear logic: theory, design, and implementation.* PhD Thesis, University of Pennsylvania, Department of Computer and Information Sciences, 1993.

[Hodas and Miller, 1994] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, **110**, 327–365, 1994.

[Hudelmaier, 1990] J. Hudelmaier. Decision procedure for propositional n- prolog. In (P. Schroeder-Heister ed.) *Extensions of Logic Programming*, pp. 245–251, Springer Verlag, 1990.

[Hudelmaier, 1990] J. Hudelmaier. An $O(n \log n)$-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, **3**, 63–75, 1993.

[Lambek, 1958] J. Lambek. The mathematics of sentence structure, *American Mathematics Monthly*, **65**, 154–169, 1958.

[Lewis, 1912] C. I. Lewis. Implication and the algebra of logic. *Mind*, **21**, 522–531, 1912.

[Lewis and Langford, 1932] C. I. Lewis and C. H. Langford. *Symbolic Logic*, The Century Co., New York, London, 1932. Second edition, Dover, New York, 1959.

[Lincoln *et al.*, 1991] P. Lincoln, P. Scedrov and N. Shankar. Linearizing intuitionistic implication. In *Proceedings of LICS'91*, G. Kahn ed., pp. 51–62, IEEE, 1991.

[Lloyd, 1984] J. W. Lloyd, *Foundations of Logic Programming*, Springer, Berlin, 1984.

[Loveland, 1991] D. W. Loveland. Near-Horn prolog and beyond. *Journal of Automated Reasoning*, **7**, 1–26, 1991.

[Loveland, 1992] D. W. Loveland. A comparison of three Prolog extensions. *Journal of Logic Programming*, **12**, 25-50, 1992.

[Masini, 1992] A. Masini. 2-Sequent calculus: a proof theory of modality. *Annals of Pure and Applied Logics*, **59**, 115–149, 1992.

[Massacci, 1994] F. Massacci. Strongly analytic tableau for normal modal logics. In *Proceedings of CADE '94*, LNAI 814, pp. 723–737, Springer-Verlag, 1994.

[McCarty, 1988a] L. T. McCarty. Clausal intuitionistic logic. I. Fixed-point semantics. *Journal of Logic Programming*, **5**, 1–31, 1988.

[McCarty, 1988b] L. T. McCarty. Clausal intuitionistic logic. II. Tableau proof procedures. *Journal of Logic Programming*, **5**, 93–132, 1988.

[McRobbie and Belnap, 1979] M. A. McRobbie and N. D. Belnap. Relevant analytic tableaux. *Studia Logica*, **38**, 187–200, 1979.

[Miller *et al.*, 1991] D. Miller, G. Nadathur, F. Pfenning and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, **51**, 125–157, 1991.

[Miller, 1992] D. A. Miller. Abstract syntax and logic programming. In *Logic Programming: Proceedings of the 2nd Russian Conference*, pp. 322–337, LNAI 592, Springer Verlag, 1992.

[Nadathur, 1998] G. Nadathur. Uniform provability in classical logic. *Journal of Logic and Computation*, **8**, 209–229,1998.

[O'Hearn and Pym, 1999] P. W. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, **5**, 215–244, 1999.

[Olivetti, 1995] N. Olivetti. *Algorithmic Proof-theory for Non-classical and Modal Logics.* PhD Thesis, Dipartimento di Informatica, Universita' di Torino, 1995.

[Ono, 1993] H. Ono. Semantics for substructural logics. In P. Schroeder-Heister and K. Došen(eds.) *Substructural Logics*, pp. 259–291, Oxford University Press, 1993.

[Ono, 1998] H. Ono. Proof-theoretic methods in non-classical logics-an introduction. In (M. Takahashi et al. eds.) *Theories of Types and Proofs*, pp. 267–254, Mathematical Society of Japan, 1998.

[Prawitz, 1965] D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, 1965.

[Pym and Wallen, 1990] D. J. Pym and L. A. Wallen. Investigation into proof-search in a system of first-order dependent function types. *In* Proc. of CADE'90, pp. 236–250. LNCS vol 449, Springer-Verlag, 1990.

[Restall, 1999] G. Restall. *An introduction to substructural logics*. Routledge, to appear, 1999.

[Russo, 1996] A. Russo. Generalizing propositional modal logics using labelled deductive systems. In (F. Baader and K. Schulz, eds.) *Frontiers of Combining Systems*, pp. 57–73. Vol. 3 of Applied Logic Series, Kluwer Academic Publishers, 1996.

[Sahlin *et al.*, 1992] D. Sahlin, T. Franzén, and S. Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, **2**, 619–656, 1992.

[Schroeder-Heister and Došen, 1993] P. Schroeder-Heister and K. Došen(eds.) *Substructural Logics*. Oxford University Press, 1993.

[Shankar, 1992] N. Shankar. Proof search in the intuitionistic sequent calculus. In *Proc. of CADE'11*, pp. 522–536. LNAI 607, Springer Verlag, 1992.

[Statman, 1979] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, **9**, 67–72, 1979.

[Thistlewaite *et al.*, 1988] P. B. Thistlewaite, M. A. McRobbie and B. K. Meyer. *Automated Theorem Proving in Non Classical Logics*, Pitman, 1988.

[Turner, 1985] R. Turner. *Logics for Artificial Intelligence*, Ellis Horwood Ltd., 1985.

[Troelstra, 1969] A. S. Troelstra. *Principles of Intuitionism*. Springer-Verlag, Berlin, 1969.

[Urquhart, 1972] A. Urquhart. The semantics of relevance logics. *The Journal of Symbolic Logic*, **37**, 159–170, 1972.

[Urquhart, 1984] A. Urquhart. The undecidability of entailment and relevant implication. *The Journal of Symbolic Logic*, **49**, 1059–1073, 1984.

[van Dalen, 1986] D. Van Dalen. Intuitionistic logic. In *Handbook of Philosophical logic, Vol 3*, D. Gabbay and F. Guenther, eds. pp. 225–239. D. Reidel, Dorderecht, 1986.

[Viganò, 1999] L. Viganò. Labelled Non-classical Logics. Kluwer Academic Publishers, 2000. to appear.

[Wallen, 1990] L. A. Wallen, *Automated Deduction in Nonclassical Logics*, MIT Press, 1990.

[Wansing, 1994] H. Wansing. Sequent Calculi for normal propositional modal logics. *Journal of Logic and Computation*, **4**, 125–142, 1994.

ARNON AVRON

ON NEGATION, COMPLETENESS AND CONSISTENCY

1 INTRODUCTION

In this Chapter we try to understand negation from two different points of view: a syntactical one and a semantic one. Accordingly, we identify two different types of negation. The same connective of a given logic might be of both types, but this might not always be the case.

The syntactical point of view is an abstract one. It characterizes connectives according to the internal *role* they have inside a logic, regardless of any meaning they are intended to have (if any). With regard to negation our main thesis is that the availability of what we call below an internal negation is what makes a logic essentially *multiple-conclusion*.

The semantic point of view, in contrast, is based on the intuitive meaning of a given connective. In the case of negation this is simply the intuition that the negation of a proposition A is true if A is not, and not true if A is true.[1]

Like in most modern treatments of logics (see, e.g., [Scott, 1974; Scott, 1974b; Hacking, 1979; Gabbay, 1981; Urquhart, 1984; Wojcicki, 1988; Epstein, 1995; Avron, 1991a; Cleave, 1991; Fagin *et al.*, 1992]), our study of negation will be in the framework of Consequence Relations (CRs). Following [Avron, 1991a], we use the following rather general meaning of this term:

DEFINITION.

(1) A *Consequence Relation* (CR) on a set of formulas is a binary relation $\vdash$ between (finite) multisets of formulas s.t.:

(I) Reflexivity: $A \vdash A$ for every formula A.

(II) Transitivity, or "Cut": if $\Gamma_1 \vdash \Delta_1$, A and $A, \Gamma_2 \vdash \Delta_2$, then $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$.

(III) Consistency: $\emptyset \nvdash \emptyset$ (where $\emptyset$ is the empty multiset).

[1] We have avoided here the term "false", since we do not want to commit ourselves to the view that A is false precisely when it is not true. Our formulation of the intuition is therefore obviously circular, but this is unavoidable in intuitive informal characterizations of basic connectives and quantifiers.

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, 287–319.

(2) A single-conclusion CR is a CR $\vdash$ such that $\Gamma \vdash \Delta$ only if Δ consists of a single formula.

The notion of a (multiple-conclusion) CR was introduced in [Scott, 1974] and [Scott, 1974b]. It was a generalization of Tarski's notion of a consequence relation, which was single-conclusion. Our notions are, however, not identical to the original ones of Tarski and Scott. First, they both considered *sets* (rather than multisets) of formulas. Second, they impose a third demand on CRs: monotonicity. We shall call a (single-conclusion or multiple-conclusion) CR which satisfies these two extra conditions *ordinary*. A single-conclusion, ordinary CR will be called *Tarskian*.[2]

The notion of a "logic" is in practice broader than that of a CR, since usually several CRs are associated with a given logic. Given a logic $\mathcal{L}$ there are in most cases two major single-conclusion CRs which are naturally associated with it: the external CR $\vdash^e_{\mathcal{L}}$ and the internal CR $\vdash^i_{\mathcal{L}}$. For example, if $\mathcal{L}$ is defined by some axiomatic system AS then $A_1, \cdots, A_n \vdash^e_{\mathcal{L}} B$ iff there exists a proof in AS of B from $A_1, \cdots A_n$ (according to the most standard meaning of this notion as defined in undergraduate textbooks on mathematical logic), while $A_1, \cdots, A_n \vdash^i_{\mathcal{L}} B$ iff $A_1 \rightarrow (A_2 \rightarrow \cdots \rightarrow (A_n \rightarrow B) \cdots)$ is a theorem of AS (where $\rightarrow$ is an appropriate "implication" connective of the logic). Similarly if $\mathcal{L}$ is defined using a Gentzen-type system G then $A_1, \cdots, A_n \vdash^i_{\mathcal{L}} B$ if the sequent $A_1, \cdots, A_n \Rightarrow B$ is provable in G, while $A_1, \cdots A_n \vdash^e_{\mathcal{L}} B$ iff there exists a proof in G of $\Rightarrow B$ from the assumptions $\Rightarrow A_1, \cdots, \Rightarrow A_n$ (perhaps with cuts). $\vdash^e_{\mathcal{L}}$ is always a Tarskian relation, $\vdash^i_{\mathcal{L}}$ frequently is not. The existence (again, in most cases) of these two CRs should be kept in mind in what follows. The reason is that semantic characterizations of connectives are almost always done w.r.t. Tarskian CRs (and so here $\vdash^e_{\mathcal{L}}$ is usually relevant). This is not the case with syntactical characterizations, and here frequently $\vdash^i_{\mathcal{L}}$ is more suitable.

2 THE SYNTACTICAL POINT OF VIEW

2.1 Classification of basic connectives

Our general framework allows us to give a completely abstract definition, *independent of any semantic interpretation*, of standard connectives. These characterizations explain why these connectives are so important in almost every logical system.

In what follows $\vdash$ is a fixed CR. All definitions are taken to be relative to $\vdash$ (the definitions are taken from [Avron, 1991a]).

[2] What we call a Tarskian CR is exactly Tarski's original notion. In [Avron, 1994] we argue at length why the notion of a proof in an axiomatic system naturally leads to *our* notion of single-conclusion CR, and why the further generalization to multiple-conclusion CR is also very reasonable.

We consider two types of connectives. The *internal* connectives, which make it possible to transform a given sequent into an equivalent one that has a special required form, and the *combining* connectives, which allow us to combine (under certain circumstances) two sequents into one which contains exactly the same information. The most common (and useful) among these are the following connectives:

Internal Disjunction: $+$ is an internal disjunction if for all Γ, Δ, A, B:

$$\Gamma \vdash \Delta, A, B \quad \text{iff} \quad \Gamma \vdash \Delta, A + B \ .$$

Internal Conjunction: $\otimes$ is an internal conjunction if for all Γ, Δ, A, B:

$$\Gamma, A, B \vdash \Delta \quad \text{iff} \quad \Gamma, A \otimes B \vdash \Delta \ .$$

Internal Implication: $\rightarrow$ is an internal implication if for all Γ, Δ, A, B:

$$\Gamma, A \vdash B, \Delta \quad \text{iff} \quad \Gamma \vdash A \rightarrow B, \Delta \ .$$

Internal Negation: $\neg$ is an internal negation if the following two conditions are satisfied by all Γ, Δ and A:

$$(1) \qquad A, \Gamma \vdash \Delta \quad \text{iff} \quad \Gamma \vdash \Delta, \neg A$$
$$(2) \qquad \Gamma \vdash \Delta, A \quad \text{iff} \quad \neg A, \Gamma \vdash \Delta \ .$$

Combining Conjunction: $\wedge$ is a combining conjunction iff for all Γ, Δ, A, B:

$$\Gamma \vdash \Delta, A \wedge B \quad \text{iff} \quad \Gamma \vdash \Delta, A \quad \text{and} \quad \Gamma \vdash \Delta, B \ .$$

Combining Disjunction: $\vee$ is a combining disjunction iff for all Γ, Δ, A, B

$$A \vee B, \Gamma \vdash \Delta \quad \text{iff} \quad A, \Gamma \vdash \Delta \quad \text{and} \quad B, \Gamma \vdash \Delta \ .$$

Note: The combining connectives are called "additives" in Linear logic (see [Girard, 1987]) and "extensional" in Relevance logic. The internal ones correspond, respectively, to the "multiplicative" and the "intensional" connectives.

Several well-known logics can be defined using the above connectives:

LL_m — **Multiplicative Linear Logic** (without the propositional constants): This is the logic which corresponds to the *minimal* (multiset) CR which includes all the internal connectives.

LL_{ma} — **Propositional Linear Logic** (without the "exponentials" and the propositional constants): This corresponds to the minimal consequence relation which contains all the connectives introduced above.

R_m — **the Intensional Fragment of the Relevance Logic** R**:**[3] This corresponds to the minimal CR which contains all the internal connectives and is *closed under contraction*.

[3] see [Anderson and Belnap, 1975] or [Dunn, 1986].

R **without Distribution:** This corresponds to the minimal CR which contains all the connectives which were described above and is closed under contraction.
RMI_m — **the Intensional Fragment of the Relevance Logic** RMI:[4] This corresponds to the minimal sets-CR which contains all the internal connectives.
Classical Proposition Logic: This of course corresponds to the minimal ordinary CR which has all the above connectives. Unlike the previous logics there is no difference in it between the combining connectives and the corresponding internal ones.

In all these examples we refer, of course, to the *internal* consequence relations which naturally correspond to these logics (In all of them it can be defined by either of the methods described above).

2.2 Internal Negation and Strong Symmetry

Among the various connectives defined above only negation essentially demands the use of multiple-conclusion CRs (even the existence of an internal disjunction does not *force* multiple-conclusions, although its existence is trivial otherwise.). Moreover, its existence creates full symmetry between the two sides of the turnstyle. Thus in its presence, closure under any of the structural rules on one side entails closure under the same rule on the other, the existence of any of the binary internal connectives defined above implies the existence of the rest, and the same is true for the combining connectives.

To sum up: internal negation is the connective with which "the hidden symmetries of logic" [Girard, 1987] are explicitly represented. We shall call, therefore, any multiple-conclusion CR which possesses it *strongly symmetric*.

Some alternative characterizations of an internal negation are given in the following easy proposition.

PROPOSITION 1. *The following conditions on* $\vdash$ *are all equivalent:*
(1) $\neg$ *is an internal negation for* $\vdash$.
(2) $\Gamma \vdash \Delta, A$ *iff* $\Gamma, \neg A \vdash \Delta$
(3) $A, \Gamma \vdash \Delta$ *iff* $\Gamma \vdash \Delta, \neg A$
(4) $A, \neg A \vdash$ *and* $\vdash \neg A, A$
(5) $\vdash$ *is closed under the rules:*

$$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \qquad \frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \ .$$

Our characterization of internal negation and of symmetry has been done within the framework of multiple-conclusion relations. Single-conclusion

[4] see [Avron, 1990a; Avron, 1990b].

CRs are, however, more natural. We proceed next to introduce corresponding notions for them.

DEFINITION.

(1) Let $\vdash_{\mathcal{L}}$ be a single-conclusion CR (in a language $\mathcal{L}$), and let $\neg$ be a unary connective of $\mathcal{L}$. $\vdash_{\mathcal{L}}$ is called *strongly symmetric* w.r.t. to $\neg$, and $\neg$ is called an *internal negation* for $\vdash_{\mathcal{L}}$, if there exists a multiple-conclusion CR $\vdash^*_{\mathcal{L}}$ with the following properties:

(i) $\Gamma \vdash^*_{\mathcal{L}} A$ iff $\Gamma \vdash_{\mathcal{L}} A$

(ii) $\neg$ is an internal negation for $\vdash^*_{\mathcal{L}}$

(2) A single-conclusion CR $\vdash_{\mathcal{L}}$ is called *essentially multiple-conclusion* iff it has an internal negation.

Obviously, if a CR $\vdash^*_{\mathcal{L}}$ like in the last definition exists then it is unique. We now formulate sufficient and necessary conditions for its existence.

THEOREM 2. *$\vdash_{\mathcal{L}}$ is strongly symmetric w.r.t. $\neg$ iff the following conditions are satisfied:*

(i) $A \vdash_{\mathcal{L}} \neg\neg A$

(ii) $\neg\neg A \vdash_{\mathcal{L}} A$

(iii) *If* $\Gamma, A \vdash_{\mathcal{L}} B$ *then* $\Gamma, \neg B \vdash_{\mathcal{L}} \neg A$.

Proof. The conditions are obviously necessary. Assume, for the converse, that $\vdash_{\mathcal{L}}$ satisfies the conditions. Define: $A_1, \cdots, A_n \vdash^s_{\mathcal{L}} B_1, \cdots, B_k$ iff for every $1 \leq i \leq n$ and $1 \leq j \leq k$:

$$A_1, \cdots, A_{i-1}, \neg B_1, \cdots, \neg B_k, A_{i+1}, \cdots, A_n \vdash \neg A_i$$
$$A_1, \cdots, A_n, \neg B_1, \cdots, \neg B_{j-1}, \neg B_{j+1}, \cdots, \neg B_k \vdash B_j \ .$$

It is easy to check that $\vdash^s_{\mathcal{L}}$ is a CR whenever $\vdash_{\mathcal{L}}$ is a CR (whether single-conclusion or multiple-conclusion), and that if $\Gamma \vdash^s_{\mathcal{L}} A$ then $\Gamma \vdash_{\mathcal{L}} A$. The first two conditions imply (together) that $\neg$ is an internal negation for $\vdash^s_{\mathcal{L}}$ (in particular: the second entails that if $A, \Gamma \vdash^s_{\mathcal{L}} \Delta$ then $\Gamma \vdash^s_{\mathcal{L}} \Delta, \neg A$ and the first that if $\Gamma \vdash^s_{\mathcal{L}} \Delta, A$ then $\neg A, \Gamma \vdash^s_{\mathcal{L}} \Delta$). Finally, the third condition entails that $\vdash^s_{\mathcal{L}}$ is conservative over $\vdash_{\mathcal{L}}$. ■

Examples of logics with an internal negation.

1. Classical logic.
2. Extensions of classical logic, like the various modal logics.
3. Linear logic and its various fragments.

4. The various Relevance logics (like R and RM (see [Anderson and Belnap, 1975; Dunn, 1986; Anderson and Belnap, 1992] or RMI [Avron, 1990a; Avron, 1990b]) and their fragments.

5. The various many-valued logics of Łukasiewicz, as well as Sobociński 3-valued logic [Sobociński, 1952].

Examples of logics without an internal negation.

1. Intuitionistic logic.

2. Kleene's 3-valued logic and its extension LPF [Jones, 1986].

Note: Again, in all these examples above it is the *internal* CR which is essentially multiple-conclusion (or not) and has an internal negation. This is true even for classical predicate calculus: There, e.g., $\forall x A(x)$ follows from $A(x)$ according to the *external* CR, but $\neg A(x)$ does not follow from $\neg\forall x A(x)$.[5]

All the positive examples above are instances of the following proposition, the easy proof of which we leave to the reader:

PROPOSITION 3. *Let $\mathcal{L}$ be any logic in a language containing $\neg$ and $\rightarrow$. Suppose that the set of valid formulae of $\mathcal{L}$ includes the set of formulae in the language of $\{\neg, \rightarrow\}$ which are theorems of Linear Logic,*[6] *and that it is closed under MP for $\rightarrow$. Then the internal consequence relation of $\mathcal{L}$ (defined using $\rightarrow$ as in the introduction) is strongly symmetric (with respect to $\neg$).*

The next two theorems discuss what properties of $\vdash_{\mathcal{L}}$ are preserved by $\vdash^s_{\mathcal{L}}$. The proofs are straightforward.

THEOREM 4. *Assume $\vdash_{\mathcal{L}}$ is essentially multiple-conclusion.*

1. *$\vdash^s_{\mathcal{L}}$ is monotonic iff so is $\vdash_{\mathcal{L}}$.*

2. *$\vdash^s_{\mathcal{L}}$ is closed under expansion (the converse of contraction) iff so is $\vdash_{\mathcal{L}}$.*

3. *$\wedge$ is a combining conjunction for $\vdash^s_{\mathcal{L}}$ iff it is a combining conjunction for $\vdash_{\mathcal{L}}$.*

4. *$\rightarrow$ is an internal implication for $\vdash^s_{\mathcal{L}}$ iff it is an internal implication for $\vdash_{\mathcal{L}}$.*

[5] The internal CR of classical logic has been called the "truth" CR in [Avron, 1991a] and was denoted there by $\vdash^t$, while the external one was called the "validity" CR and was denoted by $\vdash^v$. On the propositional level there is no difference between the two.

[6] Here $\neg$ should be translated into linear negation, $\rightarrow$ – into linear implication.

Notes:
1) Because $\vdash^s_{\mathcal{L}}$ is strongly symmetric, Parts (3) and (4) can be formulated as follows: $\vdash^s_{\mathcal{L}}$ has the internal connectives iff $\vdash_{\mathcal{L}}$ has an internal implication and it has the combining connectives iff $\vdash_{\mathcal{L}}$ has a combining conjunction.
2) In contrast, a combining disjunction for $\vdash_{\mathcal{L}}$ is not necessarily a combining disjunction for $\vdash^s_{\mathcal{L}}$. It is easy to see that a necessary and sufficient condition for this to happen is that $\vdash_{\mathcal{L}} \neg(A \vee B)$ whenever $\vdash_{\mathcal{L}} \neg A$ and $\vdash_{\mathcal{L}} \neg B$. An example of an essentially multiple-conclusion system with a combining disjunction which does not satisfy the above condition is RMI of [Avron, 1990a; Avron, 1990b]. That system indeed does not have a combining conjunction. This shows that a *single-conclusion* logic $\mathcal{L}$ with an internal negation and a combining disjunction does not necessarily have a combining conjunction (unless $\mathcal{L}$ is monotonic). The converse situation is not possible, though: If $\neg$ is an internal negation and $\wedge$ is a combining conjunction then $\neg(\neg A \wedge \neg B)$ defines a combining disjunction even in the single-conclusion case.
3) An internal conjunction $\otimes$ for $\vdash_{\mathcal{L}}$ is also not necessarily an internal conjunction for $\vdash^s_{\mathcal{L}}$. We need here the extra condition that if $A \vdash_{\mathcal{L}} \neg B$ then $\vdash_{\mathcal{L}} \neg(A \otimes B)$. An example which shows that this condition does not necessarily obtain even if $\vdash_{\mathcal{L}}$ is an ordinary CR, is given by the following CR $\vdash_{triv}$:

$$A_1, \cdots, A_n \vdash_{triv} B \quad \text{iff} \quad n \geq 1 \ .$$

It is obvious that $\vdash_{triv}$ is a Tarskian CR and that every unary connective of its language is an internal negation for it, while every binary connective is an internal conjunction. The condition above fails, however, for $\vdash_{triv}$.
4) The last example shows also that $\vdash^s_{\mathcal{L}}$ may not be closed under contraction when $\vdash_{\mathcal{L}}$ does, even if $\vdash_{\mathcal{L}}$ is Tarskian. Obviously, $\Gamma \vdash^s_{triv} \Delta$ iff $|\Gamma \cup \Delta| \geq 2$. Hence $\vdash^s_{triv} A, A$ but $\not\vdash^s_{triv} A$. The exact situation about contraction is given in the next proposition.

PROPOSITION 5. *If $\vdash_{\mathcal{L}}$ is essentially multiple-conclusion then $\vdash^s_{\mathcal{L}}$ is closed under contraction iff $\vdash_{\mathcal{L}}$ is closed under contraction and satisfies the following condition:*

$$\textit{If } A \vdash_{\mathcal{L}} B \textit{ and } \neg A \vdash_{\mathcal{L}} B \textit{ then } \vdash_{\mathcal{L}} B.$$

In case $\vdash_{\mathcal{L}}$ has a combining disjunction this is equivalent to:

$$\vdash_{\mathcal{L}} \neg A \vee A \ .$$

Proof. Suppose first that $\vdash_{\mathcal{L}}$ is closed under contraction and satisfies the condition. Assume that $\Gamma \vdash^s_{\mathcal{L}} \Delta, A, A$. If either Γ or Δ is not empty then this is equivalent to $\neg A, \neg A, \Gamma^* \vdash_{\mathcal{L}} B$ for some Γ^* and B. Since $\vdash_{\mathcal{L}}$ is closed under contraction, this implies that $\neg A, \Gamma^* \vdash_{\mathcal{L}} B$, and so $\Gamma \vdash^s_{\mathcal{L}} \Delta, A$. If both Γ and Δ are empty then we have $\neg A \vdash_{\mathcal{L}} A$. Since also $A \vdash_{\mathcal{L}} A$, the condition implies that $\vdash_{\mathcal{L}} A$, and so $\vdash^s_{\mathcal{L}} A$.

For the converse, suppose $\vdash^s_{\mathcal{L}}$ is closed under contraction. This obviously entails that so is also $\vdash_{\mathcal{L}}$. Assume now that $A \vdash_{\mathcal{L}} B$ and $\neg A \vdash_{\mathcal{L}} B$. Then $A \vdash^s_{\mathcal{L}} B$ and $\vdash^s_{\mathcal{L}} B, A$. Applying cut we get that $\vdash^s_{\mathcal{L}} B, B$, and so $\vdash^s_{\mathcal{L}} B$. It follows that $\vdash_{\mathcal{L}} B$. ■

3 THE SEMANTIC POINT OF VIEW

We turn in this section to the semantic aspect of negation.

3.1 The General Framework

A "semantics" for a logic consists of a set of "models". The main property of a model is that every sentence of a logic is either true in it or not (and not both). The logic is sound with respect to the semantics if the set of sentences which are true in each model is closed under the CR of the logic, and complete if a sentence φ follows (according to the logic) from a set T of assumptions iff every model of T is a model of φ. Such a characterization is, of course, possible only if the CR we consider is Tarskian. *In this section we assume, therefore, that we deal only with Tarskian CRs.* For logics like Linear Logic and Relevance logics this means that we consider only the *external* CRs which are associated with them (see the Introduction).

Obviously, the essence of a "model" is given by the set of sentences which are true in it. Hence a semantics is, essentially, just a set S of theories. Intuitively, these are the theories which (according to the semantics) provide a full description of a possible state of affairs. Every other theory can be understood as a partial description of such a state, or as an approximation of a full description. Completeness means, then, that a sentence φ follows from a theory T iff φ belongs to every superset of T which is in S (in other words: iff φ is true in any possible state of affairs of which T is an approximation).

Now what constitutes a "model" is frequently defined using some kind of algebraic structures. Which kind (matrices with designated values, possible worlds semantics and so on) varies from one logic to another. It is difficult, therefore, to base a general, uniform theory on the use of such structures. Semantics (= a set of theories!) can also be defined, however, purely syntactically. Indeed, below we introduce several types of syntactically defined semantics which are very natural for *every* logic with "negation". Our investigations will be based on these types.

Our description of the notion of a model reveals that *externally* it is based on two classical "laws of thought": the law of contradiction and the law of excluded middle. When this external point of view is reflected inside the logic with the help of a unary connective $\neg$ we call this connective a (strong) *semantic negation*. Its intended meaning is that $\neg A$ should be true precisely

when A is not. The law of contradiction means then that only consistent theories may have a model, while the law of excluded middle means that the set of sentences which are true in some given model should be negation-complete. The sets of consistent theories, of complete theories and of normal theories (theories that are both) have, therefore a crucial importance when we want to find out to what degree a given unary connective of a logic can be taken as a semantic negation. Thus complete theories reflect a state of affairs in which the law of excluded middle holds. It is reasonable, therefore, to say that this law semantically obtains for a logic $\mathcal{L}$ if its consequence relation $\vdash_{\mathcal{L}}$ is *determined* by its set of complete theories. Similarly, $\mathcal{L}$ (strongly) satisfies the law of contradiction iff $\vdash_{\mathcal{L}}$ is determined by its set of consistent theories, and it semantically satisfies both laws iff $\vdash_{\mathcal{L}}$ is determined by its set of normal theories.

The above characterizations might seem unjustifiably strong for logics which are designed to allow non-trivial inconsistent theories. For such logics the demand that $\vdash_{\mathcal{L}}$ should be determined by its set of normal theories is reasonable only if we start with a consistent set of assumptions (this is called strong c-normality below). A still weaker demand (c-normality) is that any consistent set of assumptions should be an approximation of at least one normal state of affairs (in other words: it should have at least one normal extension).

It is important to note that the above characterizations are independent of the existence of any internal reflection of the laws (for example: in the forms $\neg(\neg A \wedge A)$ and $\neg A \vee A$, for suitable $\wedge$ and $\vee$). There might be strong connections, of course, in many important cases, but they are neither necessary nor always simple.

We next define our general notion of semantics in precise terms.

DEFINITION. Let $\mathcal{L}$ be a logic in L and let $\vdash_{\mathcal{L}}$ be its associated CR.

1. A *setup* for $\vdash_{\mathcal{L}}$ is a set of formulae in L which is closed under $\vdash_{\mathcal{L}}$. A *semantics* for $\vdash_{\mathcal{L}}$ is a nonempty set of setups which does not include the trivial setup (i.e., the set of all formulae).

2. Let S be a semantics for $\vdash_{\mathcal{L}}$. An *S-model* for a formula A is any setup in S to which A belongs. An S-model of a theory T is any setup in S which is a superset of T. A formula is called S-valid iff every setup in S is a model of it. A formula A S-follows from a theory T ($T \vdash^S_{\mathcal{L}} A$) iff every S-model of T is an S-model of A.

PROPOSITION 6. *$\vdash^S_{\mathcal{L}}$ is a (Tarskian) consequence relation and $\vdash_{\mathcal{L}} \subseteq \vdash^S_{\mathcal{L}}$.*

Notes:

1. $\vdash^S_{\mathcal{L}}$ is not necessarily finitary even if $\vdash$ is.

2. $\vdash_{\mathcal{L}}$ is just $\vdash^{S(\mathcal{L})}_{\mathcal{L}}$ where $S(\mathcal{L})$ is the set of all setups for $\vdash_{\mathcal{L}}$.

3. If $S_1 \subseteq S_2$ then $\vdash_{\mathcal{L}}^{S_2} \subseteq \vdash_{\mathcal{L}}^{S_1}$.

EXAMPLES.

1. For classical propositional logic the standard semantics consists of the setups which are induced by some valuation in $\{t, f\}$. These setups can be characterized as theories T such that

   (i) $\neg A \in T$ iff $A \notin T$ (ii) $A \wedge B \in T$ iff both $A \in T$ and $B \in T$

   (and similar conditions for the other connectives).

2. In classical predicate logic we can define a setup in S to be any set of formulae which consists of the formulae which are true in some given first-order structure relative to some given assignment. Alternatively we can take a setup to consist of the formulae which are *valid* in some given first-order structure. In the first case $\vdash^S = \vdash^t$, in the second $\vdash^S = \vdash^v$, where $\vdash^t$ and $\vdash^v$ are the "truth" and "validity" consequence relations of classical logic (see [Avron, 1991a] for more details).

3. In modal logics we can define a "model" as the set of all the formulae which are true in some world in some Kripke frame according to some valuation. Alternatively, we can take a model as the set of all formulae which are valid in some Kripke frame, relative to some valuation. Again we get the two most usual consequence relations which are used in modal logics (see [Avron, 1991a] or [Fagin *et al.*, 1992]).

From now on the *following two conditions will be assumed in all our general definitions and propositions:*

1. The language contains a negation connective $\neg$.

2. For no A are both A and $\neg A$ theorems of the logic.

DEFINITION. Let S be a semantics for a CR $\vdash_{\mathcal{L}}$

1. $\vdash_{\mathcal{L}}$ is strongly complete relative to S if $\vdash_{\mathcal{L}}^{S} = \vdash_{\mathcal{L}}$.

2. $\vdash_{\mathcal{L}}$ is weakly complete relative to S if for all A, $\vdash_{\mathcal{L}} A$ iff $\vdash_{\mathcal{L}}^{S} A$.

3. $\vdash_{\mathcal{L}}$ is c-complete relative to S if every consistent theory of $\vdash_{\mathcal{L}}$ has a model in S.

4. $\vdash_{\mathcal{L}}$ is strongly c-complete relative to S if for every A and every *consistent* T, $T \vdash_{\mathcal{L}}^{S} A$ iff $T \vdash_{\mathcal{L}} A$.

Notes:

1. Obviously, strong completeness implies strong c-completeness, while strong c-completeness implies both c-completeness and weak completeness.

2. Strong completeness means that deducibility in $\vdash_{\mathcal{L}}$ is equivalent to semantic consequence in S. Weak completeness means that theoremhood in $\vdash_{\mathcal{L}}$ (i.e., derivability from the empty set of assumptions) is equivalent to semantic validity (= truth in all models). c-completeness means that consistency implies satisfiability. It becomes identity if only consistent sets can be satisfiable, i.e., if $\{\neg A, A\}$ has a model for no A. This is obviously too strong a demand for paraconsistent logics. Finally, strong c-completeness means that if we restrict ourselves to *normal* situations (i.e., consistent theories) then $\vdash_{\mathcal{L}}$ and $\vdash^{S}_{\mathcal{L}}$ are the same. This might sometimes be weaker than full strong completeness.

The last definition uses the concepts of "consistent" theory. The next definition clarifies (among other things) the meaning of this notion as we are going to use in it this paper.

DEFINITION. Let $\mathcal{L}$ and $\vdash_{\mathcal{L}}$ be as above. A theory in L *consistent* if for no A it is the case that $T \vdash_{\mathcal{L}} A$ and $T \vdash_{\mathcal{L}} \neg A$, *complete* if for all A, either $T \vdash_{\mathcal{L}} A$ or $T \vdash_{\mathcal{L}} \neg A$, *normal* if it is both consistent and complete. $CS_{\mathcal{L}}$, $CP_{\mathcal{L}}$ and $N_{\mathcal{L}}$ will denote, respectively, the sets of its consistent, complete and normal theories.

Given $\vdash_{\mathcal{L}}$, the three classes, $CS_{\mathcal{L}}$, $CP_{\mathcal{L}}$ and $N_{\mathcal{L}}$, provide 3 different syntactically defined semantics for $\vdash_{\mathcal{L}}$, and 3 corresponding consequence relations $\vdash^{CS_{\mathcal{L}}}_{\mathcal{L}}$, $\vdash^{CP_{\mathcal{L}}}_{\mathcal{L}}$ and $\vdash^{N_{\mathcal{L}}}_{\mathcal{L}}$. We shall henceforth denote these CRs by $\vdash^{CS}_{\mathcal{L}}$, $\vdash^{CP}_{\mathcal{L}}$ and $\vdash^{N}_{\mathcal{L}}$, respectively. Obviously, $\vdash^{CS}_{\mathcal{L}} \subseteq \vdash^{N}_{\mathcal{L}}$ and $\vdash^{CP}_{\mathcal{L}} \subseteq \vdash^{N}_{\mathcal{L}}$. In the rest of this section we investigate these relations and the completeness properties they induce.

Let us start with the easier case: that of $\vdash^{CS}_{\mathcal{L}}$. It immediately follows from the definitions (and our assumptions) that relative to it every logic is strongly c-complete (and so also c-complete and weakly complete). Hence the only completeness notion it induces is the following:

DEFINITION. A logic $\mathcal{L}$ with a consequence relation $\vdash_{\mathcal{L}}$ is strongly consistent if $\vdash^{CS}_{\mathcal{L}} = \vdash_{\mathcal{L}}$.

$\vdash^{CS}_{\mathcal{L}}$ is not a really interesting CR. As the next theorem shows, what it does is just to trivialize inconsistent $\vdash_{\mathcal{L}}$-theories. Strong consistency, accordingly, might not be a desirable property, certainly not a property that any logic with negation should have.

PROPOSITION 7.

1. *$T \vdash^{CS}_{\mathcal{L}} A$ iff either T is inconsistent in $\mathcal{L}$ or $T \vdash_{\mathcal{L}} A$. In particular, T is $\vdash^{CS}_{\mathcal{L}}$-consistent iff it is $\vdash_{\mathcal{L}}$-consistent.*

2. *$\mathcal{L}$ is strongly consistent iff $\neg A, A \vdash_{\mathcal{L}} B$ for all A, B (iff T is consistent whenever $T \nvdash A$).*

3. *Let $\mathcal{L}^{CS}$ be obtained from $\mathcal{L}$ by adding the rule: from $\neg A$ and A infer B. Then $\vdash^{CS}_{\mathcal{L}} = \vdash_{\mathcal{L}^{CS}}$. In particular: if $\vdash_{\mathcal{L}}$ is finitary then so is $\vdash^{CS}_{\mathcal{L}}$.*

4. *$\vdash^{CS}_{\mathcal{L}}$ is strongly consistent.*

We turn now to $\vdash^{CP}$ and $\vdash^{N}$. In principle, each provides 4 notions of completeness. We don't believe, however, that considering the two notions of c-consistency is natural or interesting in the framework of $\vdash^{CP}$ (c-completeness, e.g., means there that every consistent theory has a complete extension, but that extension might not be consistent itself). Accordingly we shall deal with the following 6 notions of syntactical completeness.[7]

DEFINITION. Let $\mathcal{L}$ be a logic and let $\vdash_{\mathcal{L}}$ be its consequence relation.

1. $\mathcal{L}$ is *strongly complete* if it is strongly complete relative to CP.

2. $\mathcal{L}$ is *weakly complete* if it is weakly complete relative to CP.

3. $\mathcal{L}$ is *strongly normal* if it is strongly complete relative to N.

4. $\mathcal{L}$ is *weakly normal* if it is weakly complete relative to N.

5. $\mathcal{L}$ is *c-normal* if it is c-complete relative to N.

6. $\mathcal{L}$ is *strongly c-normal* if it is strongly c-complete relative to N (this is easily seen to be equivalent to $\vdash^{N}_{\mathcal{L}} = \vdash^{CS}_{\mathcal{L}}$).

For the reader's convenience we repeat what these definitions actually mean:

1. $\mathcal{L}$ is strongly complete iff whenever $T \nvdash_{\mathcal{L}} A$ there exists a complete extension T^* of T such that $T^* \nvdash_{\mathcal{L}} A$.

2. $\mathcal{L}$ is weakly complete iff whenever A is not a theorem of $\mathcal{L}$ there exists a complete T^* such that $T^* \nvdash_{\mathcal{L}} A$.

3. $\mathcal{L}$ is strongly normal iff whenever $T \nvdash_{\mathcal{L}} A$ there exists a complete and consistent extension T^* of T such that $T^* \nvdash_{\mathcal{L}} A$.

4. $\mathcal{L}$ is weakly normal iff whenever A is not a theorem of $\mathcal{L}$ there exists a complete and consistent theory T^* such that $T^* \nvdash_{\mathcal{L}} A$.

[7] In [Anderson and Belnap, 1975] the term "syntactically complete" was used for what we call below "strongly c-normal".

5. $\mathcal{L}$ is *c*-normal if every consistent theory of $\mathcal{L}$ has a complete and consistent extension.

6. $\mathcal{L}$ is strongly *c*-normal iff whenever T is consistent and $T \nvdash_{\mathcal{L}} A$ there exists a complete and consistent extension T^* of T such that $T^* \nvdash_{\mathcal{L}} A$.

Our next proposition provides simpler syntactical characterizations of some of these notions in case $\vdash_{\mathcal{L}}$ is finitary.

PROPOSITION 8. *Assume that $\vdash_{\mathcal{L}}$ is finitary.*

1. *$\mathcal{L}$ is strongly complete iff for all T, A and B:*

$$(*) \quad T, A \vdash_{\mathcal{L}} B \quad \textit{and} \quad T, \neg A \vdash_{\mathcal{L}} B \quad \textit{imply} \quad T \vdash_{\mathcal{L}} B$$

In case $\mathcal{L}$ has a combining disjunction $\vee$ then $(*)$ is equivalent to the theoremhood of $\neg A \vee A$ (excluded middle).

2. *$\mathcal{L}$ is strongly normal if for all T and A:*

$$(**) \quad T \vdash_{\mathcal{L}} A \quad \textit{iff} \quad T \cup \{\neg A\} \quad \textit{is inconsistent.}$$

3. *$\mathcal{L}$ is strongly c-normal iff $(**)$ obtains for every* consistent *T.*

4. *$\mathcal{L}$ is c-normal iff for every consistent T and every A either $T \cup \{A\}$ or $T \cup \{\neg A\}$ is consistent.*

Proof. Obviously, strong completeness implies $(*)$. For the converse, assume that $T \nvdash B$. Using $(*)$, we extend T in stages to a complete theory such that $T^* \nvdash B$. This proves part 1. The other parts are straightforward. ■

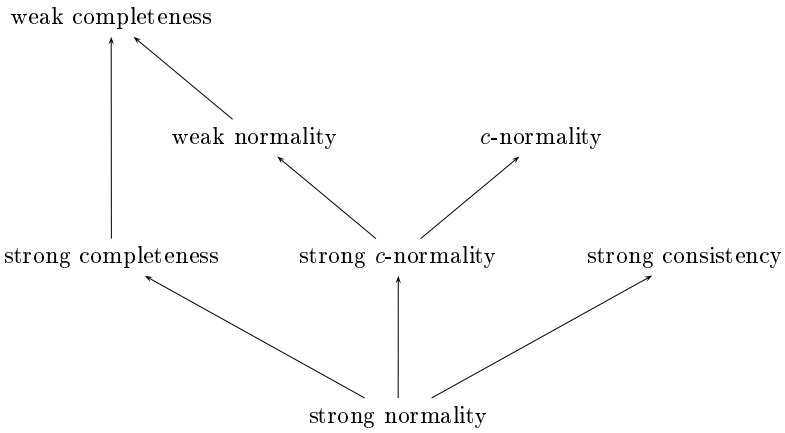
COROLLARIES.

1. If $\mathcal{L}$ is strongly normal then it is strongly symmetric w.r.t. $\neg$. Moreover: $\vdash^s_{\mathcal{L}}$ is an ordinary multiple-conclsion CR.

2. If $\mathcal{L}$ is strongly symmetric w.r.t. $\neg$ then it is strongly complete iff $\vdash^s_{\mathcal{L}}$ is closed under contraction.

Proof. These results easily follows from the last proposition and Theorems 2, 4 and 5 above. ■

In the figure below we display the obvious relations between the seven properties of logics which were introduced here (where an arrow means "contained in"). The next theorem shows that no arrow can be added to it:



THEOREM 9. *A logic can be:*

1. *strongly consistent and c-normal without even being weakly complete*
2. *strongly complete and strongly c-normal without being strongly consistent (and so without being strongly normal)*
3. *strongly consistent without being c-normal*
4. *strongly complete, weakly normal and c-normal without being strongly c-normal*
5. *strongly complete and c-normal without being weakly normal*
6. *strongly consistent, c-normal and weakly normal without being strongly c-normal (=strongly normal in this case, because of strong consistency)*
7. *strongly complete without being c-normal.*[8]

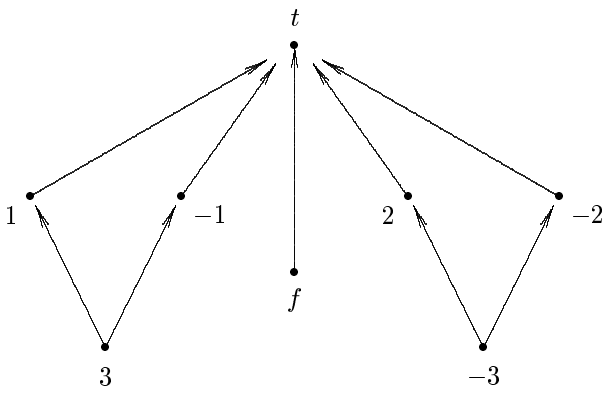
Proof. Appropriate examples for 1-6 are given below, respectively, in theorems 12, 18, 33, 19, 35 and the corollary to theorem 19. As for the last part, let $\mathcal{L}$ be the following system in the language of $\{\neg, \rightarrow\}$:[9]

[8] Hence the two standard formulations of the "strong consistency" of classical logic are *not* equivalent in general.

[9] Classical logic is obtained from it by adding $\neg A \rightarrow (A \rightarrow B)$ as axiom (see [Epstein, 1995, Ch. 2L].

$Ax1$: $A \to (B \to A)$
$Ax2$: $A \to (B \to C) \to (A \to B) \to (A \to C)$
$Ax3$: $(\neg A \to B) \to ((A \to B) \to B)$
(MP) $\dfrac{A \quad A \to B}{B}$.

Obviously, the deduction theorem for $\to$ holds for this system, since MP is the only rule of inference, and we have $Ax1$ and $Ax2$. This fact, $Ax3$ and proposition 8 guarantee that it is strongly complete. To show that it is not c-normal, we consider the theory $T_0 = \{p \to q,\ p \to \neg q,\ \neg p \to r,\ \neg p \to \neg r\}$. Obviously, T_0 has no complete and consistent extension. We show that it is consistent nevertheless. For this we use the following structure:



Define in this structure $a \to b$ as t if $a \leq b$, b otherwise, $\neg x$ as f if $x = t$, t if $x = f$ and $-x$ otherwise. It is not difficult now to show that if $T \vdash A$ in the present logic for some T and A, and v is a valuation in this structure such that $v(B) = t$ for all $B \in T$, then $v(A) = t$. Take now $v(p) = 3$, $v(q) = 1$, $v(r) = 2$. Then $v(B) = t$ for all $B \in T_0$, but obviously there is no A such that $v(A) = v(\neg A) = t$. Hence T_0 is consistent. ■

We end this introductory subsection with a characterization of $\vdash_{\mathcal{L}}^{CP}$ and $\vdash_{\mathcal{L}}^{N}$. The proofs are left to the reader.

PROPOSITION 10.

1. *$\vdash_{\mathcal{L}}^{CP}$ is strongly complete, and is contained in any strongly complete extension of $\vdash_{\mathcal{L}}$.*

2. *Suppose $\vdash_{\mathcal{L}}$ is finitary. $T \vdash_{\mathcal{L}}^{CP} A$ iff for some $B_1, \ldots, B_n$ $(n \geq 0)$ we have that $T \cup \{B_1^*, \ldots, B_n^*\} \vdash_{\mathcal{L}} A$ for every set $\{B_1^*, \ldots, B_n^*\}$ such that for all i, $B_i^* = B_i$ or $B_i^* = \neg B_i$.*

3. *If $\vdash_{\mathcal{L}}$ is finitary, then so is $\vdash_{\mathcal{L}}^{CP}$.*

PROPOSITION 11.

1. *$\vdash_{\mathcal{L}}^{N}$ is strongly normal, and is contained in every strongly normal extension of $\vdash_{\mathcal{L}}$.*
2. *If $\vdash_{\mathcal{L}}$ is finitary then $T \vdash_{\mathcal{L}}^{N} A$ iff for some $B_1, \ldots, B_n$ we have that for all $\{B_1^*, \ldots, B_n^*\}$ where $B_i^* \in \{B_i, \neg B_i\}$ $(i = 1, \ldots, n)$, either $T \cup \{B_1^*, \ldots, B_n^*\}$ is inconsistent or $T \cup \{B_1^*, \ldots, B_n^*\} \vdash_{\mathcal{L}} A$*
3. *$\vdash_{\mathcal{L}}^{N}$ is finitary if $\vdash_{\mathcal{L}}$ is.*

3.2 Classical and Intuitionistic Logics

Obviously, classical propositional logic is strongly normal. In fact, most of the proofs of the completeness of classical logic relative to its standard two-valued semantics begin with demonstrating the condition $(**)$ in Proposition 8, and are based on the fact that every complete and consistent theory determines a unique valuation in $\{t, f\}$ - and vice versa. In other words: N here is exactly the usual semantics of classical logic, only it can be characterized also using an especially simple algebraic structure (and valuations in it). One can argue that this strong normality *characterizes* classical logic. To be specific, it is not difficult to show the following claims:

1. classical logic is the only logic in the language of $\{\neg, \wedge\}$which is strongly normal w.r.t. $\neg$ and for which $\wedge$ is an internal conjunction. Similar claims hold for the $\{\neg, \rightarrow\}$ language, if we demand $\rightarrow$ to be an internal implication and for the $\{\neg, \vee\}$ language, if we demand $\vee$ to be a combining disjunction.
2. Any logic which is strongly normal and has either an internal implication, or an internal conjunction or a combining disjunction contains classical propositional logic.

The next proposition summarizes the relevant facts concerning intuitionistic logic. The obvious conclusion is that although the official intuitionistic negation has some features of negation, it still lacks most. Hence, it cannot be taken as a real negation from our semantic point of view.

PROPOSITION 12. *Intuitionistic logic is strongly consistent and c-normal, but it is not even weakly complete.*

Proof. Strong consistency follows from part 3 of Proposition 7. c-normality follows from part 4 of Proposition 8, since in intuitionistic logic if both $T \cup \{A\}$ and $T \cup \{\neg A\}$ are inconsistent then $T \vdash_H \neg A$ and $T \vdash_H \neg\neg A$, and so T is inconsistent. Finally, $\neg A \vee A$ belongs to every complete setup, but is not intuitionistically valid. ■

Note: Intuitionistic logic and classical logic have exactly the same consistent and complete setups, since any complete intuitionistic theory is closed under the elimination rule of double negation. Hence any consistent intuitionistic theory has a classical two-valued model.

What about fragments (with negation) of Intuitionistic Logic? Well, they are also strongly consistent and c-normal, by the same proof. Moreover, $((A \to B) \to A) \to A$ is another example of a sentence which belongs to every complete setup (since $A \vdash_H ((A \to B) \to A) \to A$ and $\neg A \vdash_H ((A \to B) \to A) \to A$), but is not provable. The set of theorems of the pure $\{\neg, \wedge\}$ fragment, on the other hand, is identical to that of classical logic, as is well known. This fragment is, therefore, easily seen to be weakly normal. It is still neither strongly complete nor strongly c-normal, since $\neg\neg A \vdash_H^{CP} A$. ■

Finally, we note the important fact that classical logic can be viewed as the completion of intuitionistic logic. More precisely:

PROPOSITION 13.

1. $\vdash_H^{CS} = \vdash_H$
2. $\vdash_H^{CP} = \vdash_H^N =$ *classical logic.*

Proof.

2. $\vdash_L^{CP} = \vdash_L^N$ whenever L is strongly consistent (i.e., all nontrivial theories are consistent). In the proof of the previous proposition we have seen also that $\vdash_H^{CP} \neg A \vee A$ and $\vdash_H^{CP} ((A \to B) \to A) \to A$. It is well known, however, that by adding either of this schemes to intuitionistic logic we get classical logic. Hence classical logic is contained in $\vdash_H^{CP}$. Since classical logic is already strongly complete, $\vdash_H^{CP}$ is exactly classical logic. (Note that this is true for any fragment of the language which includes negation.) ■

3.3 Linear Logic (LL)

In the next 3 subsections we are going to investigate some known substructural logics [Schroeder-Heister and Došen, 1993]. Before doing it we must emphasize again that in this section it is only the external, Tarskian consequence relation of these logics which can be relevant. This consequence relation can very naturally be defined by using the standard Hilbert-type formulations of these logics: $A_1, \ldots, A_n \vdash_{\mathcal{L}}^e B$ ($\mathcal{L} = LL, R, RM, RMI$, etc.) iff there exists an ordinary deduction of B from $A_1, \ldots, A_n$ in the corresponding Hilbert-type system. This definition is insensitive to the exact choice of axioms (or even rules), provided we take all the rules as rules of derivation and not just as rules of proof. In the case of Linear Logic one can use for this the systems given in [Avron, 1988] or in [Troelstra, 1992]. An

alternative equivalent definition of the various external CRs can be given using the standard Gentzen-types systems for these logics (in case such exist), as explained in the introduction. Still another characterization in the case of Linear Logic can be given using the phase semantics of [Girard, 1987]: $A_1, \ldots, A_n \vdash^e_{LL} B$ iff B is true in every phase model of $A_1, \ldots, A_n$. In what follows we shall omit the superscript "e" and write just $\vdash_{LL}$, $\vdash_{LL_m}$, etc.

Unlike in [Girard, 1987] we shall take below negation as one of the connectives of the language of linear logic and write $\neg A$ for the negation of A (this corresponds to Girard's $A^{\perp}$). As in [Avron, 1988] and in the relevance logic literature, we use arrow ($\rightarrow$) for linear implication.

We show now that linear logic is incomplete with respect to our various notions.

PROPOSITION 14. *LL_m (LL_{ma}, LL) is not strongly consistent.*

PROPOSITION 15. *LL_m (LL_{ma}, LL) is neither strongly complete nor c-normal.*

Proof. Consider the following theory:

$$T = \{p \rightarrow \neg p\,,\ \neg p \rightarrow p\}\ .$$

From the characterization of $\vdash_{LL_m}$ given in [Avron, 1992] it easily follows that has T been inconsistent then there would be a provable sequent of the form: $\neg p \rightarrow p, \neg p \rightarrow p, \ldots, \neg p \rightarrow p, p \rightarrow \neg p, \ldots, p \rightarrow \neg p \Rightarrow$. But in any cut-free proof of such a sequent the premises of the last applied rule should have an odd number of occurrences of p, which is impossible in a provable sequent of the purely multiplicative linear logic. Hence T is consistent. Obviously, every complete extension of T proves p and $\neg p$ and so is inconsistent. This shows that LL_m is not c-normal. It also shows that p is not provable from T, although it is provable from any complete extension of it, and so LL_m is not strongly complete. ■

PROPOSITION 16. *LL_{ma} (and so also LL) is not weakly complete.*

Proof. $\sim A \oplus A$ is not a theorem of linear logic, but it belongs to any complete theory. ■

It follows that Linear logic (and its multiplicative-additive fragment) has none of the properties we have defined in this section. Its negation is therefore not really a negation from our present *semantic* point of view.

Our results still leave the possibility that LL_m might be weakly complete or even weakly normal. We conjecture that it is not, but we have no counterexample.

We end this section by giving axiomatizations of $\vdash^{CP}_{LL}$ and $\vdash^{N}_{LL}$.

PROPOSITION 17.

1. *Let LL^{CP} be the full Hilbert-type system for linear logic (as given in* [Avron, 1988]*) together with the rule: from $!A \to B$ and $!\neg A \to B$ infer B. Then $\vdash^{CP}_{LL} = \vdash_{LL^{CP}}$.*

2. *Let LL^{N} be LL^{CP} together with the disjunctive syllogism for $\oplus$ (from $\neg A$ and $A \oplus B$ infer B). Then $\vdash^{N}_{LL} = \vdash_{LL^{N}}$.*

Proof.

1. The necessitation rule (from A infer $!A$) is one of the rules of LL.[10] It follows therefore that B should belong to any complete setup which contains both $!A \to B$ and $!\neg A \to B$. Hence the new rule is valid for $\vdash^{CP}_{LL}$ and $\vdash_{LL^{CP}} \subseteq \vdash^{CP}_{LL}$.

   For the converse, assume $T \vdash^{CP}_{LL} A$. Then there exist $B_1, \ldots, B_n$ like in proposition 10(2). We prove by induction on n that $T \vdash_{LL^{CP}} A$. The case $n = 0$ is obvious. Suppose the claim is true for $n-1$. We show it for n. By the deduction theorem for LL, $!B_1^*, \ldots, !B_n^* \Rightarrow A$ is derivable from T in LL^{CP}.[11] More precisely: $!B_1^* \otimes !B_2^* \ldots \otimes !B_n^* \to A$ is derivable from T for any choice of $B_1^*, \ldots, B_n^*$. Since $!C \otimes !D \leftrightarrow !(C \& D)$ is a theorem of LL, this means that both $!B_n \to (!(B_1^* \& \ldots \& B_{n-1}^*) \to A)$ and $!\neg B_n \to (!(B_1^* \& \ldots \& B_{n-1}^*) \to A)$. By the new rule of LL^{CP} we get therefore that $T \vdash_{LL^{CP}} !(B_1^* \& \ldots \& B_{n-1}^*) \to A$, and so $T \vdash_{LL^{CP}} !B_1^* \otimes !B_2^* \otimes \ldots \otimes !B_{n-1}^* \to A$ for all choices of $B_1^*, \ldots, B_{n-1}^*$. An application of the induction hypothesis gives $T \vdash_{LL^{CP}} A$.

2. The proof is similar, only this time we should have (by proposition 11) that $T \cup \{B_1^*, \ldots, B_n^*\}$ is either inconsistent in L^N or proves A there. In both cases it proves $A \oplus \bot$ in LL^{CP}. The same argument as before will show that $T \vdash_{LL^{CP}} A \oplus \bot$. Since $\vdash_{LL} \neg \bot$, one application of the disjunctive syllogism will give $T \vdash_{LL^{CP}} A$. It remains to show that the disjunctive syllogism is valid for $\vdash^{N}_{LL}$. This is easy, since $\{\neg A, A \oplus B, \neg B\}$ is inconsistent in LL, and so any complete and consistent extension of $\{\neg A, A \oplus B\}$ necessarily contains B. ■

3.4 The Standard Relevance Logic R and its Relatives

In this section we investigate the standard relevance logic R of Anderson and Belnap [Anderson and Belnap, 1975; Dunn, 1986] and its various extensions and fragments. Before doing this we should again remind the reader what consequence relation we have in mind: the ordinary one which is associated

[10] Note again that we are talking here about $\vdash^{e}_{LL}$!

[11] In fact, at the beginning it is derivable from T in LL, but for the induction to go through we need to assume derivability in LL^{CP} at each step.

with the standard Hilbert-type formulations of these logics. As in the case of linear logic, this means that we take both rules of R (MP and adjunction) as rules of derivation and define $T \vdash_R A$ in the most straightforward way.

Let us begin with the purely intensional (=multiplicative) fragment of R: R_m. We state the results for this system, but they hold for all its nonclassical various extensions (by axioms) which are discussed in the literature.

THEOREM 18. *R_m is not strongly consistent, but it is strongly complete and strongly c-normal.*

Proof. It is well-known that R_m is not strongly consistent in our sense. Its main property that we need for the other claims is that $T, A \vdash_{R_m} B$ iff either $T \vdash_{R_m} B$ or $T \vdash_{R_m} A \to B$. The strong completeness of R_m follows from this property by the provability of $(\neg A \to B) \to ((A \to B) \to B)$ and proposition 8(1).

To show strong c-normality, we note first that a theory T is inconsistent in R_m iff $T \vdash_{R_m} \neg(B \to B)$ for some B (because $\vdash_{R_m} \neg B \to (B \to \neg(B \to B))$). Suppose now that T is consistent and $T \nvdash_{R_m} A$. Were $T \cup \{\neg A\}$ inconsistent then by the same main property and the consistency of T we would have that $T \vdash_{R_m} \neg A \to \neg(B \to B)$ for some B, and so that $T \vdash_{R_m} (B \to B) \to A$ and $T \vdash_{R_m} A$. A contradiction. Hence $T \cup \{\neg A\}$ is consistent and we are done by proposition 8(3). ■

The last theorem is the optimal theorem concerning negation that one can expect from a logic which was designed to be paraconsistent. It shows that with respect to normal "situations" (i.e., consistent theories) the negation connective of R_m behaves exactly as in classical logic. The difference, therefore, is mainly w.r.t. inconsistent theories. Unlike classical logic they are not necessarily trivial in R_m. Strong completeness means, though, that excluded middle, at least, can be assumed even in the abnormal situations.

When we come to R as a whole the situation is not as good as for the purely intensional fragments. Strong c-normality is lost. What we do have is the following:

THEOREM 19. *R is strongly complete, c-normal and weakly normal,* [12] *but it is neither strongly consistent nor strongly c-normal.*

Proof. Obviously, R is not strongly consistent. It is also well known that $\neg p, p \vee q \nvdash_R q$. Still q belongs to any complete and consistent extension of the (even classically!) consistent theory $\{\neg p, p \vee q\}$, since $\{\neg p, p \vee q, \neg q\}$ is not consistent in R. It follows that R is not strongly c-normal. On the other hand, to any extension $\mathcal{L}$ of R by axiom schemes it is true that if $T, A \vdash_{\mathcal{L}} C$ and $T, B \vdash_{\mathcal{L}} C$, then $T, A \vee B \vdash_{\mathcal{L}} C$ [Anderson and Belnap, 1975]. Since $\vdash_R A \vee \neg A$, this and proposition 8(1) entail that any such extension is strongly complete. Suppose, next, that T is theory and A a

[12] Weak normality is proved in [Anderson and Belnap, 1975] under the name "syntactical completeness".

formula such that $T \cup \{A\}$ and $T \cup \{\neg A\}$ are inconsistent ($\mathcal{L}$ as above). Then for some B and C it is the case that $T, A \vdash_{\mathcal{L}} \neg B \wedge B$ and $T, \neg A \vdash_{\mathcal{L}} \neg C \wedge C$. It follows that $T, A \vee \neg A \vdash_{\mathcal{L}} (\neg B \wedge B) \vee (\neg C \wedge C)$. Since $A \vee \neg A$ and $\neg[(\neg B \wedge B) \vee (\neg C \wedge C)]$ are both theorems of R, T is inconsistent in $\mathcal{L}$. By proposition 8(4) this shows that any such logic is c-normal. Suppose, finally, that $\nvdash_R A$. Had $\{\neg A\}$ been inconsistent, we would have that for some B, $\neg A \vdash_R \neg B \wedge B$. This, in turn, entails that $A \vee \neg A \vdash_R A \vee (\neg B \wedge B)$, and so that $\vdash_R A \vee (\neg B \wedge B)$. On the other hand, $\vdash_R \neg(\neg B \wedge B)$. By the famous theorem of Meyer and Dunn concerning the admissibility of the disjunctive syllogism in R [Anderson and Belnap, 1975; Dunn, 1986] it would follow, therefore, that $\vdash_R A$, contradicting our assumption. Hence $\{\neg A\}$ is consistent, and so, by the c-normality of R which we have just proved, it has a consistent and complete extension which obviously does not prove A. This shows that R is weakly normal (the proof for RM is identical). ■

COROLLARY. $\vdash_R^{CS}$ is strongly consistent, c-normal and weakly normal, but it is not strongly c-normal.

Note: A close examination of the proof of the last theorem shows that the properties of R which are described there are shared by many of its relatives (like RM, for example). We have, in fact, the following generalizations:

1. Every extension of R which is not strongly consistent is also not strongly c-normal.

2. Every extension of R by axiom-schemes is both strongly complete and c-normal.

3. Every extension of R by axiom schemes for which the disjunctive syllogism is an admissible rule[13] is weakly normal.

In fact,(1)–(3) are true (with similar proofs) also for many systems weaker than R in the relevance family, like E.

Our results show that $\vdash_R^{CP} = \vdash_R$, but $\vdash_R^N \neq \vdash_R^{CS}$ (since R is not strongly c-normal). Hence $\vdash_R^N$ is a new consequence relation, and we turn next to axiomatize it.

DEFINITION. Let $\mathcal{L}$ be an extension of R by axiom schemes and let $\mathcal{L}^N$ be the system which is obtained from $\mathcal{L}$ by adding to it the disjunctive syllogism (γ) as an extra rule: from $\neg A$ and $A \vee B$ infer B.

THEOREM 20. $\vdash_{\mathcal{L}}^N = \vdash_{\mathcal{L}^N}$.

Proof. To show that $\vdash_{\mathcal{L}^N} \subseteq \vdash_{\mathcal{L}}^N$ it is enough to show that $\neg A$, $A \vee B \vdash_{\mathcal{L}}^N B$. This was already done, in fact, in the proof of the last theorem. For the converse, assume $T \vdash_{\mathcal{L}}^N A$. Since $\mathcal{L}$ is c-normal (see last note), $T \cup \{\neg A\}$

[13] See [Anderson and Belnap, 1975] and [Dunn, 1986] for examples and criteria when this is the case.

cannot be $\mathcal{L}$-consistent. Hence $T \cup \{\neg A\} \vdash_{\mathcal{L}} \neg B \wedge B$ for some B. This entails that $T \vdash_{\mathcal{L}} A \vee (\neg B \wedge B)$ and that $T \vdash_{\mathcal{L}^N} A$ exactly as in the proof of the weak normality of R. ■

3.5 The Purely Relevant Logic RMI

The purely relevant logic RMI was introduced in citeAv90a,Av90b. Proof-theoretically it differs from R in that:

(i) The converse of contraction (or, equivalently, the mingle axiom of RM) is valid in it. This is equivalent to the idempotency of the intensional disjunction + (="par" of Girard). In the purely multiplicative fragment RMI_m it means also that assumptions with respect to $\rightarrow$ can be taken as coming in *sets* (rather than multisets, as in LL_m or R_m).

(ii) The adjunction rule ($B, C \vdash B \wedge C$) as well as the distribution axiom ($A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)$) are accepted only if B and C are "relevant". This relevance relation can be expressed in the logic by the sentence $R^+(A, B) = (A \rightarrow A) + (B \rightarrow B)$, which should be added as an extra premise to adjunction and distribution (this sentence is the counterpart of the "mix" rule of [Girard, 1987]).

We start our investigation with the easier case of RMI_m.

THEOREM 21. *Exactly like R_m, RMI_m is not strongly consistent, but it is both strongly complete and strongly c-normal.*

Proof. Exactly like in the case of R_m. ■

Like in classical logic, and unlike the case of R_m, these two main properties of RMI_m are strongly related to simple, intuitive, algebraic semantics. Originally, in fact, RMI_m was designed to correspond to a class of structures which are called in [Avron, 1990a] "full relevant disjunctive lattices" (full r.d.l.). A full r.d.l is a structure which results if we take a tree and attach to each node b its own two basic truth-values $\{t_b, f_b\}$. To a leaf b of the tree we *can* attach instead a single truth-value I_b which is the negation of itself (its meaning is "both true and false" or "degenerate"). b is called abnormal in this case. Intuitively, the nodes of the tree represent "domains of discourse". Two domains are relevant to each other if they have a common branch, while b being nearer than a to the root on a branch intuitively means that b has a higher "degree of reality" (or higher "degree of significance") than a (we write $a < b$ in this case). The operation of $\neg$ (negation) is defined on a full r.d.l. M in the obvious way, while + (relevant disjunction) is defined as follows: Let $|t_a| = |f_a| = |I_a| = a$, and let $\mathrm{val}(t_b) = t$, $\mathrm{val}(f_b) = f$ and $\mathrm{val}(I_b) = I$. Define $x \leq_+ y$ if either $x = y$ or $|x| < |y|$ or $|x| = |y|$ and $\mathrm{val}(y) = t$. $(M, \leq_+)$ is an upper semilattice. Let

$x + y = \sup_{\leq_+}(x, y)$. An RMI_m-model is a pair (M, v) where M is a full r.d.l. and v a valuation in it (which respects the operations). A sentence A is true in a model (M, v) if $\mathrm{val}(v(A)) \neq f$. Obviously, every model (M, v) determines an RMI_m-setup of all the formulae which are true in it. Denote the collection of all these setups by RDL_m.

PROPOSITION 22. $CP_{RMI_m} = RDL_m$

Proof. It is shown in [Avron, 1990b] that the Lindenbaum algebra of any complete RMI_m-theory determines a model in which exactly its sentences are true. This implies that $CP_{RMI_m} \subseteq RDL_m$. The converse is obvious from the definitions. ■

CPROLLARY. [Avron, 1990b]: RMI_m is sound and complete for the semantics of full r.d.l.s. In other words: $T \vdash_{RMI_m} A$ iff A is true in every model of T.

Proof. Checking soundness is straightforward, while completeness follows from the syntactic strong compleness of RMI_m (theorem 21) and the last theorem. ■

The strong c-normality of RMI_m also has an interpretation in terms of the semantics of full r.d.l.s. In order to describe it we need first some definitions:

DEFINITION.

1. A full r.d.l is consistent iff for every x in it $\mathrm{val}(x) \in \{t, f\}$ (i.e., the intermediate truth-value I is not used in its construction). This is equivalent to: $x \neq \neg x$ for all x.

2. A model (M, v) is consistent iff M is consistent.

3. $CRDL_m$ is the collection of the RMI_m-setups which are determined by some consistent model.

Note: On every tree one can base exactly one consistent full r.d.l. (but in general many inconsistent ones).

PROPOSITION 23. $N_{RMI_m} = CRDL_m$.

Proof. In the construction from [Avron, 1990b] which is mentioned in the proof of proposition 22, a complete and consistent theory is easily seen to determine a consistent model. The converse is obvious. ■

In view of the last proposition, the strong c-normality of RMI_m and its two obvious corollaries (weak normality and c-normality) can be reformulated in terms of the algebraic models as follows:

PROPOSITION 24.

1. *If T is consistent then $T \vdash_{RMI_m} A$ iff A is true in any consistent model of T.*

2. $\vdash_{RMI_m} A$ *iff* A *is true in any consistent model.*

3. *Every consistent* RMI_m*-theory has a consistent model.*

It follows that if we restrict our attention to consistent RMI_m-theories, we can also restrict our semantics to consistent full r.d.l.s, needing, therefore, only the classical two truth-values t and f, but not I.

Exactly as in the case of R, when we pass to RMI things become more complicated. Moreover, although we are going to show that RMI has *exactly* the same properties as R, the proofs are harder.

THEOREM 25. *RMI is strongly complete.*

Proof. The proof is like the one for R given above, since RMI has the relevant properties of R which were used there (see [Avron, 1990b]). ■

Like in the case of RMI_m, the strong completeness of RMI is directly connected to the semantics of full r.d.l.s. This semantics is extended in [Avron, 1990a; Avron, 1990b] to the full language by defining the operator $\wedge$ on a full r.d.l. as follows: define $\leq$ on M by: $x \leq y$ iff $\mathrm{val}(\neg x + y) \neq f$. $(M, \leq)$ is a lattice. Let $x \wedge y = \inf_{\leq}(x, y)$. The notions of an RMI-model, consistent RMI-model and the truth of a formula A (of the language of RMI) in such models are defined as in the case of RMI_m. The classes of setups RDL and $CRDL$ are also defined like their counterparts in the case of RMI_m. Again we have:

PROPOSITION 26.

1. $CP_{RMI} = RDL$.

2. $N_{RMI} = CRDL$.

Proof. Similar to the proofs of propositions 22 and 23. ■

Again, theorem 25 and 26(1) entail the following result of [Avron, 1990b]:

COROLLARY. RMI is sound and complete for the semantics of full r.d.l.s.

THEOREM 27.

1. $\vdash_{RMI} A$ *iff* A *is valid in all the consistent models.*

2. RMI *is weakly normal.*

Proof.

1. Suppose that $\not\vdash_{RMI} A$. Then there is a model (M, v) in which A is not true. Let M' be the consistent full r.d.l based on T_M (the tree on which M is based). Let v' be any valuation in M' which satisfies the following conditions: (i) $|v'(P)| = |v(P)|$ for every atomic P,

(ii) $v'(P) = v(P)$ whenever $|v(P)|$ is normal in M. It is easy to see that conditions (i) and (ii) are preserved if we replace P by any sentence. In particular $v'(A) = v(A)$ and so A is not valid in the consistent model M'.

2. Immediate from part (1) and proposition 26(2) ■

THEOREM 28.

1. *RMI is c-normal.*
2. *Every consistent RMI-Theory has consistent model.*

Proof. (1) By proposition 8(4) it suffices to prove that if T is consistent and A a sentence then either $T \cup \{A\}$ or $T \cup \{\neg A\}$ is consistent. This is not so easy, however, since like in R, $T \cup \{A\}$ might be inconsistent even if $T \not\vdash \neg A$, while unlike in R, (γ) for $\vee$ is *not* sound for $\vdash^N_{RMI}$.

Suppose then that $T \cup \{A\}$ and $T \cup \{\neg A\}$ are both inconsistent. Since $\neg B,\ B \vdash_{RMI} \neg(B \to B)$, this means, by RMI deduction theorem for $\supset$[14] that there exist sentences B and C such that $T \vdash_{RMI} A \supset \neg(B \to B)$, $T \vdash_{RMI} \neg A \supset \neg(C \to C)$. In order to prove that T is inconsistent it is enough therefore to show that the following theory F_0 is inconsistent:

$$F_0 = \{A \supset \neg(B \to B)\ \ ,\ \ \neg A \supset \neg(C \to C)\}\ .$$

For this we show that the following sentence φ and its negation are theorems of F_0 (where $a \circ b = \neg(\neg a + \neg b)$):

$$\varphi = (B \vee [\neg A \circ R^+\,(A + C, B)]) \wedge (C \vee [(A + C) \circ R^+\,(A + B, C)])\ .$$

By the completeness theorem it suffices to show that φ gets a neutral value (I) in every model of F_0. Let (M, v) be such a model, and denote by R the relevance relation between the nodes of the tree on which M is based. It is easy to see that:

a) $|v(A)| \not< |v(B)| \qquad |v(A)| \not< |v(C)|$

b) If $|v(A)|\, R\, |v(B)|$ or if $v(A)$ is designated then $v(B)$ is neutral.

c) If $|v(A)|\, R\, |v(C)|$ or if $v(\neg A)$ is designated then $v(C)$ is neutral.

Denote, for convenience, $v(A)$ by a, $v(B)$ by b, $v(C)$ by c, and the two conjuncts of φ by φ_1 and φ_2 respectively. Then:

(i) If $|b|\, R\, (|a| \vee |c|)$ then $v(\varphi_1) = b$. Also we have then that $|c| \leq |a| \vee |c| < |a| \vee |b| = |a + b|$ (since always $(|a| \vee |b|)\, R\, (|a| \vee |c|)$). Hence $|c|\, R\, |a + b|$ and so $v\,(\varphi_2) = t_{|a| \vee |b| \vee |c|}$. It follows that $v\,(\varphi) = b$ and so $v\,(\varphi)$ is neutral by b) above.

[14] See [Avron, 1990b]. The connective $\supset$ is defined there by $a \supset b = b \vee (a \to b)$.

(ii) If $|b|\, R\, (|a| \vee |c|)$ and either $|a| < |a| \vee |c|$ or $\mathrm{val}(a) = f$ then, by a), $|b| \leq |a| \vee |c|$ and either $|a|\, R\!\!\!/\, |c|$ or $v(\neg A)$ is designated. Hence c is neutral by c). It follows (since either $|a|\, R\!\!\!/\, |c|$ or $\mathrm{val}(a) = f$), that either $|a|\, R\!\!\!/\, |c|$ or $|c| < |a|$. In both cases $v(A + C) = f_{|a|\vee|b|\vee|c|}$, $v(\varphi_2) = c$, and $v(\varphi_1) = t_{|a|\vee|b|\vee|c|}$. Hence $v(\varphi) = c$, which is neutral.

(iii) If $|b|\, R\, (|a| \vee |c|)$, $|a| = |a| \vee |c|$ and a is designated then, by a), $|a| = |a| \vee |b| \vee |c|$. If $\mathrm{val}(a) = I$ then also $\mathrm{val}(b) = I$ and $\mathrm{val}(c) = I$, and so $\mathrm{val}(v(\varphi)) = I$. If $\mathrm{val}(a) = t$ then by b) b is neutral and so $|b| < |a|$ ($|a|$ is normal!). Obviously $|c| \leq |a|$ in this case, and so $v(\varphi_1) = b$, $v(\varphi_2) = t_{|a|} = a$ and $v(\varphi) = b$, which is neutral.

(2) Immediate from (1) and proposition 26(2). ■

PROPOSITION 29. *RMI is not strongly c-normal.*

Proof. Let ψ_1 and ψ_2 be the two elements of the theory F_0 from the last proof. Let $T = \{\psi_1\}$, $A = \neg\psi_2$. Then T is consistent (even classically!) and A is provable in every consistent and complete extension of T (since F_0 is inconsistent). Hence $T \vdash^N_{RMI} A$. However, $T \not\vdash_{RMI} A$ since it is easy to construct a full model of ψ_1 in which $\neg\psi_2$ is not true. (ψ_1 is neutral in this model.) ■

Like in the case of R, our results show that $\vdash^N_{RMI}$ is stronger than $\vdash_{RMI}$ and $\vdash^{CS}_{RMI}$. We now construct a formal system for this consequence relation.

DEFINITION. The system $RMIC$ is RMI strengthened by *M.T.* for $\supset$:

$$A \supset B\ , \quad \neg B \vdash \neg A\ .$$

THEOREM 30.

1. *$T \vdash_{RMIC} A$ iff $T \vdash^N_{RMI} A$*
2. *$\vdash_{RMIC} A$ iff $\vdash_{RMI} A$.*

Proof.

1. Obviously, if both $A \supset B$ and $\neg B$ are true in a consistent model (M, v) then so is $\neg A$. Hence if $T \vdash_{RMIC} A$ then $T \vdash^N_{RMI} A$. For the converse, suppose $T \vdash^N_{RMI} A$. Then by Theorem 26 $T \cup \{\neg A\}$ has no consistent model. This means, by Theorem 28, that $T \cup \{\neg A\}$ is inconsistent. Hence $T \vdash_{RMI} \neg A \supset \neg(B \to B)$ for some B. Since also $\vdash_{RMI} \neg\neg(B \to B)$, we have that $T \vdash_{RMIC} \neg\neg A$, by applying M.T. Hence $T \vdash_{RMIC} A$.
2. Immediate from 1) and theorem 27(2). ■

Notes:

1. From 30(2) it is clear that the system RMI is closed under M.T. for $\supset$. By applying this rule to theories we can make, however, any inconsistent theory trivial. This resembles the status of (γ) in R and E. Indeed (γ) may be viewed as M.T. for the usual implication as defined in classical logic. A comparison of theorems 30 and 20 deepens the analogy (note that RMI is *not* an extension of R and 20 fails for it!).

2. Despite 30(2) RMI and $RMIC$ are totally different even for consistent theories, as we have seen in prop. 29. *It is important, however, to note that theory T is consistent in RMI iff it is consistent in $RMIC$.* This follows easily from theorem 28.

3.6 Three Valued Logics

Like in section 2, we consider here only the 3-valued logic which we call in [Avron, 1991b] "natural" (in fact, only those with Tarskian CR). All these logics have the connectives $\{\neg, \wedge, \vee\}$ as defined by Kleene. The weaker ones have only these connectives as primitive. The stronger ones have also an implication connective which reflect their consequence relation.

Suppose the truth-values are $\{t, f, I\}$. t and f correspond to the classical truth values. Hence t is designated, f is not. The 3-valued logics are therefore naturally divided into two main classes: those in which I is not designated, and those in which it is. The first type of logics can be understood as those in which the law of contradiction is valid, but excluded middle is not. The second type – the other way around.

Kleene's basic 3-valued logic

This logic, which we denote by $K\ell$, has only t as designated and $\{\neg, \vee, \wedge\}$ as primitives. It has no valid formula, but it does have a non-trivial consequence relation, defined by the 3-valued semantics. A setup in this semantics is any set of the form $\{A \mid v(A) = t\}$ where v is a 3-valued valuation, and the consequence relation $\vdash_{K\ell}$ is defined by this semantics. A sound and strongly complete Gentzen-type or natural deduction formulations have been given in several places (see, e.g., [Barringer *et al.*, 1984] or [Avron, 1991b]).

The properties of $\vdash_{K\ell}$ which are relevant to the present paper are summarized in the following theorem:

THEOREM 31.

1. *Like intuitionistic logic, $\vdash_{K\ell}$ is strongly consistent, c-normal but not even weakly complete.*

2. *$\vdash_{K\ell}^{CP}$ is classical logic.*

Proof.

1. Since $\neg A,\ A \vdash_{K\ell} B$, $\vdash_{K\ell}$ is strongly consistent. Since $\vdash^{CP}_{K\ell}\ A \vee \neg A$ but $\not\vdash_{K\ell} A \vee \neg A$, $\vdash_{K\ell}$ is not weakly complete.

   We turn now to c-normality. First we need a lemma

   LEMMA 32. *If T has a 3-valued model then it has also a classical, two valued model.*

   Proof of the lemma: It is enough to show that every finite subset of T has a two-valued model (by compactness of classical logic). So let Γ be a finite set which has a 3-valued model. Since De-Morgan laws and the double-negation laws are valid for the three-valued truth tables, we may assume that all the formulas in Γ are in negation normal form. We prove now the claim by induction on the number of $\wedge$ and $\vee$ in Γ. If all the formulas in Γ are either atomic or negations of atomic formula, then the claim is obvious. If $\Gamma = \Gamma_1 \cup \{A \wedge B\}$ then Γ has a model iff $\Gamma_1 \cup \{A, B\}$ has a model, and so we can apply the induction hypothesis to $\Gamma_1 \cup \{A, B\}$. If $\Gamma = \Gamma_1 \cup \{A \vee B\}$ then Γ has a model iff either $\Gamma_1 \cup \{A\}$ or $\Gamma_1 \cup \{B\}$ has, and we can apply the induction hypothesis to the one which does, getting by this a two-valued model for Γ. ■

   To complete the proof of the theorem, let T be a consistent $\vdash_{K\ell}$-theory. The definitions of consistency and of $\vdash_{K\ell}$ imply in this case that it has some 3-valued model. By the lemma it has also a two-valued model. Let T^* be the set of all the formulae that are true in that two-valued model. Then T^* is a $\vdash_{K\ell}$-setup which is consistent (even classically), complete, and an extension of T.

2. Since $\vdash^{CP}_{K\ell}\ \neg A \vee A$ and $\neg A \vee C,\ A \vee B \vdash_{K\ell} C \vee B$, it is easy to show, using (for example) Shoenfield's axiomatization of classical logic in [Shoenfield, 1967] that $\vdash_{C\ell} \subseteq \vdash^{CP}_{K\ell}$. The converse is obvious, since $\vdash_{K\ell} \subseteq \vdash_{C\ell}$ and $\vdash_{C\ell}$ is strongly complete (by $\vdash_{C\ell}$ we mean here classical logic). ■

$LPF/Ł_3$

LPF was developed in [Barringer *et al.*, 1984] for the VDM Project. As explained in [Avron, 1991b], it can be obtained from $\vdash_{K\ell}$ by adding an internal implication $\supset$ so that $T, A \vdash_{LPF} B$ iff $T \vdash_{LPF} A \supset B$. The definition of $\supset$ is: $a \supset b = t$ if $a \neq t$, b if $a = t$. Alternatively one can add to the language Łukasiewicz's implication, or the operator Δ used in [Barringer *et al.*, 1984]. All these connectives are definable from one another with the help of $\neg, \wedge$ and $\vee$.

THEOREM 33.

1. $\vdash_{LPF}$ *is strongly consistent but neither weakly complete nor c-normal.*

2. $\vdash^{CP}_{LPF}$ *is classical logic.*

Proof.

1. That $\vdash_{LPF}$ is strongly consistent but not weakly normal follows from the corresponding fact for $\vdash_{K\ell}$, since $\vdash_{LPF}$ is a conservative extension of $\vdash_{K\ell}$. As for c-normality, it is enough to note that $\{(A \vee \neg A) \supset B, \neg B\}$ is consistent in LPF (take $v(A) = I$, $v(B) = f$) but obviously has no consistent and complete extension.

2. Again, take any axiomatization of classical logic in the LPF-language and check that all the axioms and rules are valid in $\vdash^{CP}_{LPF}$. ■

The Basic Paraconsistent 3-valued logic PAC

This logic, which we call PAC in [Avron, 1991b] [15], has the same language (with the same definitions of the connectives) as $\vdash_{K\ell}$. The difference is that here both t and I are designated. A setup in the intended semantics is, therefore, this time a set of the form $\{A \mid v(A) = t \text{ or } v(A) = I\}$,where v is a three-valued valuation. A sound and strongly complete (relative to the 3-valued semantics) Gentzen-type axiomatization is given in [Avron, 1991b].[16]

THEOREM 34.

1. $\vdash_{PAC}$ *is strongly complete, weakly normal and c-normal. It is neither strongly consistent nor strongly c-normal.*

2. $\vdash^{N}_{PAC}$ *is identical to classical logic.*

[15] It is a fragment of several logics which got several names in the literature – see next subsection.

[16] Giving a faithful Hilbert-type system is somewhat a problem here, since the set of valid formulas is identical to that of classical logic, but the consequence relation is not.

Proof.

1. The strong completeness theorem for the Gentzen-type system entails that $\vdash_{PAC}$ is finitary. Hence to show strong syntactical completeness it is enough to show that the condition in 8(1) obtains. This is easy. Weak normality is immediate from the fact that $\vdash_{PAC} A$ iff A is a classical tautology (see [Avron, 1991b]) and that $\vdash_{PAC} \subseteq \vdash_{C\ell}$. c-normality is proved exactly as for R (it is easy to check that $\vdash_{PAC}$ has all the properties which are used in that proof). It is also easy to check that $\neg p, p \not\vdash_{PAC} q$ and that $\{\neg p, p \vee q\}$ is consistent, that $\neg p, p \vee q \vdash^{N}_{PAC} q$ but $\neg p, p \vee q \not\vdash_{PAC} q$ (take $v(p) = I$, $v(q) = f$). Hence $\vdash_{PAC}$ is not strongly c-normal and not strongly consistent.

2. Since all classical tautologies are valid in $\vdash_{PAC}$ and MP for classical implication is valid for $\vdash^{N}_{PAC}$, $\vdash_{C\ell} \subseteq \vdash^{N}_{PAC}$. The converse is obvious, since $\vdash_{C\ell}$ is strongly c-normal and $\vdash_{PAC} \subseteq \vdash_{C\ell}$. ■

RM_3/J_3

This logic is obtained from PAC by the addition of certain connectives while keeping the same CR. There are two essential ways that this has been done (independently) in the literature (they were shown equivalent in [Avron, 1991b]):

(i) Adding an implication $\rightarrow$, defined as in [Sobociński, 1952]. In this way we get the strongest logic in the relevance family: the three-valued extension of RM. It is in this way that this logic arose in the relevance literature. The corresponding matrix is called there M_3 and the logic RM_3. It can be axiomatized by adding to R the axioms $A \rightarrow (A \rightarrow A)$ and $A \vee (A \rightarrow B)$.

(ii) Adding an implication $\supset$, defined by (see [da Costa, 1974]) $a \supset b = t$ if $a = f$, $a \supset b = b$ otherwise. For this connective the deduction theorem holds. In this form the logic was called J_3 in [D'Ottaviano, 1985] (see also [Epstein, 1995]) [17]. It was independently investigated also in [Avron, 1986] and in [Rozonoer, 1989]. Strongly complete Hilbert-type formulations with M.P. for $\supset$ as the only rule of inference were given in those papers, and a cut-free Gentzen-type formulation can be found in [Avron, 1991b].

In what follows we shall use the neutral name Pac^* for the CR of PAC in the extended language. The next theorem shows that the main difference between Pac^* and PAC is that Pac^* is *not* weakly normal.

[17] [D'Ottaviano, 1985] and [Epstein, 1995] consider a language with more connectives, but we shall not treat them here.

THEOREM 35.

1. *Pac^* is strongly complete and c-normal. It is neither strongly consistent nor weakly normal.*
2. *$\vdash^N_{Pac^*}$ is identical to classical logic.*

Proof.

1. Strong completeness and c-normality can easily be proved. Since $\vdash_{Pac^*}$ is a conservative extension of $\vdash_{Pac}$, it is not strongly consistent. Finally $\vdash^N_{Pac^*} A \wedge \neg A \supset B$, since $\neg(A \wedge \neg A \supset B) \vdash_{Pac^*} A \wedge \neg A$, but $\nvdash_{Pac^*} A \wedge \neg A \supset B$ (the same argument applies to $(A \wedge \neg A \rightarrow B)$).
2. It is provable in [Dunn, 1970] that classical logic is the only proper extension of RM_3 in the language of $\{\neg, \vee, \wedge, \rightarrow\}$ (from the point of view of theoremhood). Since we have just seen that the set of valid sentences in $\vdash^N_{Pac^*}$ is such a proper extension, and since MP for $\rightarrow$ is valid for it, $\vdash^N_{Pac^*}$ should be identical to $\vdash_{C\ell}$ (in this language). The same argument works for the $\{\neg, \vee, \wedge, \supset\}$ language using the results of [Avron, 1986]. Alternatively, it is not difficult to show that by adding $\neg A \wedge A \rightarrow B$ to the Hilbert-type formulation of RM_3 or $\neg A \wedge A \supset B$ to that of J_3 we get classical logic in the corresponding languages. ■

4 CONCLUSION

We have seen two different aspects of negation. From our two points of view the major conclusions are:

- The negation of classical logic is a perfect negation from both syntactical and semantic points of view.
- Next come the intensional fragments of the standard relevance logics (R_m, RMI_m, RM_m). Their negation is an internal negation for their associated internal CR. Relative to the external one, on the other hand, it has the optimal properties one may expect a semantic negation to have in a paraconsistent logic. In the full systems (R, RMI, RM) the situation is similar, though less perfect (from the semantic point of view). It is even less perfect for the 3-valued paraconsistent logic.
- The negation of Linear Logic is a perfect internal negation w.r.t. its associated internal CR. It is not, however, a negation from the semantic point of view. The same applies to Łukasiewicz 3-valued logic.
- The negations of intuitionistic logic and of Kleen's 3-valued logic are not really negations from the two points of view presented here.

In addition we have seen that within our general semantic framework, any consequence relation which is not strongly normal naturally induces one or more derived consequence relations in which its negation better deserves this name. We gave sound and complete axiomatic systems for these derived relations for all the substructural logics we have investigated.

Department of Computer Science, Tel Aviv University, Israel.

BIBLIOGRAPHY

[Anderson and Belnap, 1975] A. R. Anderson and N. D. Belnap. *Entailment* vol. 1, Princeton University Press, Princeton, NJ, 1975.

[Anderson and Belnap, 1992] A. R. Anderson and N. D. Belnap. *Entailment* vol. 2, Princeton University Press, Princeton, NJ, 1992.

[Avron, 1986] A. Avron. On an Implication Connective of RM, *Notre Dame Journal of Formal Logic*, **27**, 201–209, 1986.

[Avron, 1988] A. Avron. The Semantics and Proof Theory of Linear Logic, *Journal of Theoretical Computer Science*, **57**, 161–184, 1988.

[Avron, 1990a] A. Avron. Relevance and Paraconsistency - A New Approach., *Journal of Symbolic Logic*, **55**, 707–773, 1990.

[Avron, 1990b] A. Avron. Relevance and Paraconsistency - A New Approach. Part II: the Formal systems, *Notre Dame Journal of Formal Logic*, **31**, 169–202, 1990.

[Avron, 1991a] A. Avron. Simple Consequence relations, *Information and Computation*, **92**, 105–139, 1991.

[Avron, 1991b] A. Avron. Natural 3-valued Logics— Characterization and Proof Theory, *Journal of Symbolic Logic*, **56**, 276–294, 1991.

[Avron, 1992] A. Avron. Axiomatic Systems, Deduction and Implication *Journal of Logic and Computation*, **2**, 51–98, 1992.

[Avron, 1994] A. Avron. What is a Logical System?, in [Gabbay, 1994; pp. 217–238].

[Barringer *et al.*, 1984] H. Barringer, J. H. Cheng and C. B. Jones. A Logic Covering Undefiness in Program Proofs, *Acta Informatica*, **21**, 251–269, 1984.

[Cleave, 1991] J. P. Cleave. *A Study of Logics*, Oxford Logic Guides, Clarendon Press, Oxford, 1991.

[da Costa, 1974] N. C. A. da Costa. Theory of Inconsistent Formal Systems, *Notre Dame Journal of Formal Logic*, **15**, 497–510, 1974.

[D'Ottaviano, 1985] I. M. L. D'Ottaviano. The completeness and compactness of a three-valued first-order logic, *Revista Colombiana de Matematicas*, **XIX**, 31–42, 1985.

[Dunn, 1970] J. M. Dunn. Algebraic completeness results for R-mingle and its extensions, *The Journal of Symbolic Logic*, **24**, 1–13, 1970.

[Dunn, 1986] J. M. Dunn. Relevant logic and entailment. In *Handbook of Philosophical Logic*, 1st edition, Vol III, D. M. Gabbay and F. Guenthner, eds. pp. 117–224. Reidel: Dordrecht, 1986.

[Epstein, 1995] R. L. Epstein. *The Semantic Foundations of Logic, vol. 1: Propositional Logics*, 2nd edition, Oxford University Press, 1995.

[Fagin *et al.*, 1992] R. Fagin, J. Y. Halpern and Y. Vardi. What is an Inference Rule? *Journal of Symbolic Logic*, **57**, 1017–1045, 1992.

[Gabbay, 1981] D. M. Gabbay. *Semantical investigations in Heyting's intuitionistic logic* , Reidel: Dordrecht, 1981.

[Gabbay, 1994] D. M. Gabbay, ed. *What is a Logical System?* Oxford Science Publications, Clarendon Press, Oxford, 1994.

[Girard, 1987] J.-Y. Girard. Linear Logic, *Theoretical Computer Science*, **50**, 1–101, 1987.

[Hacking, 1979] I. Hacking. What is logic? *The Journal of Philosophy*, **76**, 185–318, 1979. Reprinted in [Gabbay, 1994].

[Jones, 1986] C. B. Jones. *Systematic Software Development Using VDM*, Prentice-Hall International, UK, 1986.

[Rozonoer, 1989] L. I. Rozonoer. On Interpretation of Inconsistent Theories, *Information Sciences*, **47**, 243–266, 1989.

[Scott, 1974] D. Scott. Rules and derived rules. In *Logical Theory and Semantical Analysis*, S. Stenlund, ed. pp. 147–161, Reidel: Dordrecht, 1974.

[Scott, 1974b] D. Scott. Completeness and axiomatizability in many-valued logic. In *Proceeding of the Tarski Symposium*, Proceeding of Symposia in Pure Mathematics, vol. XXV, American Mathematical Society, Rhode Island, pp. 411–435, 1974.

[Schroeder-Heister and Došen, 1993] P. Schroeder-Heister and K. Došen, eds. *Substructural Logics*, Oxford Science Publications, Clarendon Press, Oxford, 1993.

[Shoenfield, 1967] J. R. Shoenfield. *Mathematical Logic*, Addison-Wesley, Reading, Mass., 1967.

[Sobociński, 1952] B. Sobociński. Axiomatization of partial system of three-valued calculus of propositions, *The Journal of Computing Systems*, **11**, 23–55, 1952.

[Troelstra, 1992] A. S. Troelstra. *Lectures on Linear Logic*, CSLI Lecture Notes No. 29, Center for the Study of Language and Information, Stanford University, 1992.

[Urquhart, 1984] A. Urquhart. Many-valued Logic. In *Handbook of Philosophical Logic*, Vol III, first edition. D. Gabbay and F. Guenthner, eds. pp. 71–116. Reidel: Dordrecht,, 1984.

[Wojcicki, 1988] R. Wojcicki. *Theory of Logical Calculi*, Synthese Library, vol. 199, Kluwer Academic Publishers, 1988.

TON SALES

LOGIC AS GENERAL RATIONALITY: A SURVEY

Logic today is urged to confront and solve the problem of reasoning under non-ideal conditions, such as *incomplete information* or *imprecisely formulated statements*, as is the case with *uncertainty*, *approximate* descriptions or linguistic *vagueness*. At the same time, Probability theory has widened its traditional field of analysis (the *expected frequency* of physical phenomena) so as to encompass and analyze general *rational expectations*. Thus, Probability has placed itself in the position of offering Logic a solution for its own long-awaited generalization. The basis for that turns out to be precisely the shared base underlying the two disciplines. This theoretical base predates their common birth, as seen in the early efforts of Bernoulli and Laplace, as well as in Boole's 1854 attempt to formalize the "laws of thought" and then, as he claimed, to "derive Logic and Probability" from them. Once we recover (following Popper's 1938 advice) the underlying *formalism*, we come, by interpreting it in two different directions, back into either *Logic* or *Probability*. The present survey explains the story so far and does the reconstruction work from the logical point of view. The stated aim is to generalize *Logic* so as to cover, as Boole intended, the whole of *rationality*.

INTRODUCTION

This survey could as well be entitled: *"How Logic was once the same as Probability, and then they diverged —and how they may again be formally the same"*, or *"Logic and classical Probability: recovering the lost common ground"*. Before we begin, let us say that the implied desideratum of the title(s) is long overdue. Indeed, that *(a)* standard Logic *can* be generalized, and that *(b)* the natural generalization of Logic *is* —or derives from, or is suggested by— Probability theory seems at present the shared conviction of a number of logicians and probabilists. Thus, to cite a few of the latter, Ramsey wrote (in 1926) that the laws of probability are actually laws of *consistency* (or *rational* behavior), an extension of Formal Logic to cover partial information, and that Probability theory could become the "logic of consistency" which would control and guarantee, as Mathematics does, that our beliefs are not self-contradictory. At about the same time de Finetti concluded that Probability theory is the only possible "logic" to generalize standard Logic. All the same, Patrick Suppes was considering in 1979 that Probability theory is the natural extension of classical deductive inference

D. Gabbay and F. Guenthner (eds.),
Handbook of Philosophical Logic, Volume 9, 321–366.

rules, while, more recently, Glenn Shafer declared that "probability is not really about numbers; it is about the structure of reasoning".

Why the technicalities of Probability theory should be viewed today as a powerful *generalizer* of standard Logic and a suitable formal *unifier* of the two may come as a surprise both to probabilists and logicians. Actually, the idea —that we pursue in our generalization below— is very simple: Probability and Logic are but *two interpretations of a same underlying concept*. This is how the founders of both theories saw it and what Popper later explicitly said (and Kolmogorov claimed he had done). But this old notion, over which notable thinkers like Reichenbach or Carnap agonized after the 1930s, is shared nowadays by a surprisingly exiguous minority of specialists (in both disciplines).

Logic and Probability are overwhelmingly seen today as two completely disparate fields, with a very few, if any, points of contact. Logic deals with *reasoning* and *truth*, Probability with inference on poor data. They seem to have nothing in common. Though they both start with a set $\mathcal{B}$ of Boolean-structured objects (respectively *sentences* and *events* —or, confusingly, "propositions") and though they assign them values in a simple number system containing the one and the zero (here logicians prefer 'truth' and 'falsity', though), at this point the similarity apparently ends, for the two valuations are perceived to be very different: the probabilistic $P : \mathcal{B} \to [0, 1]$ obeys a set of axioms set forth by Kolmogorov in the 1930s (that do not actually follow from any particularly "probabilistic" rules but rather reduce it to a simple 'measure' —in the technical sense— of the "event" objects), while the *logical* valuation is felt to be of a quite different nature and regulated by a semantics set forth by Tarski, also in the thirties, and buttressed by elaborate, specialized logical considerations. Moreover, either field not only has a different type of problem to solve, it also has a different, incompatible set of base concepts and interpretations to work on. And the diverging traditions have bred different strokes of unrelated practitioners and two methodologies that are seen by the mainstream mathematician as far distant (even lying at opposite fields of Mathematics, i.e. *real* vs. *discrete*).

However, this is not how things were seen in the first stages of the modern theories of Probability and Logic. Their founders, notably Bernoulli and Laplace, or Boole and Peirce, dithered a lot on what might "probability" or "truth" mean, and often tended to explain one through the other in incipient, half-baked intersecting intuitions, as can be readily seen by browsing into the original literature. Later developments, as well as the progressively firmer foundations and the more specialized and mutually deviating interpretations that either field painstakingly acquired, created an increasing gap between Probability and Logic, in which both contenders apparently never found a ground or occasion to reconcile into one unified approach (which, as we suggest below, is not only desirable but feasible and even natural).

Why striving to recover an encompassing view should be interesting at all may be not obvious at first, but there are strong reasons for it. First, because, as we said, both theories are two differing interpretations of the same idea. Second, because both probabilists and logicians have recently been hard pressed against the limits of their own disciplines when confronting new challenges with consecrated methods. Prominent challenges include: (1) for probabilists, how to clarify the ultimate meaning, and the practical import, of the apparently obvious idea of "probability" (doubters here include names as Keynes [1921], Ramsey [1926] or de Finetti [1931], and the questions raised prolong well to this day into widely-discussed conundrums as the status of *subjective probability*, *rational belief* or *bayesianism*); or (2) for logicians, how to validate reasoning under *uncertainty* or with *incomplete* or *approximate* information (a problem that eventually gave rise, also around the 1930s, to non-standard formalisms such as the *many-valued logics* of Łukasiewicz [1920] (and [1930], with Tarski) or Kleene [1938], or the attempts at defining a *probability logic* by Reichenbach [1935a,b], discussed by Carnap [1950].

The material below is structured in two parts: the first is a short survey explaining why Logic and classical Probability were once the same thing —and gave the (common) pioneers (Bernoulli, Hume, Laplace, Boole) lots of cross-supporting arguments— and why they soon diverged to the point of being considered unrelated. The second part –considerably longer– is a summary of how we locate Logic firmly in the Logic/Probability common heritage; it is based on former work by the author (Sales [1982a,b, 92,94,96]) and the starting point is Popper's 1938 suggestion (see Popper [1959]) to set forth a *unique* algebraic *uninterpreted* formalism as the common source from which, through distinct interpretations, both Logic *and* Probability can be formally derived as particular instances. The common formal idea we advance is, as will be later explained, that we can postulate an *additive valuation* (in e.g. [0,1]) of the elements of a given abstract Boolean structure $\mathcal{B}$ —that is later interpreted by Probability as a (set-theoretic) *event*, and by Logic as a (non-set) *sentence*. Our generalization proceeds from this point on as an exclusively *logical* reading of the common uninterpreted formalism. (The development is satisfactory also in a second, non-formal sense, since it can be seen as a vindication and reconstruction of the pioneers' historical common source of insights.)

A. The Probability/Logic Interface

1 THE VIEW FROM PROBABILITY

1.1 Classical Probability

Around the year 1700, Jakob Bernoulli tried to define the *probability* of a phenomenon as a non-evident and non-subjective something that fortunately had effects: the observable frequency, or relative number of cases in which the phenomenon manifested itself. This number was supposed fixed and objective. So, measuring frequencies was the way to estimate "probability". Conversely, knowing the probability of a phenomenon allowed to predict its expected frequency. This is, in essence, Bernoulli's theorem. It is the first clear, albeit implicit, definition of probability. It is also the first instance of a duality that is present since in Probability theory: probability P —a supposedly *objective* property of phenomena— is conceived simultaneously as (1) the ratio of positive cases (call it P_c) and (2) the number we have (call it P_b, b for 'belief') to estimate P_c. The first is assumedly an objective reality, the second an inevitably subjective entity that depends on our past history of observations (a paradox that is the common theme of many reflections, like those of e.g. de Finetti). Obviously, the fewer our interactions, the more subjective our P_b estimate is. The idea is that P_b "aproximates" P_c, and the aim is getting $P_b = P_c$ (in some limit situation).

The Rev. Bayes developed Bernoulli's idea of P_b converging to P_c through observational updates and came up with his celebrated formula (posthumously revealed in 1763) to compute P. Hailed by observational scientists for more than a century, it is now the heart of a debate about what is this Bayes-computed probability. Called "a priori" probability, anti-Bayesians contend it is nothing more than simple, non-objective belief based on a hypothetical view of our ignorance.

Laplace, in 1774 and later, defined probability as P_c, the ratio of favorable *cases*, all assumed having "the same probability". The obvious circularity raised some eyebrows in the 1920s, but Laplace's has been the standard and successful definition since, at least for non-sophisticated applications. Note that it places probability clearly on the *frequency* side, and cavalierly dismisses any subjective-sounding *belief* content, perhaps the reason for its long-standing success. Note, too, that *cases* are a logical notion, since they can be defined —and were by Laplace himself— as the *true* instances of a proposition. Thus, Laplace's [1774,1820] probability can be seen as an early generalization of Logic, particularly of the concept of validity (resp. consistency), now interpretable as "true in all (resp. some) cases". Some years before this, Hume had already implied too that probability was a generalization of logical inference by considering that, given a proposition

obtained from some conditions or premises, its probability was the proportion of premises (or premise extensions) in which the proposition was satisfied.

The classical probabilists thus conceived probability, more than as something associated with indeterminism or uncertainty, as a measure of our knowledge of phenomena in the presence of incomplete information (or, dually, as a measure of our partial ignorance). This concept was considered objective because, though not directly measurable, (1) it referred to a supposedly objective situation indirectly parameterizable through its observable effects, and (2) it was manipulable through rules of an objective calculus. So, after Laplace, Probability theory came to be dominated by the probability-as-frequency view. This was convenient as it was objective and "scientific" and adequately eschewed the estimation or "belief" problem.

It lasted until the 1920s, when von Mises, dissatisfied with the classical solution, formalized (in 1928) the estimation or approximation problem by postulating a "sample space" Ω, defining frequency in it and computing probability as the result of some limit process, in which the number of observations tended to infinity. Since this was no ordinary limit and the process not quite satisfactory, Kolmogorov [1933] came up with the now universally accepted solution: probability is just the *measure* of an *event* (an event being a set of outcomes); this measure is taken in the mathematical sense, i.e. as a countably additive valuation (though the need for countable additivity has been challenged by many, notably de Finetti [1970] or Popper [1959]). An interesting thing to note: Kolmogorov declared that his formalization was "neutral" in the sense that it was abstract (and thus previous to any interpretation); in his words, probability had to be formal, pure mathematics, merely ruled by axioms. Nevertheless he also declared that his measured entities (in theory merely the members of a σ-algebra over Ω) *are* actually sets. And though he added that this was irrelevant, Popper protested (in 1955) that it *is* relevant in some important cases, and noted that a truly abstract formalization must admit *any* interpretations, including those in which the measured entities are not sets. (But, we add, this is precisely the case of Logic, where we have non-set entities, namely *sentences* from a *language*, not *events* from a *sample space*.) Popper [1959] offered an axiomatic alternative first suggested in 1938, fully developed in 1955, and now fashionable (under the guise of "probabilistic semantics" or "Popper functions").

1.2 'Subjective probability", "Probability logic" and "Logical probability"

Following Keynes's [1921] lead, Ramsey [1926] was the first to consider that the belief side of probability, already present in Bernoulli or the Bayes' formula, was the core of the concept, since the "true" value of probability was

beyond our reach and the best we could do is approximate it through a careful, consistent, rational procedure; so he defined probability as *belief* and defined this as a number obtained on the basis of *consistency* considerations about the belief-holder's *rational* behavior (as deducible from betting protocols); the rationality-induced consistency insured that the number, though inevitably "subjective", was nevertheless the most objective measure one could obtain. In a similar spirit, and in same years, Bruno de Finetti [1931,37] approached probability as an inherently non-objective concept. He summarized it in his well-known slogan "probability does not exist" (i.e. objectively, at least "not more than the cosmic ether"), and nothing beyond consistency assures its imagined objectivity. According to him, probability is merely what we expect on the basis of past experience and the assumed consistency of what we do. As a number, probability (which de Finetti [1937,70] constructed formally on the basis of a vector space of rational expectations), is "objective" as far as the procedure to obtain it obeys coherent assumptions.

The "subjective probability" thesis of Ramsey/de Finetti has found continuation till now in the work of Jeffreys [1939], Koopman [1940], Savage [1954] or Jeffrey [1965], to name a few, all of which reject the epithet "subjective"; they prefer to be called simply probabilists and at most admit that the probability they deal with is a (non-subjective) partial or rational belief, i.e. the value we assign propositions in absence of complete information.

On the other hand, in a series of studies beginning in 1932 Hans Reichenbach [1935a,b], a physicist with an interest in foundations, interpreted probability as a logic. The logic (*probability logic* he called it) was not truth-functional, but he could subsume all classical tautologies as particular cases of propositions p that had unit probability (i.e. $|p| = 1$). He obtained formulas for the value (probability) of the connectives which are basically like the ones we obtain below; he says that e.g. $|p \vee q|$ is a function of $|p|$ and $|q|$ *plus* a third parameter k he calls Kopplungsgrad or 'degree of coupling' (defined roughly as the relative size of the intersection of overlapping areas or classes to which a measure is applied that coincides with the conditional probability). This is equivalent to what we obtain in our generalization below, but note that Reichenbach never moves out of probability and events: he always speaks of probability in its standard meaning and only in a translation of senses he says he can interpret the probability of an event sequence —a sequence of binary truth values— as its (non-binary) "truth". His world is clearly that of Probability, and what he obtains is a Logic only in the sense that he speaks of truth, albeit probability by another name. Moreover, though his contemporary critics (including Tarski [1935a]) argued against the construction, they did not because of the subsidiary role of truth in it (as a surrogate for a probability of one) but on the arguable ground that a proper logic ought to be truth-functional (see Urquhart's [1986] comment below as to the contrary).

(Work on 'probability logic' by Kemeny *et al.*(see Leblanc [1960]), Bacchus [1990], Halpern [1990] or Fagin et al. [1990] is not mentioned here because what these authors deal with is not what is understood by that name in the the classical tradition. Instead, they apply the standard treatment of any first-order theory *inside* Logic, i.e. augmenting ordinary first-order logic with a number of specific axioms that syntactically describe the real numbers (or an ordered field) and the probability operations on them, so that the Probability laws can be derived formally as theorems.)

Another indirect view of Logic-as-valuation was the one adopted by Rudolf Carnap [1950,62], starting in the 1940s. The Logic/Probability link began in his case by trying to justify probability logically. In his view (that he called *logical probability* and surmised as theoretical ground on which to base a "logic of induction" to which Popper came to be fiercely opposed), the probability of an event was the proportion or, more generally, the measure ("ratio of ranges of propositions") of the intervening circumstances (described as logical sentences) concurring in the event. For this measure he said he was inspired by a definition of Wajsberg —which was inspired in turn by proposition *5.15 of Wittgenstein's [1922] Tractatus (ultimately Bolzano-inspired, see below). Carnap hesitated and changed his approach often along the 1950s; for instance, notably, he came to value sentences instead of events, but came back to events later, shortly before giving up the whole scheme. Wittgenstein and Wajsberg's extensional rendition and Carnap's use of them is, like Reichenbach's implicit grounding of probability on rather obscure "overlapping classes", strongly reminiscent of the Stone representation we obtain below out of our general, non-probabilistic truth valuation of logical sentences. It is interesting to note that Carnap's logically-described components of events correspond rather precisely to what Laplace had called the (positive i.e. true) "cases" concurring in an event. They are also almost interchangeable with Boole's *cases* (his "conceivable sets of circumstances") underlying a logical proposition (or with equivalent descriptions by McColl and Peirce and Wittgenstein, see below).

2 THE VIEW FROM LOGIC

2.1 The pioneer logicians

The laplacian idea of having "cases" (a logical concept, we noted) and then measuring the proportion of the true ones seem to have been floating all over. Laplace's uninfluential contemporary, the Austrian philosopher Bolzano, had this to say about first-order propositions and truth: propositions, he says, have an associated "degree of validity", a number in [0,1] which equals "the proportion of true 'variants'" (Bolzano's "variants" are our term substitutions).

And then Boole [1854], when just after studying classes he sets out to analyze *propositions* (in 1847), conceives them by means of an alternative interpretation of his elective symbol x (already introduced for classes) and says it now stands for the *cases* (defined informally as "conceivable sets of circumstances") —out of a given hypothetical "universe" (a De Morgan's idea)— in which the proposition is *true*. In the last chapters of his 1854 book (significantly entitled "An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities") Boole even likens the product $x \cdot y$ of two propositions (i.e. the conjunction value, actually) to the *probability* of simultaneously having them both and the (value of) the sum $x + y$ to the *probability* of having either (provided they are mutually exclusive).

An equivalent idea is present in MacColl's [1906] partially published reflections (started before 1897), where he says that propositions are generally "variable", meaning they are sometimes the case, depending on their (basically probabilistic) modality. Peirce [1902], in an unpublished work that deliberately follows MacColl's steps, sets out to distinguish "necessary" from "contingent" propositions, most being the latter sort, characterized by their (probabilistic) occurrence.

In a similar vein Wittgenstein [1922] considers a little later that propositions reflect —and are basically decomposable into— "states of affairs" (an idea borrowed from Leibniz). That those states of affairs (reminiscent of Laplace's or Boole's cases) are in some way a measurable universe whose proportions gave information on the truth of the composite propositions is obvious from the Tractatus (and is the inspiration of the extensional view of Wajsberg and Carnap mentioned above).

2.2 'Multi-valued logic"

While some probabilists (from Ramsey to Reichenbach) agonized in the 1930s over their base concepts, there was intense soul searching also in the logicians' camp. The main new idea came from Łukasiewicz in 1930 (down from antecedents since 1918) when he postulated a logic in which "truth" values could take any value from the infinite real [0,1] interval (see Łukasiewicz & Tarski [1930]). To compute the value of composite propositions in his "many-valued logic" he obtained (truth-functional) formulas which are exactly the ones we obtain below except for the fact that they presuppose full *compatibility* (a concept we explain below) among *all* propositions, no matter some are the negation of others. This was not perceived as a problem at the time but *it was*, as some fellow logicians pointed out to him at the 1938 Zurich workshop (see Łukasiewicz [1938]). They considered that either we assign $p \wedge \neg p$ the value $\min(|p|,\ 1 - |p|)$ given by the formulas (thus blatantly contradicting the basic logical *law of non-contradiction*) or else we assign it the –correct– zero value (meaning *falsity*, so in accor-

dance with ordinary Logic) but then we arbitrarily disobey the postulated truth-functionality. As this predicament found no decisive solution, many-valued logic has continued to this day —along with unexpected offshoots like "fuzzy logic"— consecrating truth-functionality as a rigid principle and thus putting itself *out* of mainstream Logic, of which is a weak incompatible variety (unless we add, as suggested by van Fraassen [1968], some super-valuation mechanism to it). At the end of a detailed survey of multi-valued logic, Alasdair Urquhart [1986] comments that it is hardly surprising that those systems have remained logical toys or curiosities since "there seems to be a fundamental error [truth functionality] at the root". Some modern cultivators wonder whether is it possible to combine the two best-known formalisms, Probability and Logic, in any way (but Lee [1972] admits that "we do not seem how to do this"); others, like Hamacher [1976], Zimmerman [1977] or Trillas *et al.* [1982], in asking what are the correct truth-value formulas for the fuzzy calculi, hesitate among a variety of candidates, while still another, Minker (with Aronson *et al.* [1980]), would like to know the "truth bounds" of many-valued conclusions obtained from premises.

3 THE VIEW FROM THE OUTSIDE

3.1 Mathematics

Alfred Tarski straightforwardly supposed (in Tarski & Horn [1948]) he had simply a *Boolean algebra* and then set out to analyze thoroughly all possible *measures* in it. So did Gaifman [1962] —and Scott (with Krauss,[1965])— who extended this analysis to first-order logical formulas; these were assigned (additive) values, that were called 'measures' by Gaifman (and 'probabilities' by Scott). True to mathematicians' fashion (i.e. approaching topics in uninterpreted, "abstract" formalisms), they did not understand 'probability' as other people do; they just used the word as a synonym for *normalized measure* (a *measure* being a σ-additive valuation on the positive reals). In this sense, their "probability" is a blanket term for any common generalization —such as the one we attempt here— for the two (heavily interpreted) fields of Logic and Probability.

Also in this line, J. Łoś [1962] explored general "probability" valuations of logical *sentences* and came up with a (reasonably unsurprising) *representation theorem* of probabilities on a (set-theoretical) space of *models* (or *interpretations*) in the logical sense. Łoś's line has been consistently followed by Fenstad since 1965. (Fenstad's papers [1967,68,80,81] have been a source of inspiration for our generalization below.)

3.2 Philosophy

Philosophers have also been exploring the common material. A few representative examples are Hintikka and Suppes (both presenting first results in 1965), Stalnaker (in 1968–70), Lewis (in 1972) or Popper (e.g. in 1987, with Miller). The first (Jaakko Hintikka [1968]), inspired by Bar-Hillel's (and Carnap's,[1952]) information-measure ideas, was suggesting in 1965 (with Pietarinen,[1968]) various formulas to parameterize the information contained in sentences. Also in 1965, the second (Patrick Suppes [1968]) did a circumscribed analysis of the Modus Ponens logical inference rule from a generalized perspective (what he called 'probabilistic inference') in which he got formulas fully consistent with the ones we obtain below. Another philosopher traditionally preoccupied with logic/probability differences (especially those centered on the conditional/conditioning operation), Robert Stalnaker [1970], revealed some fine points (among which our "$A \to B$" $\neq$ "$B|A$" conceptual and practical distinction). His work and David Lewis's [1976] have done much to clarify and distinguish concepts shared by logicians and probabilists.

But, prior to these 1960s efforts, the single philosopher to do this most explicitly is surely Popper [1959], in lucid but little known pioneering work. He did not only see (in 1938) that the two concepts were different interpretations of a (yet to be written) formalism —Kolmogorov [1933] also saw this— but he designed one in a very simple and intuitive way by defining a valuation in [0,1] on pairs (a, b) of sentences (of a very elementary language) that was directly constructible by users (i.e. reasoners and probability-estimators alike) and that gave way naturally to a Boolean structure with the usual properties (including measurability). Whatever sense the user gave the valuation ("probability", "truth likelihood", "truth content" or simply "truth") it was the user's concern. Popper later used his own formalism (and the derived Booleanity assumption) to deduce properties of his 'truth content' measure and so emit (with David Miller,[1987]) a post-mortem indictment against Carnap's [1950,62] "inductive logic".

3.3 "Fuzzy Logic"

From a logical point of view, Fuzzy Logic (under development since 1965) can be considered as an "interpreted" variety of Łukasiewicz & Tarski's [1930] infinite-valued logic. ("Interpreted" because it adds to many-valued logic an extensional interpretation of predicates in terms of non-standard sets.) Thus, it was already mentioned in a former section, where we considered it as an (unexpected) offshoot of the many-valued logic family, and we dedicated it some short comments. Nevertheless, the overgrown "fuzzy" tradition, now largely applications-oriented, has its own self-contained rules and momentum and is *not* exactly logic nor probability. Nor, it claims, has

barely anything to do with them, with which it is pretendedly "orthogonal", only devoted to linguistically-motivated imprecision (i.e. *vagueness*).

One may doubt the claim by fuzzy theorists that the issues they currently discuss have no *logical* bearing. On the contrary, they seem fully relevant for logical discussion, so we have dedicated an appendix to comment rather expansively on fuzzy 'logic', as well as to mention an early generalization of Logic that arose inside the fuzzy tradition (by B.R. Gaines [1978]).

3.4 Artificial Intelligence

Since the moment the first "expert systems" dealt with uncertain information (the obvious cases are Mycin and Prospector), AI as a discipline got involved too in the Probability/Logic dilemma about what is the ultimate nature of "truth" measures of sentences presented to the expert system user (see Shortliffe [1976]). Leaving aside Mycin's "uncertainty factors" (later revealed to be actually measuring belief *change*, see Heckermann [1986]), the typical measures are Prospector's "*probability* assignments" (see Duda *et al.* [1976]), that are considered unproblematic and intuitive (to the user, who can easily estimate them), and are combined according to Bayes' formula as though they were really what their chosen name implied. Whatever the true status (probabilistic, or logical) of the calculus, the formulas on offer happen to become corollaries of our generalized calculus below (where, unlike in AI's rather *ad hoc* formalisms, nothing is assumed about whether the measures are actually "probabilities" or "truths" or something else).

Nils Nilsson's [1986] stated goal in his 'probabilistic logic' paper (orally anticipated in 1983) was to rationalize past work in the Prospector expert system project (1976-80) and give it a formal background by propounding a 'probabilistic entailment' that would do for this formalism what the Modus Ponens rule (A , $A \to B \ \vdash B$) does for ordinary Logic. He obtained the well-known bounds for the probability of B (see e.g. our formula (8) below) by Venn diagram techniques, that he extended to the study of convex hulls in a "probability space" of "possible worlds" (the latter terms are both familiar terminology to de Finetti and Łoś readers). As Nilsson [1993] acknowledged later, his method is similar to work by Good [1950] and Smith [1961] (not to mention de Finetti [1937,70]), authors of whom he was unaware at the time. His goal is, in fact, shared by many since the first 1980s —including the present author and others mentioned in previous and later paragraphs. (The Nilsson effort is briefly discussed in the next section.)

The Artificial Intelligence context has continued to breed practical motivation for the Logic/Probability demarcation. A series of special conferences (*'Uncertainty in A.I.'*) has been called (beginning in 1986, see Kanal & Lemmer [1986]) and given useful insights into the differences and similarity of the once-separate fields, including expert system coefficient analysis by Heckermann [1986], Grosof [1986] and others or the theoretical framework

called *belief networks*, developed for the efficient computation of "probabilities" (or whatever they are) by Judea Pearl [1988]. These contributions to general and practical Logic, worthwhile as they are, have the implicit bias that what is actually manipulated is the *probability* of distribution-driven events (rather than the *belief*, *commitment* or *assertiveness* of linguistic sentences), and thus the interpretations are always loaded with unnecessary concessions to probabilistic terminology and methods. So, for instance, Pearl, whose formalism is nominally about "beliefs", is nevertheless overwhelmed with computing probability distributions of *facts* and with assuming simplifying conditions (such as independence or conditioning) to obtain the final value; if this asignment is to be a real "belief", as stated, then presumably the "facts" and their distributional assumptions should be less real and objective than supposed: probably a consistent calculus (consistent in the Ramsey/de Finetti sense) based on possible or estimated (rather than actual) "facts" would suffice —and for this the possible-worlds or the rational-expectations analyses are already at hand (and ready to be usefully supplemented by a practical procedure such as Pearl's).

An unsuspected benefit of Logic-oriented analysis by Artificial Intelligence practitioners has been their growing awareness that a system of premises (what they tend to call "knowledge base") from which predictions are made (or actions are taken) is essentially a set of *beliefs* to which the agent is commited. This is now already clear in classic AI textbooks as Genesereth & Nilsson's [1987], where the distinction is made between inference procedures where the user's *full* commitment must be kept *throughout* the inference process and those where the belief premises are "qualified" (e.g. modally, with a belief operator) or "quantified" (with a "probability" assignment); in the latter cases, it is assumed, the commitment is less than absolute and the conclusion strength, therefore, less than guaranteed —however formally valid the reasoning may be. This approach is welcome, since it implies that however we treat premises in a logical argument — either as commitment-inducing beliefs or as admittedly weak probes— they all take part in the inference process and share with it a common goal: knowing to what extent can we *rely* on conclusions.

3.5 Probabilistic logic

As mentioned in the previous section, Nilsson [1986] sets out to investigate how Logic would generalize if one were to "assign probabilities instead of truth values to sentences". Though he calls his probabilities *probabilistic truth values* he treats them as real *probabilities* (at least to the extent that Prospector's numerical assignments are). This shows clearly in his subsequent treatment of (sentence) conditioning, that he considers plainly a Bayes process and relates to considerations by authors as Pearl, Heckermann, Grosof or Cheeseman (who explicitly deal with *probability* distribu-

tions). Based on entropy considerations by the latter author, Nilsson refines his bounding formulas so as to spot an exact value for the B in his 'probabilistic entailment' (= generalized Modus Ponens) rule —that happens to be the midpoint of the bounding interval (compare that with formulas (9-11) below).

Nilsson's grounding for his 'probabilistic logic' is basically semantic: he exhaustively generates and examines the "possible worlds" inherent in a formula; but the way he then discards some of the worlds —before assigning values to them— amounts to introduce consistency considerations. Akin to this method is what Paass [1988] proposes in a survey: assign basic "subjective probabilities", construct a universe of "relevant propositions" —which turns out to be isomorphic to Shafer's "frame of discernment" (or to our Θ below)— and then evaluate the resulting probability distributions on it. The computation may be done in Dempster-Shafer's terms (see Shafer [1976]) or by other methods: linear programming, stepwise simplification, Pearl's "belief networks" (with interactions) or statistical simulation (see Paass [1988]).

Though they may not use the name (invented by Nilsson), many so-called "probability logics" do not descend from Reichenbach but are really *probabilistic logics* and share Nilsson's conception and aim: finding a logical foundation for the use of [0,1] *probability* assignments to sentences taking part in an inference. Most of them formally derive their technical motivation and analysis from Gaifman [1962] and Scott & Krauss [1968]. The author of one of the first such 'probability logics', Theodore Hailperin [1984] —who also motivates his analysis historically (and also mentions the classics, from Bernoulli to Keynes and beyond)— sets out to generalize "truth" values and Logic in model-theoretical style, through the use of a modified version of the *model* and *consequence* concepts. This is done too by Bolc & Borowik [1992], who base their analysis on Scott & Krauss [1968] and Adams [1966], and by Gerla [1994] in an interesting attempt parallel to ours below.

4 BRIDGING THE GAP

4.1 Attempts at a synthesis

That the need for a generalization of Logic is widely felt, and that the time is now come to try it, is attested by the many surveys —and attempts at Logic/Probability synthesis (like the present one)— that are appearing of late (see for instance Gärdenfors [1988], Paass [1988], Garbolino *et al.* [1991], Bolc & Borowik [1992] or Gerla [1994]). But other such efforts deserve mention: Kyburg's [1993] is an exhaustive survey on logics of "uncertainty" where the author probably respects too much the usual division that insulates the surveyed authors' self-assigned topics, as he divides his

survey, too prolixly, in "objectivism", "subjectivism", Nilsson's "probabilistic logic", "belief functions", "measures", "probabilities", "statistical facts", "updating" and "inference", where the very exhaustivity gets in the way of a comprehensive attempt at synthesis. Similarly division-respecting are several 1995 drafts by Friedman & Halpern on plausibility measures, where the probabilistic bias dominates —in the terminology, in the chosen operations, etc.— though apparently the original intention was to widen "the measure" to a general "plausibility" concept. In the case of the swift and consistent work done by Dubois & Prade [1987,93], now a respected tradition, here the drawback to attain wide method-independent generality is the self-imposed limitation to (fuzzy) possibility measures —though some convergence with non-fuzzy approaches may occur in the future (see below on subadditivity). For the sake of completeness, we must add here work in progress by two researchers with a long tradition in trying to bridge the Probability/Logic gap: Richard Jeffrey [1995] and Glenn Shafer [1996].

4.2 Attempts at finding a meaning for the value

Confronted with the meaning that a "truth value" may have –or may be given– when extended to points in the [0,1] interval, different people have reacted in a number of ways. Here we mention only those who did not surrender to the temptation of subsuming truth value into *probability* (carrying with a it a heavily loaded interpretation of theory). Popper [1972] thought that, in terms of theories rather than sentences, truth value could be made to mean *truth content* (of the theory), degree of *approximation* to truth or, interchangeably, its (appropriately defined) *distance to falsehood*. Haack [1974] saw it could also be interpreted as *partial truth*, roughly defined as the proportion of true components of a sentence or theory, or –equivalently– the "truth" of their conjuntion. (An unwilling distant relative of Haack's is the quantum physicists' interpretation of the "truth" of a probabilistic quantum event sequence, which is similarly defined by reduction –conjunction, actually– to its elementary event components; see e.g. Reichenbach [1935a,b] or Watanabe [1969]). Dana Scott [1973] tried to answer by defining *truth value* as one minus the *error* we commit when ascertaining or deciding it, clearly in analogy with what we do in the observational sciences. Based on ideas advanced in the 1950s by Bar Hillel (with Carnap,[1952]), Johnson-Laird [1975,83] gave too a definition of truth value, albeit indirectly, by positing as a new concept the *informativeness* or "degree of information" of a sentence (a quantity negatively correlated with the "truth-table probability") to see how it evolves through the reasoning process, with an eye more on guiding the process than on controlling the *degree of truth* (or *assertiveness*, or whatever), which is what Logic puts the proper emphasis on.

4.3 Popper, and Probabilistic Semantics

As repeatedly mentioned above, Popper [1959] (in 1938 and 1955) had decided that Probability and Logic had to be given at last their long-overdue common formalism. Disagreeing with Kolmogorov's [1933] solution (a [0,1]-valuation on sets) because it was already (semi)interpreted —the valued objects were *sets*— and had a bias toward Probability, he proposed instead a totally abstract, uninterpreted system consisting of (1) an elementary algebra of "sentences" (not necessarily Boolean, merely closed by "conjunction" and "negation"), and (2) a [0,1]-valuation $v(a, b)$ on *pairs* of such sentences satisfying very basic and reasonable conditions (see the appendices in Popper [1959]). Such valuations, called "Popper functions", have become a vogue now (under the name of "probabilistic semantics"), following efforts by Harper [1975], Field [1976] or Leblanc [1979,83] to base ordinary (i.e. "unary") probability on it. Great advantages of the Popper formalism are:

- the formulas may be interpreted at will either "logically" or "probabilistically": when in the latter mode, the "sentences" are elementary *events*, the basic operations are *intersection* and *complementation*, and the valuation is just plain *conditional probability* (but with an unsuspected plus: there is no need that the "conditioning" event should be assigned a non-zero (unary) probability)

- there is no need for σ-additivity (as Popper [1959] himself bothers to show), nor is σ-additivity abstract enough for Kolmogorov's [1933] pretendedly neutral formalism to qualify as really neutral (since it is satisfied in an interpretation but discards certain others)

- the resulting quotient algebra *modulo* equi-valuation *is* automatically a *Boolean algebra*, which is not only simple and extraordinarily convenient but, because obtained from very simple assumptions, disarmingly *natural*

- each quotient algebra *class* is interpretable, at will, as an ordinary logical *sentence* or an ordinary probabilistic *event*, and its value turns out to be automatically its *truth value* or, respectively, its (so-called "unary", i.e. ordinary) *probability*

- the formalism being completely abstract (i.e uninterpreted) and the interpretation totally free, the "probability" may be –with equal legitimacy– subjective, objective or whatever; in particular it may be Popper's [1962] *truth content*, or its *probability* (the latter is, according to him, the value we give a theory when nothing is known about its *content*, which correlates *negatively* with it)

- likewise, the "truth value" may be *truth* or simple *belief*; in the present author's view, many other interpetations are equally legitimate, such as "degree of commitment", "assertive value" or any other that gives us information on the *reliability* —subsuming *truth*— of sentences (including premises and conclusion), and that allows us to have *control* over their behavior along a chain of reasoning

- if one does not accept Popper's simple conditions, the same result can provably be obtained by accepting alternative and equally simple conditions set forth –independently– by Cox [1961] (and, lately, also the ones by Woodruff [1995]).

As stated, "truth values" may be interpreted in various legitimate ways. Furthermore, any truly abstract formalism requires that they must. In the following sections —to the end of the article— we consider, in genuine Popperian fashion, several fully *logical* interpretations of the "truth value" concept (*belief*, *assertiveness*, etc.), all motivated by what should be included in any study of Logic: *invariance* (of *truth* —or of what the truth value stands for, be it *approximation* to truth, *reliability* or whatever else) along the whole *reasoning* process (assumed formally *valid*). But, compared to the standard Popper schema, our method proceeds just in the reverse direction: where Popper first defines the two-place probability function and then the unary probability is obtained by taking the quotient, we begin instead by a unary valuation and then we subsidiarily define conditioning (that we prefer to call "truth relativity" or "relative truth") to obtain Popper's basic two-placed function. The process inversion is unimportant, as we could as well have begun by a two-place valuation (of the *assertive value* —or whatever else— of a sentence *relatively* to the others) and then obtain its *absolute* assertive value by taking the quotient. And note that when we proceed in our direction rather than Popper's, evaluating the mutually *relative* position of sentences through the α and σ parameters (see below) amounts to just computing the basic two-place Popper function.

B. Steps toward a General Logic of Rationality

5 MOTIVATION

Let us advance what is our aim here by starting with an obvious remark. When we argue, we do not always fully assert what we say. We often make half-hearted assertions of sentences we are not sure about, or we even use as assertions sentences we hardly believe to be the case. And yet we proceed by reasoning from such weak premises. If we admit we do, and want to treat this inside Logic, we need to qualify assertions, or, if possible, to quantify

their strength, and try to follow and control what effects weak assertions may have in the reasoning process, whether and how they affect its logical validity and how we can tell the strength of the conclusion. All this is indeed a proper *logical* subject (that, however, classical logic never set out to confront).

By tradition, Logic is about *truth*; or, more precisely, it is about truth-preserving manipulations that allow us to validate arguments. *Arguments* are lists of sentences that we note by " $\Gamma \vdash C$ ", where C is called the *conclusion* and Γ is a –possibly empty, or infinite– list of sentences called *premises*. It is no obligation for sentences to be true (or even to have meaning). We merely use them to see whether certain formal manipulations —the *inference rules*— assure us that the prediction embodied by the conclusion C is *true* whenever the premises are. The whole process is dependent on the truth of the premises: if we cannot assure the truth of *all* of them, the whole procedure becomes redundant. This is how ordinary Logic approaches reasoning.

Now, at least two questions arise. First, suppose we are not sure whether some premise applies, yet we want to know the "truth" of C (or what is left of it, or, in other words, the *reliability* we can still attach to C as a prediction). Second, suppose that we deliberately want to *weaken* some true premise to see whether –or up to what point– the conclusion still *holds* (this is : probe the conclusion's dependence on its premises, or, in other words, the argument's "robustness"). By tradition, none of this is approached by Logic (so far). To see how we can generalize Logic to cover weak premises we must get a closer look on how we assign truth to sentences. In Logic the

base material is the *sentence*, say A. (Note A is *not* a set, but merely a member of a given language $\mathcal{L}$.) Once we interpret it we get what we can call a *proposition*. Then, by looking at what is the case, we get a value (a "truth value"). Following Tarski's [1935b] well-known schema

(T) 'A' is true if and only if A is true

the reasoner can verify the sentence (i.e examine the proposition A, obtained by "unquoting" the sentence A) and declare it true whenever the translation A of the object-language sentence A is found true. A is thus assigned the one or *true* value, and we say that A has full credibility and eventually we assert it with the full confidence that truth warrants. If we find A false, we assign it the zero or *false* value and give it null credibility. Note it is the reasoner who is full command of the sentences and the translation process, and thus the only one who can validate their truth. As sentences are used to *assert*, and assertions are defined as "true, believed or merely hypothesized" sentences, it is left to the reasoner who uses them to count actively on the quality of assertions as part of the reasoning process itself (so, for instance,

the reasoner usually qualifies the conclusions, conditioned on the strength the original assertions carried).

Now suppose we are reasoning in Physics and A (a premise) is the positive result of an experiment. If we are sure that A is true, then we are done: we can use it in a reasoning as a true premise and proceed with the assumedly valid argument to obtain the conclusion. But suppose that, as is usually the case in Physics, we have some qualms about the truth of A, so we quantify the error ε of the experiment. Now the "truth" of the assertion 'result is positive' is no longer 1 as before but, say, "$1-\varepsilon$". We then perform the formal —valid— reasoning. The question now is : what *confidence* —as a function $\varphi(\varepsilon)$ of ε— may we have in the conclusion?

Most reasonings are like this. We perform as though premises were really true, often unconvincedly. They are provably true sometimes, but most of the time they are 'assumed true' for the sake of the argument. Now, because the (T) validation process to declare a sentence true is under control of the user (who performs the translation and decides whether the unquoted statement is observed to be the case), so the reasoner is the only one who can qualify the truth with the appropriate provisos, according to the difficulties met in the validation (unquoting) process. It seems only natural to ask this user not only to qualify but to *quantify* (with a number in e.g.[0,1]) what is the degree of credibility (or belief) (s)he assigns it. The user *can* usually do it consistently (this is the "rational" behavior studied by Ramsey [1926]), thereby defining (by de Finetti's [1937] theorem) an *additive* valuation. (S)he can always assign the sentence A the *value* $v(A)$ —or, as we will write hereafter, $[\![A]\!]$—; this value ("truth value" we will call it) may be computed in an unspecified way (by betting preferences, belief networks, simulation, statistical survey, or whatever) and based on *any* preferred interpretation of A, be it standard *probability* of A as an event, or Popper's *truth likelihood* or *truth content* of A as a proposition or theory (that, as Popper found in the 1930s, is negatively correlated with its probability), or its *partial truth* in Haack's [1974] sense, or Shafer's [1976] *belief* (and its dual, *plausibility*), or its *reliability*, or the *credibility* of –or the (user's) *belief* in– A, or whatever (provided the assignment is done consistently). $[\![A]\!]$ represents a rough index of the *confidence* we have in A being the case and –consequently– the force with which we feel we can *assert* A in a particular argument (or the assertiveness we can commit into it). This measure is always possible, provided the user is "rational" in Ramsey's sense (but "non-rational" measures are also possible: they merely give *non*-additive values; see the final section, on subadditivity). What is measured is the user's *belief* in and *commitment* to A: a zero value means that the premise is to be taken as false, 1 means that it is a true (and therefore fully assertable) premise or –more often– that it is to be *assumed true* (and fully endorsed), and $v(A) = 1-\varepsilon$ $(\varepsilon > 0)$ means that we can assert A but with some apprehension or risk (that we assume) ε.

Now, approached in a most general way, the problem to solve is as follows. Suppose we have the reasoning $A_1, A_2, \ldots \vdash B$. Suppose we assign degrees of confidence or assertiveness to the premises. The question is : what will be the effect of those degrees in the confidence or reliability of B ? (We would thus probe the argument's robustness.) And what if we vary our confidence levels in some premises? Can it happen that, though the reasoning may be formally *valid*, the reliability of B turns out to be zero (thus making the argument *unsound*)? Or does B maintain its "truth" (or reliability = 1) though all the premises get null confidence themselves (thus making B's truth independent from the premises)? This analysis, like the physicist's $\varphi(\varepsilon)$ estimation problem above, *is* a legitimate logician's concern. (It is what we proceed to develop in our *proof theory* below.)

We illustrate this with an example. This is the well-known sorites about bald men: "If a man with i hairs is not bald then a man with $i-1$ hairs is still not bald. Suppose a man has n hairs. Therefore, a man with 0 hairs is still not bald". Formally:

$$\frac{\begin{array}{l} A_i \to A_{i-1} \quad (i : 1, \ldots, n) \\ A_n \end{array}}{A_0}$$

This is a paradox because the reasoning is formally correct (it consists of merely n applications of the Modus Ponens rule), the $n+1$ premises are deemed flawless, but the conclusion is outright false (or, more precisely, a *contradictio in terminis*). Usually, it is the length of the argument that is put to blame. There is, however, a more concrete and satisfactory answer we can offer. The n premises $A_i \to A_{i-1}$ cannot obviously be asserted with the same assurance whatever the index value. That's why the argument fails: for low values of i the premises simply cannot be asserted, even if the rest can, so we can *never* have all premises asserted, and the reasoning is formally valid but vacuously so.

Formally, what happens is that the value $[\![A_i \to A_{i-1}]\!]$ decreases with i, so that when i is n (or even, say, around $n/2$ or $n/3$) it is 1 or very near 1, but when i approaches, say, $n/10$ —and surely when it becomes zero— the value of $A_i \to A_{i-1}$ (= the predisposition we have to assert it —or the willingness to assume the risk) comes down to an exceedingly low number. According to a simple proof theory (that we describe below), the conclusion A_0 has the same truth value, at best, as that lowest of numbers (and, thus, the reasoner would be willing to assert the conclusion just no more than he or she would willing to assert $A_1 \to A_0$).

Note that, though we use words such as 'belief' or 'commitment' as surrogates for truth, *truth* itself is not merely reducible to belief (or probability): compare de Finetti's well-known position ("probability tells us only what to expect, not what will actually be the case") with the more recent comments of Cohen [1990] (a probabilist), who admits that when we say that

something is true with probability, say, .26, "this result tells us nothing about the truth" of the predicted fact or the postulated hypothesis, which will be –is, actually— true or not regardless of our (expectation-inducing) probability computations. This is not to say that Logic should continue to treat truth exclusively, as it now does. On the contrary, we contend that Logic, as it becomes a *general theory of rationality*, should center on a new object of study: the *assertive value* of sentences (that subsumes *truth* and which we will hereafter call –somewhat misleadingly– *truth value*), because this is what is really manipulated in arguments, and because this concept may let us analyze them in full generality, be it through weak premises, strong conclusions or argument robustness.

6 "TRUTH" VALUATIONS OVER SENTENCES

We assume we have a set $\mathcal{L}$ of sentences that form a Boolean algebra (with respect to the three connectives and two special sentences $\bot$ and $\top$). Now we have the whole Proof Theory of Sentential Logic by identifying the " $\vdash$ " order defined by the Boolean algebra with the deductive consequence relation. Thus the algebra of sentences we started with automatically becomes the Lindenbaum-Tarski algebra of all sentences modulo the interderivability relation " $\dashv\vdash$ " given by the $\vdash$ order (i.e. $A \dashv\vdash B$ iff $A = B$). We then assume that all sentences are valued in $[0,1]$, which we do in the standard way of a *normalized measure* $v : \mathcal{L} \longrightarrow [0,1] : A \mapsto [\![A]\!]$, by just requiring that $\top$ gets a value of 1 (1 is the only 'designated value' we consider) and that the valuation v is *additive* (i.e. $[\![A \vee B]\!] = [\![A]\!] + [\![B]\!] - [\![A \wedge B]\!]$). So we now have also the whole Model Theory of Sentential Logic.

This "truth" valuation is merely a (finitely aditive) *probability* in all technical senses, but here A is a *sentence* (in a language $\mathcal{L}$), not an *event* (in a sample space Ω). $[\![A]\!] = 1$, $[\![A]\!] = 0$ and $[\![A]\!] = 1/2$ here just mean – respectively– *truth* (or, more precisely, that "A is taken as truth"), *falsity* and *undecided belief* (when expressly *asserting* the A *sentence*); this is to be contrasted with (respectively) probabilistic "certainty", zero-probability or balanced odds (when evaluating the *uncertain* outcome of A as an event). We do not require that the valuations —even when interpreted fully as "truth" valuations— to be "extensional" or "truth-functional" as done in many-valued logics. As for the Booleanity of the sentences, either this is assumed (which is undemanding) or it derives from the "minimal algebra" of sentences suggested by Popper [1959] —or from provably equivalent simple assumptions (e.g. by Cox [1961] or Woodruff [1995]). Also, the additive character of the valuation amounts to having a 'rational' (Ramsey [1926]) or 'coherent' (De Finetti [1937,70]) *belief*, a concept so akin to 'strength of assertion' in Logic as to be all but exchangeable.

From the Booleanity of $\mathcal{L}$ and the above properties of the v valuation we

immediately obtain:

$$[\![\neg A]\!] = 1 - [\![A]\!] \tag{1}$$

$$\begin{aligned} [\![A \wedge B]\!] &\leq \min([\![A]\!], [\![B]\!]) \\ [\![A \vee B]\!] &\geq \max([\![A]\!], [\![B]\!]) \\ [\![A \to B]\!] &= 1 - [\![A]\!] + [\![A \wedge B]\!] \\ [\![A \leftrightarrow B]\!] &= 1 - [\![A \vee B]\!] + [\![A \wedge B]\!] \end{aligned} \tag{2}$$

7 SENTENCES AS SET EXTENSIONS, AND TRUTH AS MEASURE

Any Boolean Algebra has a *representation* on a set structure (a field of sets) as Stone proved long ago in a famous theorem (see, for example, Koppelberg *et al.* [1989]). Thus, given the Boolean sentence algebra $\mathcal{L}$, there exist both a set Θ (whatever the meaning we give to its elements θ) *and* a 'representation' function that can be characterized as an isomorphism of $\mathcal{L}$ into the Boolean subalgebra $\mathcal{B}$ of clopens in $\mathcal{P}(\Theta)$, i.e.

$$\rho : \mathcal{L} \longleftrightarrow \mathcal{B} : A \mapsto \mathbf{A} \quad (\mathcal{B} \subset \mathcal{P}(\Theta), \mathbf{A} \subset \Theta).$$

We call the members of Θ *possible worlds*, or *cases* (as Laplace [1774] or Boole [1854]) or *possibilities* (Shafer [1976]) or even *observers*, *states*, etc. Θ is the *universe of discourse* or *reference frame* (the set of *possible worlds*). It coincides with Fenstad's [1968] *model space* (where the θs are *interpretations* in the standard logic sense).

We can establish a general, one-to-one correspondence between the two worlds (the language world $\mathcal{L}$ and the referential universe Θ, both made up of "propositions") and their constituent parts, thus:

$\mathcal{L}$	$\Longleftrightarrow$	$\mathcal{B}$
$A (A \in \mathcal{L})$	$\Longleftrightarrow$	$\mathbf{A} (\mathbf{A} \subset \Theta)$
$A \wedge B$	$\Longleftrightarrow$	$\mathbf{A} \cap \mathbf{B}$
$A \vee B$	$\Longleftrightarrow$	$\mathbf{A} \cup \mathbf{B}$
$\neg A$	$\Longleftrightarrow$	$\mathbf{A}^c$
$\top$	$\Longleftrightarrow$	Θ
$\bot$	$\Longleftrightarrow$	$\emptyset$
$A \vdash B$	$\Longleftrightarrow$	$\mathbf{A} \subset \mathbf{B}$

If $\mathcal{L}$ has a finite number of generators, then it has 2^n atoms a and the two bijective correspondences $\mathcal{L} \Longleftrightarrow \mathcal{P}(\Theta)$ and $a \Longleftrightarrow \{\theta\}$ also hold.

The valuation v and the representation isomorphism ρ induce a [0,1]-valued measure μ in $\mathcal{B} \subset \mathcal{P}(\Theta)$, in such a way that $\mu = v \circ \rho^{-1}$, i.e. $\mu(\mathbf{A}) = [\![A]\!]$.

Intuitively, the measure $\mu(\{\theta\})$ of each individual θ in a *finite* Θ universe is the relevance or the degree of realizability of the given possible world. The μ measure corresponds to the weighing function λ in Fenstad's [1968] *model space*. As it is known, μ (or λ) is not only additive but –by the compactness property– countably so; thus μ is eligible as a standard "probability" measure (in the technical sense).

8 CONNECTIVES AND SENTENTIAL STRUCTURE

If we want to compute the truth value of composite sentences, the task is easy. For the negation connective, the formula is given by (1) above. For the binary sentences composed of A and B we have the formulas below, where we observe that, besides $[\![A]\!]$ and $[\![B]\!]$, we now need a third parameter that we note " α_{AB} " and that we call "*compatibility* between A and B"; its value is defined by $\alpha_{AB} =_{df} 1 - \beta_{AB}$, where:

$$\beta_{AB} = \frac{\min\,([\![A]\!],[\![B]\!]) \;-\; [\![A \wedge B]\!]}{\min\,(\,[\![A]\!],\;[\![B]\!],\;1-[\![A]\!],\;1-[\![B]\!]\,)} \quad .$$

We call β_{AB} the "*degree of incompatibility* of sentences A and B" and we rename the denominator by calling it "Δ_{AB}". Then, the formulas for the connectives are:

$$\begin{array}{lcl}
[\![A \wedge B]\!] & = & \min([\![A]\!],[\![B]\!]) \;-\; \beta_{AB} \cdot \Delta_{AB} \\
[\![A \vee B]\!] & = & \max([\![A]\!],[\![B]\!]) \;+\; \beta_{AB} \cdot \Delta_{AB} \\
[\![A \to B]\!] & = & \min(1, 1 - [\![A]\!] + [\![B]\!]) \;-\; \beta_{AB} \cdot \Delta_{AB} \\
[\![A \leftrightarrow B]\!] & = & 1 - \;|\,[\![A]\!] - [\![B]\!]\,| \;-\; 2 \cdot \beta_{AB} \cdot \Delta_{AB}
\end{array}$$

So, by knowing a single value (either of $[\![A \wedge B]\!]$, $[\![A \vee B]\!]$, $[\![A \to B]\!]$, $[\![A \leftrightarrow B]\!]$, α_{AB} or β_{AB} —or $[\![A|B]\!]$ or $[\![B|A]\!]$, see below) we can compute, via α_{AB} (or β_{AB}), the other seven. The parameter α_{AB} (which is a modern version of Reichenbach's [1935b] "Kopplungsgrad") acts as an indicator or measure of the "relative position" of $\mathbf{A}$ and $\mathbf{B}$ inside Θ, while "Δ_{AB}" is a quadruple minimum that only depends on the values of $[\![A]\!]$ and $[\![B]\!]$. Note that if we suppose that $\beta_{AB} = 0$ for *all* A and B then the above formulas are

$$\begin{array}{lcl}
[\![A \wedge B]\!] & = & \min([\![A]\!],[\![B]\!]) \\
[\![A \vee B]\!] & = & \max([\![A]\!],[\![B]\!]) \\
[\![A \to B]\!] & = & \min(1, 1 - [\![A]\!] + [\![B]\!]) \\
[\![A \leftrightarrow B]\!] & = & 1 - |[\![A]\!] - [\![B]\!]|
\end{array}$$

and coincide with those given in ordinary *many-valued logics*. Instead, if $\beta_{AB} = 1$, the connectives are

$$\begin{aligned}
[\![A \wedge B]\!] &= \max(0, [\![A]\!] + [\![B]\!] - 1) \\
[\![A \vee B]\!] &= \min(1, [\![A]\!] + [\![B]\!]) \\
[\![A \to B]\!] &= \max(1 - [\![A]\!], [\![B]\!]) \\
[\![A \leftrightarrow B]\!] &= |[\![A]\!] + [\![B]\!] - 1|
\end{aligned}$$

and coincide with those given in *threshold logics*.

9 RELATIVE TRUTH

Now suppose we want to express the conjunction value as a product:

$$[\![A \wedge B]\!] = [\![A]\!] \cdot \tau.$$

With the current $\mathcal{L}/\mathcal{P}(\Theta)$ representation in mind, we obtain:

$$(3) \quad \tau = \frac{\mu(\mathbf{A} \cap \mathbf{B})}{\mu(\mathbf{A})}$$

We define τ as the *relative truth* " $[\![B|A]\!]$ " (i.e. the "truth of B relative to A"). Though this definition exactly parallels that of conditional probability, the account we give leaves out any probabilistic interpretation of the concept and retrieves it for exclusively logical contexts.

In particular, if $[\![B|A]\!] = [\![B]\!]$ then we say that A and B are *independent*. In that case, the conjunction can be expressed as the product:

$$[\![A \wedge B]\!] \;=\; [\![A]\!] \cdot [\![B]\!].$$

In any other case we say that A and B are mutually *dependent* and speak of the *relative truth* of one with respect to the other. Note the dependence goes both ways and the two situations are symmetric. We have (assuming $[\![A]\!] \neq 0$):

$$[\![A]\!] \cdot [\![B|A]\!] = [\![B]\!] \cdot [\![A|B]\!] = [\![A \wedge B]\!] \quad \text{(a logical } \textit{"Bayes formula"}\text{)}$$
$$[\![B|A]\!] = 1 - \tfrac{1 - [\![A \to B]\!]}{[\![A]\!]}.$$

From the latter, note that, in general,

$$[\![B|A]\!] \;\neq\; [\![A \to B]\!].$$

Particularly, we have *always*

$$[\![B|A]\!] \;<\; [\![A \to B]\!]$$

except when either $[\![A]\!] = 1$ or $[\![A \to B]\!] = 1$, in which cases (and they are the *only* ones) $[\![B|A]\!] = [\![A \to B]\!]$. (These facts have been repeatedly noticed by many people, notably by Reichenbach [1935b], Popper [1959], Stalnaker [1970] or Lewis [1976].)

When two sentences A and B are *independent* then (and this is a necessary and sufficient condition for that to happen): $\alpha_{AB} = \max([\![A]\!], [\![B]\!])$ if $[\![A]\!] + [\![B]\!] \leq 1$ $= \max([\![\neg A]\!], [\![\neg B]\!])$ if $[\![A]\!] + [\![B]\!] \geq 1$ and then the connectives obey the formulas

$$\begin{aligned} [\![A \wedge B]\!] &= [\![A]\!] \cdot [\![B]\!] \\ [\![A \vee B]\!] &= [\![A]\!] + [\![B]\!] - [\![A]\!] \cdot [\![B]\!] \\ [\![A \to B]\!] &= 1 - [\![A]\!] + [\![A]\!] \cdot [\![B]\!]. \end{aligned}$$

The statement '$A \to B$' can have, among other readings, one logical ("A is sufficient for B" or "B is necessary for A"), another (loosely) "causal" ("A occurs and B follows"). But because $A \to B$ is valued in [0,1], its value $[\![A \to B]\!]$ (and the values $[\![B|A]\!]$ and $[\![A|B]\!]$) now mean only *degrees*, and so $B \to A$ may be —and usually is— read "evidentially" ("B is evidence for A"). Within such a frame of mind,

- $[\![B|A]\!]$ (or "$\sigma_{A(B)}$") could be termed "degree of *sufficiency* or *causality*" of A (or "*causal support* for B"), to be read as "degree in which A is sufficient for B" or "degree in which A is a cause of B". In view of (3), it is roughly a measure of how much of **A** is contained in **B**.
- $[\![A|B]\!]$ (or "$\nu_{A(B)}$") could be termed "degree of *necessity*" or "*evidence*" of A (or "*evidential support* for A", to be read as "degree in which A is necessary for B" or "degree in which B is evidence (= support of hypothesis) for A (=the hypothesis)". With (3) in mind, it can be seen as how much of **B** overlaps with **A**.

Such measures may be directly estimated by experts, normally by interpreting the θs frequentially, in terms of *cases*, like Boole [1854], *possibilities* (Shafer [1976]), *elementary events* in Θ, or possible *interpretations*; at any rate, they may be statistically-based or simply imagined, presumably on the basis of past experience or sheer plausibility. Thus, $\sigma_{A(B)}$ in a causal reading of "$A \to B$" would be determined by answering the question: "How many times (proportionally) —experience shows— A occurs and B follows?" For $\nu_{A(B)}$, the question would be: "How many times effect B occurs and A has occurred previously?" (Similarly for the evidential reading of "$A \to B$".) Once σ and ν have been guessed, they may be adjusted (via the

$$\frac{\sigma_{A(B)}}{\nu_{A(B)}} = \frac{[\![B]\!]}{[\![A]\!]}$$

relation) and then lead —by straightforward computation— to $[\![A \to B]\!]$, $[\![B \to A]\!]$ and α_{AB}, which allows one to compute all other values for connectives and also to get a picture of the structural relations linking A and B. This process of eliciting the σ value for every $\langle A, B\rangle$ sentence pair is closely equivalent to computing Popper's *binary probability* function.

Note first that a similar computing process takes place in "Bayesian reasoning", and in the "approximate reasoning" methods as implemented in marketable expert systems, though none of them satisfactorily explains on what *logical* grounds are the procedures justified, nor can they avoid supplying a (nominally) *probabilistic* account of them. Such an account would be here clearly misplaced, since there is usually neither (a) a sample space of *events*, but a language of *sentences* (i.e. linguistic descriptions —not necessarily of any "event"), nor (b) a measure based on *uncertainty* and *outcomes* (the topics Probability is supposed to deal with) but rather simple *beliefs* or, at most, mere *a priori* estimates, nor (c) an adequate updatable statistical or probabilistic basis to compute the values of the "events" (and their ongoing, dynamical change).

Note also that any reasoning can proceed here in both directions (from A to B and from B to A), because both conditionals $A \to B$ and $B \to A$ claim a non-zero value and thus a "causal" top-down reasoning can be complemented by an "evidential" bottom-up reasoning on the same set of given sentences (as also happens in Pearl's [1988] belief networks).

10 THE GEOMETRY OF LOGIC: DISTANCE, TRUTH LIKELIHOOD, INFORMATION CONTENT AND ENTROPY IN $\mathcal{L}$

The fact that we have:

$$[\![A \leftrightarrow B]\!] = 1 - ([\![A \vee B]\!] - [\![A \wedge B]\!])$$

strongly suggests using $1 - [\![A \leftrightarrow B]\!] = [\![A \vee B]\!] - [\![A \wedge B]\!]$ as a measure of the *distance* $\overline{AB}$ (under a given valuation v). So we do. (We remark that all definitions we give here of distance and related concepts are not only applicable to sentences but to *theories* as well, because for a general lattice $\mathcal{L}$ the lattice $\hat{\mathcal{L}}$ of theories derived from each sentence in $\mathcal{L}$ is isomorphic to $\mathcal{L}$.)

DEFINITION 1. *Distance* (or *Boolean distance*) between two sentences or theories A and B is:

$$\begin{aligned} d(A,B) &=_{df} 1 - [\![A \leftrightarrow B]\!] \\ &= [\![A \vee B]\!] - [\![A \wedge B]\!] \\ &= |[\![A]\!] - [\![B]\!]| + 2 \cdot \beta_{AB} \cdot \Delta_{AB} \end{aligned}$$

DEFINITION 2. *Compatible distance* between two sentences or theories A and B is:

$$d^+(A,B) =_{df} |[\![A]\!] - [\![B]\!]| \;=\; [\![A]\!] + [\![B]\!] - 2\,\min([\![A]\!],[\![B]\!])$$

We now define a *truth likelihood* value for A —approximating Popper's [1972] (and Miller's [1978]) *truth likelihood* or *verisimilitude* measure— by making it to equal the distance from A to falsehood, i.e. $d(A,\bot)$. We obtain, immediately:

$$\begin{aligned} d(A,\bot) &= d(\top,\bot) - d(\top,A) = 1 - d(A,\top) \\ &= 1 - d(A \bigtriangleup \top, \bot) \\ &= 1 - d(\neg A,\bot) = 1 - [\![\neg A]\!] = [\![A]\!] \end{aligned}$$

So here we have a further interpretation of our "truth values" $[\![A]\!]$ in terms of Popper's [1972] *truth likelihood* or *verisimilitude*. We might as well consider $[\![A]\!]$ as a rough measure of *partial truth* or *truth content* of A. In a similar vein, we may recall that Scott [1973] suggested the "truth value" $[\![A]\!]$ of many-valued logics could be interpreted as *one* (meaning *truth*) less the *error* of A (or rather of a measure settling the truth of A) or the *inexactness* of A (as a theory); in this framework, it comes out that, in our terms, $[\![A]\!] = 1 - \varepsilon_A$ and $\varepsilon_A = 1 - [\![A]\!] = d(A,\top)$.

We further observe that, for any sentential letters P and Q, any uniform truth valuation yields $[\![P]\!] = [\![\neg P]\!] = .50$, $[\![P \wedge Q]\!] = .25$ and $[\![P \vee Q]\!] = .75$, which is like saying that, if all letters are equiprobable, the given values are the probability of the given sentence being true (a number that Johnson-Laird [1975,83] calls, appropriately, "truth-table probability"). This value's complement to one is reasonably made to correspond to the amount of information —in a loose sense— we have when the sentence is true. This is precisely what Johnson-Laird [1975,83] defines as "degree of information", "semantical information" or *informativeness* $I(A)$ of a sentence A. (Viewed in our terms, this *information content* $I(A)$ equals $1-[\![A]\!]$, or $I(A) = [\![\neg A]\!] = d(A,\top) = \varepsilon_A$.) The concept, based on Bar-Hillel's (and Carnap's,[1952]) ideas, was originally designed to model the reasoning process, assumed to be driven by an increase both of informativeness and parsimony. What is interesting is that the informativeness of composite sentences is computed by combining them according to non-truth-functional rules which yield values that coincide with those predicted by our formulas.

A new measure we can define, which can also be used as an *entropy* (in the sense of De Luca & Termini [1972]), is this one:

DEFINITION 3. *Imprecision* (or, perhaps, "fuzziness") of a sentence (or theory) A is the value for A of the function

$$f : \mathcal{L} \longrightarrow [0,1] \text{ such that } f(A) = 1 - d^+(A,\neg A)$$

It is immediate that:

$$f(A) = 2 \min ([\![A]\!], 1 - [\![A]\!]),$$

which is equivalent to saying that the imprecision of a sentence A equals twice the error we make when we evaluate on A the truth of the law of non-contradiction or of the excluded middle by considering there really *is* maximum compatibility between A and $\neg A$. (Actually, there is *null* compatibility, as α_{AB} is provably *zero* when B is $\neg A$.) In connection with this measure, we note in passing that:

- *classical* (two-valued) *logic* is the special case of ours in which *all* sentences in $\mathcal{L}$ have *zero* imprecision.

- ordinary *multi-valued logics* —like Łukasiewicz-Tarski's $Ł_\infty$— are the ones where at least *one* sentence in $\mathcal{L}$ has non-zero imprecision. This measure being —as it is— an error function, imprecision is here just the degree in which these logics fail to distinguish contradictions (their lack of "resolving power") .

11 ELEMENTARY PROOF THEORY FOR GENERAL ASSERTIONS

Once we have valued sentences in $[0, 1]$ we now need a Proof Theory that in the most natural way extends standard logic so as to treat imprecise statements or weak assertions, and measure and control whatever effect they may have on reasoning (as well as to explain some results in approximate reasoning methods from Artificial Intelligence).

To begin with, suppose a valid argument, noted $\Gamma \vdash B$ (where Γ are the premises, or a finite subset of them). Classical logic declares it *valid* if B is derivable from Γ in an appropriate deduction calculus. By the completeness property, this amounts to assert the truth of B whenever the premises in Γ are true. Now, as we said, the ultimate judge of the truth of the premises is the reasoner. It is the reasoner who decides that each premise used is true (or to be considered true). To justify such a decision, the reasoner applies a truth criterion such as Tarski's [1935b] (T) schema. Thus the reasoner declares A true when assured that what A describes is precisely the case. If the reasoner is not sure of the result of his/her validation or does not want to commit him/herself to it, then the reasoner may choose not to make a full assertion by claiming that A's verification does not yield an obvious result. In that case, the reasoner may rather easily "qualify" the assertion by assigning numbers in $[0, 1]$ such as $v(A)$ —that we noted "$[\![A]\!]$"— or $\varepsilon(A)$ $[\, = 1 - v(A)\,]$ meaning that the reasoner believes or is willing to assert A to the degree $v(A)$ or assume it with a risk or estimated error of $\varepsilon(A)$.

The proof theory we now sketch is a slightly extended version of the standard one. Here we understand by *proof theory* the usual syntactical deduction procedures *plus* the computation of numerical coefficients that we must perform alongside the standard deductive process. We do that because a final value of zero for the conclusion would invalidate the whole argument as thoroughly as though the reasoning were formally —syntactically— invalid. As always, any formally valid *argument* will have, by definition, the following *sequent* form:

$$\Gamma \vdash B$$

where B is the conclusion and Γ stands for the list —or, rather, the conjunction— of the premises (or, by the compactness property, of a finite number of them). We have, elementarily:

(4) $\quad \Gamma \vdash B \quad \Rightarrow \quad [\![\Gamma]\!] \leq [\![B]\!]$.

We henceforth assume that we have a *valid argument* (so $\Gamma \vdash B$ will always hold), and that all premises are non-zero (i.e. $\forall i \ [\![A_i]\!] > 0$). We distinguish four possible cases:

1. $[\![\Gamma]\!] = 0$ (i.e. the premises are –materially– *inconsistent*). Here by (4) $[\![B]\!]$ can be anywhere between 0 and 1; this value is in principle undetermined, and uncontrollably so.

2. $[\![B]\!] = 0$. This entails, by (4), $[\![\Gamma]\!] = 0$ and we are in a special instance of the previous case. The reasoning is formally valid, the premises are not asserted, and the conclusion is false.

3. $[\![\Gamma]\!] \in (0, 1)$ (i.e. the premises are consistent). Then, by (4), $[\![B]\!] > 0$. We have a formally valid argument, we risk assessing the premises (though with some apprehension) and get a conclusion which can be effectively asserted though by assuming a –bounded– risk. This will be the case we will set to explore below.

4. $[\![\Gamma]\!] = 1$. This condition means that $[\![A_1]\!] = \ldots = [\![A_n]\!] = 1$ and, by (4), $[\![B]\!] = 1$. So the premises are *all* asserted —with no risk incurred— and the conclusion *holds* inconditionally (remember $\Gamma \vdash B$ is formally valid). This is the classical case studied by ordinary two-valued Logic.

We are interested in examining *case 3* above, i.e. formally valid reasoning *plus* assertable premises (though not risk-free assertions) *plus* assertable conclusion (but at some measurable cost). Cases 3 and 4 characterize in a most general way all *sound* reasoning. Case 2 characterizes *unsound* arguments (since in this case having a formally valid argument $\Gamma \vdash B$ does

not preclude getting an irrelevant conclusion ($[\![B]\!] = 0$). As this case is the one to avoid, we have:

DEFINITION 4. *Unsoundness* of a valid argument $\Gamma \vdash B$ is having $[\![B]\!] = 0$ though the premises are themselves non-zero. So:

DEFINITION 5. *Soundness* of a valid argument $\Gamma \vdash B$ is having $[\![B]\!] > 0$ whenever the premises non-zero.

With that in mind, we can now turn to the basic inference rule, the Modus Ponens (MP). From a strictly logic point of view, this rule is

$$\begin{array}{ll} A & m \\ A \to B & n \\ \hline B & p \end{array} \tag{5}$$

where m, n and p stand for the strength or force (or "truth value") we are willing to assign each assertion; so, in our terms, m, n and p are just our $[\![A]\!]$, $[\![A \to B]\!]$ and $[\![B]\!]$. They are numbers in [0,1] that take part in a (numerical) computation which parallels and runs along the logical, purely syntactical deduction process. This is well understood and currently exploited by reasoning systems in Artificial Intelligence that must rely on numerical evaluations —given by users— that amount to *credibility* assignments (or "certainty factors"), *belief* coefficients, or even —rather confusingly— *probabilities* (often just *a priori* probability estimates); this is the case of successful *expert systems* such as Prospector or Mycin. The trouble with such systems is that they tend to view Modus Ponens as a probability rule (this is made explicit in systems of the Prospector type, see Duda *et al.* [1976]). They use it to present the MP rule in this way:

$$\begin{array}{l} A(m) \\ A \to B(\sigma) \\ \hline B(p) \end{array} \tag{6}$$

where m and p are the 'probability' (a rather loose term here) of A and B, and "$A \to B(\sigma)$" means that "whenever A happens, B happens with probability σ". Here σ turns out to be just $v(B|A)$ or "$[\![B|A]\!]$", the "relative truth" of B given A; this concept, modelled on a close analogy —*ceteris paribus*— with that of (probabilistic) conditioning, is what we have defined above (in (3)) and called "degree of sufficiency" σ of A —or of necessity of B—, assumed easily elicitable by experts. So it is just natural, and immediate, to compute the p value thus:

$$p \geq \sigma \cdot m$$

or, in our notation,

$$[\![B]\!] \geq [\![B|A]\!] \cdot [\![A]\!]$$

which is just another version of formula (2).

The problem is that what we have, from our purely logical, probability-rid standpoint, is (5), not (6), and in (5) n is not $[\![B|A]\!]$ but $[\![A \to B]\!]$. Recall that $[\![B|A]\!]$ and $[\![A \to B]\!]$ not only do *not* coincide but mean different things (as repeatedly noticed by logicians, and as explained above). Indeed, $[\![A \to B]\!]$ is the value ("truth" we may call it, or "truth minus risk") we assign to the (logical) assertion $A \to B$. Instead, $[\![B|A]\!]$ is a relative measure linking materially, *factually*, A and B (or, better still, the **A** and **B** sets), with no concern whether a true *logical* relation between them exists; we might even have $[\![B|A]\!] < [\![B]\!]$, thereby indicating there exists an *anti*correlation (thus rather contradicting any —logical or other— reasonable kind of relationship between A and B). So we turn back to our (5) rule; note that $m + n \geq 1$ (this *always* holds) and that $[\![B|A]\!]$ can be obtained from $[\![A \to B]\!]$, or vice versa: $[\![A \to B]\!]$ from $[\![B|A]\!]$ through

$$(7) \quad [\![A \to B]\!] \;=\; 1 - [\![A]\!] \cdot (1 - [\![B|A]\!])$$

(which is useful, since $[\![B|A]\!]$ is directly obtainable from experts).

The following theorem states the *soundness* condition for the MP rule.

The Modus Ponens rule

$$\begin{array}{ll} A & m \\ A \to B & n \\ \hline B & p \end{array} \quad \text{(we assume } m \text{ and } n \text{ are both non-zero)}$$

is *sound* (and thus $[\![B]\!] \neq 0$) if one of these four equivalent conditions hold:

1. $m + n > 1$
2. $[\![B|A]\!] > 0$
3. $[\![A \wedge B]\!] > 0$
4. Either $[\![A]\!] + [\![B]\!] > 1$ (and thus $[\![B]\!] > 1 - m$) or both A and B are compatible ($\alpha_{AB} > 0$) and not binary-valued.

In both sound and unsound cases we have the following easily computable bounds for the value $[\![B]\!]$ of the MP conclusion (Sales [1992,96]):

$$(8) \quad [\![A]\!] + [\![A \to B]\!] - 1 \;\leq\; [\![B]\!] \;\leq\; [\![A \to B]\!]$$

or equivalently, in shorter notation:

$$m + n - 1 \;\leq\; p \;\leq\; n \;.$$

(Such bounds have been discovered again and again by quite diverse authors; see e.g. Genesereth & Nilsson [1987]). The lower bound —which equals $[\![A \wedge B]\!]$ — is reached when $\alpha_{AB} = 1$ and $[\![A]\!] \geq [\![B]\!]$, while the upper bound is reached when $\alpha_{AB} = 0$ and $[\![A]\!] + [\![B]\!] \geq 1$. Naturally we usually know neither $[\![B]\!]$ nor α_{AB} beforehand, so we don't know whether the actual value $[\![B]\!]$ reaches either bound or not, nor which is it; we can merely locate $[\![B]\!]$ inside the $[m + n - 1, n]$ interval.

But this interval can be narrowed. Since a very reasonable constraint a conditional $A \to B$ may be expected to fulfill is that A and B be (assumedly) non-independent –and not binary– and *positively* correlated (i.e.: $[\![A \wedge B]\!] > [\![A]\!] \cdot [\![B]\!]$), so we have:

$$(9) \quad [\![A]\!] + [\![A \to B]\!] - 1 \quad \leq \quad [\![B]\!] \quad < \quad [\![B|A]\!].$$

Here, if A and B are fully or strongly compatible, $[\![B]\!]$ will be nearer the lower bound. Thus, we can only increase our $[\![B]\!]$ if we are assured that A and B are independent (in the sense that $[\![A \wedge B]\!]$ equals $[\![A]\!] \cdot [\![B]\!]$, see above): we then obtain the highest value $[\![B]\!] = [\![B|A]\!]$. On the other hand, the more we confide instead in a strong logical relation between A and B, the more we should lean towards the low value given by

$$(10) \quad [\![B]\!] = [\![A \wedge B]\!] = m + n - 1.$$

Under very reasonable elementary hypotheses (like this one: $[\![A \to B]\!] > [\![A \to \neg B]\!]$ or, equivalently, $\sigma_{A(B)} > 1/2$, see Sales [1996]), we easily get these bounds for $[\![B]\!]$: $[\![A]\!]/2 \; < \; [\![B]\!] \; \leq \; [\![A]\!]$.

Now, if what we want is not an interval, however narrowed, but a precise value for $[\![B]\!]$ we should favor the lower value, the one given by (10) above. There are lots of reasons (some mentioned in Sales [1996]) for this choice of value in the absence of more relevant information.

But if we want not merely a pair of bounds —or a favored lower bound— for the conclusion B of an MP but the *exact* value $[\![B]\!]$, the obvious candidate formula for this follows easily: suppose we are given not only $[\![B|A]\!]$ but also $[\![A|B]\!]$ (that we note by σ and ν) and we assume them estimated by experts. We then formulate MP as

$$\frac{\begin{array}{l} A(m) \\ A \to B(\sigma, \nu) \end{array}}{B(p)}$$

which is exactly (6) except that the conditional has prompted evaluation of relative truths of A and B in both directions. The value is computable at once from the above definition of $[\![A|B]\!]$:

$$[\![B]\!] \; = \; \frac{[\![B|A]\!] \cdot [\![A]\!]}{[\![A|B]\!]} \qquad \text{or} \qquad p \; = \; \frac{\sigma \cdot m}{\nu}.$$

Note that the above logical formula coincides formally with Bayes's theorem —whence the adjective ("Bayesian") for any calculus that uses it— except that it deals not with hard-to-compute probabilities of events but with beliefs (or assertion strengths) of sentences. Note also that this value is the one that some *approximate reasoning* systems (e.g. Prospector) unqualifiedly assign to $[\![B]\!]$ supposedly on purely probabilistic grounds —and falsely assuming that $[\![B|A]\!]$ is the same as $[\![A \to B]\!]$—; see, for instance, the Genesereth & Nilsson [1987] text, where the logical equation above is said to be Bayes's formula.

If we wanted the MP presented in the more traditional *logical* way (5), first we would directly estimate the truth value $[\![A \to B]\!]$ of the conditional, or compute it from σ through (7) —or both, and use each estimate as a cross-check on the other—, so we would now have, along with the expert guess of ν:

$$\begin{array}{ll} A & m \\ A \to B & n(\nu) \\ \hline B & p \end{array}$$

(where $n = 1 - m \cdot (1 - \sigma)$), and so

$$(11)\quad [\![B]\!] \;=\; \frac{[\![A \wedge B]\!]}{[\![A|B]\!]} \;=\; \frac{m + n - 1}{\nu}$$

that naturally fits the (9) bounds (when ν runs along from 1 to $[\![A]\!]$).

12 THREE COROLLARIES AND ONE EXTENSION

A few remarks can be made on some apparent advantages of the "logic-as-truth-valuation" approach that, following the advice of Popper *et alii*, we have advocated and described above. First, we mention three well-known fields that have been traditionally perceived as separate but that now automatically become special cases of a single formulation. Then, in subsection *b*, we hint at an obvious extension of the approach.

12.1 The special cases

1. *Classical* (two-valued) *logic* is the special case of our general logic in which *every* sentence is *binary* (i.e. $\forall A \in \mathcal{L} \;\; [\![A]\!] \in \{0, 1\}$) or, equivalently, in which *every* sentence has *zero* imprecision (i.e. $\forall A \in \mathcal{L} \;\; f(A) = 0$).

2. For *three-valued logics* first note that classical examples, especially Kleene's system of strong connectives [1938] and Łukasiewicz's [1920]

$Ł_3$, give the following tables for the values of connectives (where U stands for "undetermined"):

$\wedge$	0	U	1		$\vee$	0	U	1		$\rightarrow$	0	U	1
0	0	0	0		0	0	U	1		0	1	1	1
U	0	X	U		U	U	Y	1		U	U	Z	1
1	0	U	1		1	1	1	1		1	0	U	1

with $X = Y = U$ in both Kleene's and Łukasiewicz's tables, and $Z = U$ in Kleene's (but $Z = 1$ in Łukasiewicz's).

Now, if we abbreviate the "$[\![A]\!] \in (0,1)$" of our general logic by "$[\![A]\!] = U$" (as in Kleene or Łukasiewicz) the values given in the above tables coincide exactly with those that would have been computed by our formulas, *except* that X, Y and Z would remain undetermined until we knew α_{AB}. In general, our values would match Kleene's, but in certain cases they would yield differing results:

(a) If $[\![A]\!]+[\![B]\!] \leq 1$ and A and B are *incompatible* (typically because $\mathbf{A} \cap \mathbf{B} = \emptyset$) then $X = 0$.

(b) If $[\![A]\!]+[\![B]\!] \geq 1$ and A and B are *incompatible* (typically because $\mathbf{A} \cup \mathbf{B} = \Theta$) then $Y = 1$.

(c) If $[\![A]\!] \leq [\![B]\!]$ and A and B are *compatible* (typically because $\mathbf{A} \subset \mathbf{B}$) then $Z = 1$.

Note that in the particular case in which B is $\neg A$ we have *always* $[\![A \wedge \neg A]\!] = 0$ and $[\![A \vee \neg A]\!] = 1$ for any valuation, so that, for instance, the three classical Aristotelian principles ($\vdash A \rightarrow A$, $\vdash \neg(A \wedge \neg A)$ and $\vdash A \vee \neg A$), which do not hold in these logics (except that the first one does in $Ł_3$), do now hold in ours. These three results are perfectly classical and in full agreement with what is to be expected from a (Boolean) logic.

(d) *Łukasiewicz & Tarski*'s [1930] $Ł_\infty$ logic, as ours, generalizes classical (two-valued) logic in the sense that it allows the members of the sentential lattice to take values in [0,1] other than 0 or 1. Both systems of logic include classical two-valued logic as a special case. Nevertheless, while $Ł_\infty$ renounces *Booleanity*, we renounce *functionality* (though not quite, since each connective is actually truth-functional in *three* arguments: $[\![A]\!]$, $[\![B]\!]$ and a third parameter such as, e.g., α_{AB}). In fact, if the $\{0,1\}$ truth set is extended to [0,1] those two properties of classical logic cannot be both maintained, and one must sacrificed; we find much easier to justify logically, and more convenient, the sacrifice of truth-functionality.

Our general logic admits $Ł_\infty$ as a special case since, indeed, $Ł_\infty$ behaves exactly as ours would do if *no* sentence in $\mathcal{L}$ could be recognized as a *negation* of some other and, then, it would be assigned systematically the maximum compatibility ($\alpha = 1$) connective formulas. Naturally that would give an error in the values of composite sentences involving non-fully-compatible subsentences, but it would also restore the lost truth-functionality of two-valued logic. $Ł_\infty$ amounts —from our perspective— to viewing *all* sentences as having *always* maximum mutual compatibility. That means that $Ł_\infty$ conceives *all* sentences as *nested* (i.e. for every A and A', either $\mathbf{A} \subset \mathbf{A}'$ or $\mathbf{A}' \subset \mathbf{A}$) (Sales [1994]). Such a picture is strongly reminiscent of Shafer's [1976] description of conditions present in what he calls *consonant* valuations, and which entails the fiction of a *total*, linear order $\vdash$ in $\mathcal{L}$ —and $\subset$ in $\mathcal{P}(\Theta)$— (that means a coherent, negationless universe) ... which is probably the best assumption we can make when information on sentences is lacking and negations are not involved —or cannot be identified as such. (Such an option is as legitimate as that of assuming, in the absence of information on sentences, that these are independent.) In $Ł_\infty$ obvious negations may be a problem, but it can be solved by applying error-correcting *supervaluations* (van Fraassen [1968]) —that our logic supplies automatically. And note that, in particular, the error incurred in by $Ł_\infty$ when failing to distinguish between a sentence and its negation —thus not being able to recognize a contradiction— is just the quantity we called *imprecision* (or curiously, for the historical record, what Black called *vagueness* and defined formally as we did with imprecision, see Sales [1982b]).

12.2 Ignorance as subadditivity

If we now suppose that $\mathcal{L}$ is still in a Boolean algebra but the v valuation we impose on $\mathcal{L}$ is *subadditive*, i.e.:

$$\{[A \wedge B]\} + \{[A \vee B]\} \geq \{[A]\} + \{[B]\} \quad \textit{(Subadditivity)}$$

(where we note explicitly by the $\{[\ ]\}$ brackets that v is subadditive), then v can be characterized as a *lower probability* (see Good [1962]) or as a *belief* $Bel(A)$ (in Shafer's [1976] sense). If the inequality sign were inverted, the valuation would become an *upper probability* or (Shafer's) *plausibility* $Pl(A)$. The defective value —the part of the value $v(A)$ attributed neither to A nor to $\neg A$— can be expressed as:

$$\begin{aligned} \partial(A) &= 1 - (\{[A]\} + \{[\neg A]\}) \\ &= (1 - \{[\neg A]\}) - \{[A]\} \\ &= Pl(A) - Bel(A) \end{aligned}$$

In the finite $\mathcal{L}$ case we know (Shafer [1976]) that a subadditive valuation like $v : \mathcal{L} \longrightarrow [0,1] : A \mapsto \{[A]\}$ —or, better, the measure $\mu : \mathcal{P}(\Theta) \longrightarrow [0,1] : \mathbf{A} \mapsto \mu(\mathbf{A})$ induced on $\mathcal{P}(\Theta)$ by that valuation— defines a function $m : \mathcal{P}(\Theta) \longrightarrow [0,1]$ (called "*basic assignment*" by Shafer) that satisfies:

1. $m(\varnothing) = 0$
2. $\sum_{\mathbf{A} \subset \Theta} \; m(\mathbf{A}) = 1$

so that the $\mu(\mathbf{A})$ ($= \{[A]\}$) values are computed from this measure through

$$\{[A]\} = \sum_{\mathbf{B} \subset \mathbf{A}} m(\mathbf{B})$$

and, conversely, the $m(\mathbf{A})$ values can be obtained from $\{[A]\}$ ($= \mu(\mathbf{A})$) through

$$m(\mathbf{A}) = \sum_{\mathbf{B} \subset \mathbf{A}} (-1)^{|\mathbf{A}-\mathbf{B}|} \; \mu(\mathbf{B}) \text{ for any } \mathbf{A} \subset \Theta.$$

$Bel(A)$ and $Pl(A)$ happen to coincide with the traditional concept (in Measure Theory) of *inner measure* (P_*) and *outer measure* (P^*), so that the following chain of equivalences has a transparent meaning (notice Shafer calls $Bel(\neg A)$ "degree of doubt of A"):

$$\begin{aligned} Pl(A) &= P^*(\mathbf{A}) = \textstyle\sum_{\mathbf{B} \cap \mathbf{A} \neq \phi} m(\mathbf{B}) \\ &= \textstyle\sum_{\mathbf{B} \subset \Theta} m(\mathbf{B}) - \sum_{\mathbf{B} \subset \mathbf{A}^c} m(\mathbf{B}) \\ &= 1 - P_*(\mathbf{A}^c) = 1 - Bel(\neg A) \end{aligned}$$

We know, also, that an *additive* valuation μ is just a basic assignment m such that $m(\mathbf{A}) = 0$ for all $\mathbf{A} \subset \Theta$ *except* for the singletons $\{\theta\}$ of Θ. This is what Shafer calls '*Bayesian* belief'.

Subadditive belief derives from "non-rational" valuations of evidence by a reasoner (in the Ramsey/de Finetti sense). It can model and explain situations like this classic result: Confronted with the question "Should the Government allow public speeches against democracy?", one user assented 25% of the time. Substituting the word "prohibit" for "allow" elicited a 54% of assenting responses. Since both words are antonyms (the contrary of prohibiting is allowing), it is clear that this user had an unattributed gap left between those two complementary concepts, thus:

$$\{[A]\} + \{[\neg A]\} = .25 + .54 \leq 1$$

which reveals that sentence A (=speeches allowed) was being valued *sub-additively*, and also that "allow" and "fail to prohibit" are here analyzable, in Shafer's terms, as $Bel(A)$ and $Pl(A)$, respectively, with values .25 (for belief) and .46 (for plausibility).

Ignorance (or, rather *total ignorance*) is the particular instance of subadditive valuation in which all non-*true* sentences get the zero value, i.e.

$$\{[A]\} = 1 \quad \text{iff } A = \top\ , \qquad \{[A]\} = 0 \text{ otherwise}$$

(this is what Shafer calls '*vacuous* belief function'). It is just the particular instance of valuation in which all non-*true* sentences get the zero value: indeed, we have, for a given A , $\{[A]\} = \{[\neg A]\} = 0$. In a strict parallel with *total ignorance*, subadditive valuations can also adequately formalize *total certainty* (meaning that $\{[A]\} = 1$ while at the same time $\{[B]\} = 0$ for all $B \vdash A$).

Subadditivity enables us to analyze other interesting situations related to Logic. For instance, suppose we have a subadditive valuation assigning A the value $\mu(\mathbf{A}) = \{[A]\} = Bel(A)$, and that a set $\mathbf{A}'$ (not necessarily a subset of $\mathbf{A}$) can be found in $\mathcal{P}(\Theta)$ such that there is an *additive* valuation which assigns $\mathbf{A}'$ precisely the same value. We denote $\mathbf{A}'$ by "$\Box\mathbf{A}$" (where $\Box$ is a set-operator) and $A' = \delta(\mathbf{A}')$ by "$\Box A$". So we have:

$$Bel(A) \;=\; \{[A]\} \;=\; [\![\Box A]\!]$$

The " $\Box$ " is here a linguistic operator that acts on a sentence A and transforms it into another whose value is the belief (= subadditive truth-value) one can assign non-additively to A, so it seems proper to interpret " $\Box$ " syntactically as "it is believed that", and " $\Box_\alpha$" or " $[\alpha]$ " as "α (= the name of a subject or agent) believes that". Further, we would have

$$\begin{aligned} Pl(A) &= 1 - Bel(\neg A) \;=\; 1 - \{[\neg A]\} \\ &= 1 - [\![\Box\neg A]\!] \;=\; [\![\neg\Box\neg A]\!] \;=\; [\![\Diamond A]\!] \end{aligned}$$

where we have defined a new operator "$\Diamond$" as an abbreviation for "$\neg\Box\neg$" to be interpreted as syntactically as "it is *plausible* that" or "α admits as credible" (so that "$\Diamond_\alpha A$" —or $\langle\alpha\rangle A$— would read "α finds that A can be believed"), because α just does not believe the contrary.

Dubois & Prade [1987] speak rather of "necessity" (or "degree of knowledge") and write $Nec(A) =_{df} \{[A]\}$ $(= [\![\Box A]\!])$, and "possibility" (or "degree of admissibility") $Poss(A) =_{df} 1 - \{[\neg A]\}$ $(= [\![\Diamond A]\!])$; naturally, $Nec(A) \leq [\![A]\!] \leq Poss(A)$ for any A (and also:

$$Nec(A) + Poss(\neg A) \;=\; Poss(A) + Nec(\neg A) \;=\; 1\ ).$$

If necessity (or knowledge) of A and $\neg A$ are totally incompatible (in the sense that $Nec(\neg A) = 0$ whenever $Nec(A) > 0$) then $Nec(A \vee B) = \max(Nec(A), Nec(B))$ and $Poss(A \wedge B) = \min(Poss(A), Poss(B))$.

It is interesting to notice that a subadditive valuation may be superimposed on a sentential lattice without breaking its Boolean character, so that $A \wedge \neg A = \bot$ (bivalence) and $A \vee \neg A = \top$ (excluded middle) still hold,

while at the same time $\{[A]\} + \{[\neg A]\} \leq 1$. This slightly paradoxical fact may explain that many often-encountered situations —where mere subadditivity ($\{[A]\} + \{[\neg A]\} \leq 1$) was probably the case— have been analyzed historically as invalidating the law of the *excluded middle*, because it was felt that there was a "third possibility" between A and $\neg A$ making for the unattributed value $1 - \{[A]\} - \{[\neg A]\}$, covered neither by A nor $\neg A$. If our analysis is correct, such situations are analyzable in terms of incomplete valuations, but this does not imply the breaking of bivalence of any Boolean algebra (except, naturally, for intuitionistic logic, where the algebra is explicitly non-Boolean).

Subadditivity valuations on a Boolean algebra allow also analysis not only of the concept of *ignorance* and *certainty* (as we sketched above) but of the paradoxes of Quantum Logic as well. These arise, according to the 'Quantum Logic' proponents (e.g. Reichenbach in 1944), in explaining why the distributivity fails in this logic, following the standard interpretation of certain experimental results where:

$$p(a) \cdot p(b|a) + p(\bar{a}) \cdot p(b|\bar{a}) < p(b)$$

a relationship we write in this way:

(12) $\{[A]\} \cdot \{[B|A]\} + \{[\neg A]\} \cdot \{[B|\neg A]\} < \{[B]\}$.

What is odd is that this inequality is normally interpreted by quantum logicians (e.g. Watanabe) as meaning:

(13) $(A \wedge B) \vee (\neg A \wedge B) \neq B$

which obviously signals the breaking of distributivity. However, the even-handed transcription of the value-version (12) into the algebraic one (13) is clearly abusive: (13) is much stronger than (12), since it is equivalent to requiring that (12) hold for *all* conceivable valuations. Actually, in our notation, the reading of the experimental results translates immediately into (12), not (13). From there we conclude that:

$$\{[A \wedge B]\} + \{[\neg A \wedge B]\} < \{[B]\}$$

which is in clear violation of additivity, but not of distributivity. So we need not consider that quantum phenomena occur in a non-Boolean algebra (orthomodular lattices are the preferred alternatives) because a *sub-additively*-valued Boolean lattice surely would do for most quantum-logic applications.

Finally we mention that subadditive valuations can as well satisfactorily model how a scientific explanation frame (i.e. the appropriate lattice of theories that cover any observed or predictable true fact) is dynamically replaced by another once the first can no longer account for observed facts: as the valuation of explanations turns subadditive —reflecting that they

no longer cover all predictable facts— one is naturally forced to replace the original sentential structure of theories by a new one (still a Boolean lattice) provided with a new —now again additive— valuation on it that restores the balance; the augmented lattice generators are the new vocabulary, and the elementary components of the new structure are the required new explanatory elements (atomic theories) for the presently observed facts. Such a simple valuation-revision and lattice-replacement mechanism may serve to illustrate the basic dynamics of theory change in scientific explanation (see Sales [1982b]).

APPENDIX

A ON FUZZY LOGIC

A.1 The "Fuzzy Logic" tradition

In a joint reflection with Richard Bellman in 1964 Lotfi Zadeh, then a process control engineer, considered the nonsense of painstakingly computing numerical predictions in control theory contexts where the situation is complex enough to render them meaningless. So he proposed instead to rely confidently on broad –and inherently *vague*– linguistic descriptions like 'high' (for a temperature) or 'open' (for a valve) rather than on misleadingly precise values. He then went to suggest (in Zadeh [1965]) a non-standard extensional interpretation of Predicate Calculus. Though first advanced by Karl Menger (in a 1951 note to the French Académie entitled *Ensembles flous et fonctions aléatoires*), the idea was nevertheless original and simple: an atomic predicate sentence like 'the temperature is high' is assigned truth values in [0,1] reflecting the applicability of the sentence to circumstances (the actual temperatures); those values then define a "set", a "fuzzy" set, by considering them to be the values of a generalized *characteristic* or *set membership* function, in a way that is strictly parallel to the standard procedure for defining predicate extensions (e.g. of 'prime number') by equating the $\{0,1\}$ truth values with the characteristic function of the set (i.e 'the prime numbers') so that if for instance $[\![\text{ Prime } (3)]\!] = 1$ then $\chi_{\mathbf{Primes}}(3) = 1$, i.e. $3 \in \mathbf{Primes}$ (and so now, accordingly, if e.g. $[\![\text{ High } (170)]\!] = 0.8$ then $\chi_{\mathbf{High\_temps}}(170) = 0.8$ or $170 \in_{0.8} \mathbf{High\_temps}$).

Some snags soon arose to question the utter simplicity of the scheme, doubts such as: should set inclusion (defined as $\mathbf{A} \subset \mathbf{B}$ iff $\forall x \ \chi_{\mathbf{A}}(x) \leq \chi_{\mathbf{B}}(x)$) fail if a single point x does not satisfy the relation? or, more fundamentally: what are the appropriate formulas for the connectives? Zadeh had first proposed the usual Łukasiewicz connective formulas (the *min* and the *max* for the $\wedge$ and $\vee$) but then (in 1970, again with R. Bellman) considered that these were "non-interactive" and to be preferred only in the absence of more relevant information, so he offered further formulas he called "soft"

or "interactive" (the product and sum-minus-product). Bellman and Gierz [1973] showed that under certain pre-established conditions (notably, *truth-functionality*) the only possible connectives were the *min* and the *max*. For a (standard) logician this is an unfortunate result, since the *min* formula when applied to a (0,1)-valued sentence A and its negation $\neg A$ yields always a non-zero value ($[\![A \wedge \neg A]\!] = \min([\![A]\!], 1 - [\![A]\!]) \neq 0$), so explicitly *negating* the classical law of non-contradiction (never questioned before by any Logic) and thus placing Fuzzy Logic outside the standard logic mainstream, for which $\neg(A \wedge \neg A)$ is always guaranteed theoremhood (and so, necessarily, $[\![A \wedge \neg A]\!] = 0$ for any valuation —provided 1 is the only designated value, as we assume). Another unfortunate by-product, this one algebraic in character, is that the neatness and simplicity of the Boolean algebra structure —preservable only by sacrificing truth-functionality— are irrecoverably lost.

Note that *(a)* Boolean structure had to be sacrificed in Fuzzy Logic just by technical reasons (the *min* formula), not by a deliberate or methodological bias, and that *(b)* the choice of connectives was historically motivated, in fuzzy-set theory, by pragmatic reasons (prediction accuracy in applications) rather than by logical method: thus, many formulas were tried and discussed (see e.g Rodder [1975] and Zimmerman [1977]). (Interestingly, while theoreticians stuck to Łukasiewicz's *min*, practitioners —e.g. Mamdani [1977]— preferred the product, perhaps recognizing that in complex or poorly-known systems the best policy is to suppose sentences *independent* –in our sense, see above–; whence the product.) Anyway, after a brief flurry of discussion in the 1970s about the right connectives, fuzzy-set theory proceeded from then onwards *non-compatibly* by emphasizing that its basic theme is linguistic *vagueness*, not *logic* or *uncertainty*, and that the [0,1]-values are grounded on a (possibly non-additive) valuation called "possibility", for which a complete subtheory has been elaborated since.

The initially intuitive "fuzzy logic" approach has since become an independent growth industry. From a logical point of view, some foundational points are arguable: (1) the apparently undisputable *truth-functionality* requirement, already present in Łukasiewicz, imposes a radical departure from (ordinary) logic: the sentences *cannot* form any longer a Boolean structure, and traditional logic principles are gone forever; (2) the theory's justification for using (0,1)-values (values that reflect *imprecision*, since $[\![A]\!] \in (0,1)$ implies $f(A) > 0$) is strictly linguistic: the cause of imprecision is attributed solely to the *vagueness* of language and explicitly excludes any non-linguistic component or dimension such as *uncertainty* (considered non-intersectingly to be the domain of Probability theory) or simply *approximation*.

The emphasis fuzzy theorists place on *vagueness* and its "orthogonality" with respect to other causes of unattributed truth value is understandable considering the basic tenets of the theory but questionable from a non-partisan stand.

First: in all cases we finally obtain a number, and this has a transparent function in reasoning: keeping track of our confidence in what we say. In the vagueness case the value we assign is clearly the *degree of applicability* of the sentence to the circumstances at hand (and this is seen by executing the (T) schema); in the uncertainty case it is our "belief" or its "probability" (in some more or less standard sense) what emerges from the (T) evaluation procedure. In either case (*vagueness* or *uncertainty*), what we try to capture and measure is the *degree of approximation* (to truth, or to full reliability) that we can confidently assign the sentence, and what we have in both cases is *imprecision* (difficulties in ascertaining the $\{0,1\}$ truth value and also, consequently, doubts about committing ourselves to it along a whole inference process). Plausibly, it is easier to assign values consistently (in the [0,1] interval) to the sentence, regardless of where or why imprecision arose in the first place, because what we are mainly interested in is the *degree of confidence* we attach to this piece of information we are manipulating through the (hopefully truth-preserving) inferences.

Second: the two dimensions of imprecision, *linguistic* and *epistemic* (for vagueness and uncertainty, resp.), are not so separable as claimed, either conceptually or practically. (We do not consider here occasional claims –often made by Zadeh– that vagueness is *in things* –even in truth–, i.e. it is not linguistic, but ontological). As seen through application of the Tarski (T) schema, when the agent is uncapable of making up his/her own mind as to the truth of the (unquoted) sentence, the *cause* for the under-attribution and the origin of the ε residual value need not be considered, only the resulting *confidence* matters. The non-attribution of "normal" (i.e. $\{0,1\}$) truth values may originate in language (i.e. the sentence is *vague*) or in imperfect verification conditions (i.e. the sentence, even being linguistically precise, is nevertheless *uncertain* due to identification or measurement difficulties or other causes), but spotting the source is mostly an academic exercise: consider, for example, the sentence 'this is probable', which is imprecise in either or both senses, or the historically motivating illustration by Bellman/Zadeh (the 'high temperature' process-control case), where a discrimination of origins, either linguistic (the expression is vague) or epistemic (we don't know what is the precise case, or we have difficulties in identifying it) is indifferent or pointless.

Moreover, not only such boundary examples cast doubts on the claimed vagueness/uncer-tainty orthogonality; even the *linguistic* character of fuzziness (or of "possibility") is arguable. First, because a vague sentence, that to the utterer may mean just that the expression lacks straightforward *applicability* to actual fact, to the hearer –if not involved in the described situation– it may be *epistemic* information about what to expect (for instance, hearing a temperature is "high" may set the unknowing hearer into a –correspondingly imprecise– alert state; (s)he then even may usefully turn

the "membership" or "possibility" number into an *a priori*-probability or belief estimate). Second, because the number we assign to approximate membership in a class, which reflects the *applicability* of the sentence to the observed situation (a semantic quantity shared by the speakers of the language), is *constructed* and continually adjusted by the language speakers on the base of their experience of past cases, in a process which is the same as that of constructing all other $[\![A]\!]$ assignments, however called ("truths", "degrees", "applicabilities", "probabilities", "beliefs", "approximation" or whatever). The process, described above, consists in setting a universe Θ (here the application instances of the sentences in the language) and then considering cases $\theta \in \rho(A)$ (here the θs are application instances – *utterances*– of A) from past experience; the weighed result is $[\![A]\!] = \mu(\rho(A))$ (here the *applicability* of the –vague– sentence A, i.e. its "fuzziness"), and the relationship between A and all other sentences can also be user-evaluated through the compatibilities α or the sufficiency/causality degrees σ introduced in the text (in a way that otherwise amounts to Popper's 'binary probability' evaluation process).

This process of *constructing* values may be considered a further instance of our general approach to rationality based on the universe Θ of *cases*, and the resulting number $[\![A]\!]$ may be used in general reasoning as all other "truth values" are. We thus assure compatibility inside a shared formalism from which Logic, Probability and (e.g.) fuzzy reasoning or belief theories can be derived directly without costly or unnatural translations. To do this we merely need that the different interpretations (including the "fuzzy" one) obey the same laws and have the same formal components: (1) a common sentential language equipped with a *Boolean* structure (easy to justify and convenient for preserving the commonly accepted laws of logic), (2) *coherence* in attributing the values (regardless of the meaning we give them) to assure *additivity*, and (3) a predisposition to sacrifice truth-functionality when required. If the notion of fuzziness, however justified, could be conceived in this way, then the original 1964 Bellman/Zadeh proposal could be subsumed and solved in a very natural way, and we probably could do without a separate, costly and incompatible additional formalism. (In other words, we probably wouldn't need "fuzzy logic".)

A.2 Gaines's 'Standard Uncertainty Logic'

In an interesting effort, born inside the "fuzzy" tradition, to construct a common formalization by discriminating between algebraic *structures* and truth *valuations* on them —somewhat paralleling and anticipating the aim of our present development— Gaines [1978] set out to define what he called 'Standard Uncertainty Logic', that covered and formalized two known subcases. This logic postulates (a) a proposition lattice with an algebraic structure that is initially assumed *Boolean* but —by technical reasons—

finally admitted to be merely distributive, and (b) a finitely *additive* valuation of the lattice on the [0,1] interval. Gaines then distinguishes two special cases of his logic: (1) what he calls 'probability logic' and defines to be the particular case of his logic where the law of the excluded middle holds, and (2) what he terms 'fuzzy logic', defined (in one among several alternative characterizations) to be his logic when that law does not hold. Apparently, Gaines's encompassing logic should correspond to our general logic above (that covers classical logic as a special case, just as Gaines's logic becomes his 'probability logic' when "all propositions are binary"), but closer examination reveals that Gaines's confusingly called 'probability logic' turns out to be actually coextensional with *classical* logic, while his 'fuzzy logic' is, simply, *Łukasiewicz's* $Ł_\infty$. This fact is spelled out by the property he mentions of propositional equivalence (here in our notation):

$$[\![A \leftrightarrow B]\!] \;=\; \min(1 - [\![A]\!] + [\![B]\!], 1 - [\![B]\!] + [\![A]\!])$$

in which the right hand is clearly $1 - |[\![A]\!] - [\![B]\!]|$. This is an expression that is deduced from Gaines's postulates *only* if, necessarily, either $A \vdash B$ or $B \vdash A$ (note this either/or condition —exactly corresponding to our $\alpha_{AB} = 1$ full-compatibility situation— is explicitly mentioned by Gaines as a characteristic property of his 'fuzzy logic'). So Gaines makes the implicit assumption that the base lattice is *linearly* ordered (perhaps induced to it by the $\leq$ symbol used for the –partial– propositional order in the lattice). A confirmation for this comes from the fact that classical logic principles —that hold, by definition, in his 'probability logic'— do not hold in this one. No wonder, then, the base lattice cannot be but merely distributive.

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya. Spain.

BIBLIOGRAPHY

[Adams, 1968] E. W. Adams. Probability and the logic of conditionals, in: J. Hintikka & P. Suppes, eds. *Aspects of inductive logic*, North Holland, 1968.

[Aronson *et al.*, 1980] A. R. Aronson, B. E. Jacobs, and J. Minker. A note on fuzzy deduction, *Journal of the Assoc. for Computing Machinery*, **27**, 599–603, 1980.

[Bacchus, 1990] F. Bacchus. *Representing and reasoning with probabilistic knowledge*, MIT Press, 1990.

[Bar-Hillel and Carnap, 1952] Y. Bar-Hillel and R. Carnap. An outline of a theory of semantic information, TR 247 Research Lab. Electronics, MIT, 1952; reproduced in: Y. Bar-Hillel, *Logic and information*, Addison Wesley (1964)

[Bellman and Gierz, 1973] R. Bellman and M. Gierz. On the analytical formalism of the theory of fuzzy sets, *Info. Sci.*, **5**, 149–156, 1973.

[Bellman and Zadeh, 1970] R. Bellman and L. Zadeh. Decision-making in a fuzzy environment, *Management Science*, **17**, 141–162, 1970.

[Bellman and Zadeh, 1976] R. Bellman and L. Zadeh. Local and fuzzy logics, in: J. M. Dunn & G. Epstein, eds., *Modern Uses of Multiple-Valued Logic*, Reidel, 1976.

[Black, 1937] M. Black. Vagueness, *Phil. Sci.*, **4**, 427–455, 1937.

[Bolc and Borowik, 1992] L. Bolc and P. Borowik. *Many-valued logics*, Springer, 1992.

[Boole, 1854] G. Boole. *An Investigation of the Laws of Thought*, Dover, N.Y. (1958), 1954.

[Carnap, 1950] R. Carnap. *Logical Foundations of Probability*, U. Chicago Press, 1950.

[Carnap, 1962] R. Carnap. The aim of inductive logic, in: E. Nagel, P. Suppes & A. Tarski, eds., *Logic, Methodology and Philosophy of Science*, Stanford, 1962.

[Cohen, 1990] J. Cohen. What I have learned (so far), *Am. Psychologist*, 1307–1308, 1990.

[Cox, 1961] R. T. Cox. *The algebra of probable inferences*, Johns Hopkins U. Press, 1961.

[de Finetti, 1931] B. de Finetti. Sul significato soggettivo della probabilità, *Fundamenta Mathematicae*, **17**, 298–329, 1931.

[de Finetti, 1937] B. de Finetti. La prévision, ses lois logiques, ses sources subjectives, *Annales de l'Institut Henri Poincaré*, **7**, 1-68, English translation in: H. E. Kyburg Jr. & H.E. Smokler, eds., *Studies in subjective probability*, John Wiley (1964), 1931.

[ce Finetti, 1970] B. de Finetti. *Teoria delle probabilità*, Einaudi, English translation: *Theory of probability*, John Wiley (1974), 1070.

[de Luca and Termini, 1972] A. de Luca and S. Termini. A definition of a non-probabilistic entropy in the setting of fuzzy sets theory, *Info. and Control*, **20**, 301, 1972.

[Dubois and Prade, 1987] D. Dubois and H. Prade. *Théorie des possibilités*, Masson (2nd edition), 1987.

[Dubois *et al.*, 1993] D. Dubois, J. Lang and H. Prade. Possibilistic logic, in: S. Abramsky, D. Gabbay & T.S. Maibaum, eds., *Handbook of Logic in Artificial Intelligence* (Vol. 3), Oxford Univ. Press. 1993.

[Duda *et al.*, 1976] R. Duda, P. Hart and N. Nilsson. Subjective Bayesian methods for rule-based information systems, in: B.W. Webber & N. Nilsson, eds., *Readings in Artificial Intelligence*, Morgan Kaufmann, San Mateo (1981), 1976.

[Fagin *et al.*, 1990] R. Fagin, J. Y. Halpern and N. Megiddo. A logic for reasoning about probabilities, *Information & Computation*, *bf 87*, 78–128, 1990.

[Fenstad, 1967] J. E. Fenstad. Representations of probabilities defined on first order languages, in: J.N. Crossley, ed., *Sets, Models and Recursion Theory*, North Holland, 1967.

[Fenstad, 1968] J. E. Fenstad. The structure of logical probabilities, *Synthese*, **18**, 1–23, 1968.

[Fenstad, 1980] J. E. Fenstad. The structure of probabilities defined on first order languages, in: R.C. Jeffrey, ed., *Studies in inductive logic and probability II*, U. of California Press, 1980

[Fenstad, 1981] J. E. Fenstad. Logic and probability, in: E. Agazzi, ed., *Modern Logic. A survey*, Reidel, 1981.

[Field, 1977] H. H. Field. Logic, meaning and conceptual role, *J. of Phil. Logic*, **74**, 379–409, 1977.

[Friedman and Halpern, 1995] N. Friedman and J. Y. Halpern. Plausibility measures; a user's guide (draft available on WWW at http/robotics.stanford.edu), 1995.

[Gaifman, 1964] H. Gaifman. Concerning measures in first order calculi, *Israel J. of Mathematics*, **2**, 1–18, 1964.

[Gaines, 1978] B. R. Gaines. Fuzzy and probability uncertainty logics, *Info. and Control*, **38**, 154–169, 1978.

[Garbolino *et al.*, 1991] P. Garbolino, H. E. Kyburg,*et al.*. Probability and logic, *J. of Applied Non-Classical Logics*, **1**, 105–197, 1991.

[Gärdenfors, 1988] P. Gärdenfors. *Knowledge in flux*, MIT Press, 1988.

[Genesereth and Nilsson, 1987] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, 1987.

[Gerla, 1994] G. Gerla. Inferences in probability logic, *Artificial Intelligence*, **70**, 33–52, 1994.

[Good, 1950] I. J. Good. *Probability and the weighting of evidence*, Griffin, London, 1950.

[Good, 1962] I. J. Good. Subjective probability as the measure of a non-measurable set, in: E. Nagel, P. Suppes & A. Tarski, eds., *Logic, Methodology and Philosophy of Science*, Stanford, 1962.

[Gottwald, 1989] S. Gottwald. *Mehrwertige Logik*, Akademie-Verlag, 1989.

[Grosof, 1986] B. Grosof. Evidential confirmation as transformed probability, in: Kanal, L.N., & Lemmer, J.F., eds. : 1986, *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[Haack, 1974] S. Haack. *Deviant logic*, Cambridge, 1974.

[Hailperin, 1984] T. Hailperin. Probability logic, *Notre Dame J. of Formal Logic*, **25**, 198–212, 1984.

[Hájek, 1996] P. Hájek. Fuzzy sets and arithmetical hierarchy, *Fuzzy Sets and Systems* (to appear)

[Halpern, 1990] J. Y. Halpern. An analysis of first order logics of probability, *Artificial Intelligence*, **46**, 311–350, 1990.

[Hamacher, 1976] H. Hamacher. On logical connectives of fuzzy statements and their affiliated truth function, *Proc. 3rd Eur. Meet. Cyber. & Systems Res.*, Vienna, 1976.

[Harper, 1975] W. L. Harper. Rational belief change, Popper functions, and counterfactuals, *Synthese*, **30**, 221–262, 1975.

[Heckermann, 1986] D. Heckermann. Probabilistic interpretations for MYCIN's certainty factors, in: Kanal, L.N., & Lemmer, J.F., eds. : 1986, *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[Hintikka and Pietarinen, 1968] J. Hintikka and J. Pietarinen. Probabilistic inference and the concept of total evidence, in: J. Hintikka & P. Suppes, eds. *Aspects of inductive logic*, North Holland, 1968.

[Horn and Tarski, 1948] A. Horn and A. Tarski. Measures in Boolean algebras, *Trans. Am. Math. Soc.*, **64**, 467–497, 1948.

[Kanal and Lemmer, 1986] L. N. Kanal and J. F. Lemmer, eds. *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[Keynes, 1921] J. M. Keynes. *A treatise on probability*, Macmillan, NY, 1921.

[Kleene, 1938] S. C. Kleene. On notation for ordinal numbers, *J. of Symbolic Logic* **3**, 150–155, 1938.

[Kolmogorov, 1933] A. N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*, Springer, English translation: *Foundations of the theory of probability*, Chelsea, NY (1956), 1933.

[Koopman, 1940] B. O. Koopman. The bases of probability, *Bulletin Am. Math. Soc.*, **46**, 763–774, 1940.

[Koppelberg *et al.*, 1989] S. Koppelberg, J. D. Monk and R. Bonnet, eds. *Handbook of Boolean algebras* (3 vols.), North Holland, 1989.

[Kyburg, 1993] H. E. Kyburg Jr. Uncertainty logics, in: S. Abramsky, D. Gabbay & T.S. Maibaum, eds., *Handbook of Logic in Artificial Intelligence* (Vol. 3), Oxford Univ. Press, 1993.

[Jeffrey, 1965] R. Jeffrey. *The logic of decision*, North Holland, 1965.

[Jeffrey, 1995] R. Jeffrey. From logical probability to probability logic, *10th Int. Congress of Logic, Methodology & Phil. of Sc.*, Aug. 19-25, 1995, Florence, 1995.

[Jeffreys, 1939] H. Jeffreys. *Theory of probability*, Oxford, 1939.

[Johnson-Laird, 1975] P. N. Johnson-Laird. Models of deduction, in: R.J. Falmagne, ed., *Reasoning: Representation and process in children and adults*, L. Erlbaum, 1975.

[Johnson-Laird, 1983] P. N. Johnson-Laird. *Mental models*, Cambridge Univ. Press, 1983.

[de Laplace, 1774] P. S. de Laplace. Mémoire sur la probabilité des causes par les événements, *Mémoires de l'Académie des Sciences de Paris*, Tome **VI**, 621, 1774.

[de Laplace, 1820] P. S. de Laplace. *Théorie analytique des probabilités* (3rd ed.), Courcier, 1820.

[Leblanc, 1960] H. Leblanc. On a recent allotment of probabilities to open and closed sentences, *Notre Dame J. of Formal Logic*, **1**, 171–175, 1960.

[Leblanc, 2001] H. Leblanc. Alternatives to standard first-order semantics, in: D. Gabbay & F. Guenthner, eds., *Handbook of Philosophical Logic*, Second Edition, Volume 2, pp. 53–132 Kluwer, Dordrecht, 2001.

[Lee, 1972] R. C. T. Lee. Fuzzy logic and the resolution principle, *Journal of the Assoc. for Computing Machinery*, **19**, 109–110, 1972.

[Lewis, 1976] D. Lewis. Probabilities of conditionals and conditional probabilities, *Philosophical Review*, **85**, 297–315, 1976.

[Łoś, 1962] J. Łoś. Remarks on foundations of probability, *Proc. Int. Congress of Mathematicians*, Stockholm, 1962.

[Łukasiewicz, 1920] J. Łukasiewicz. O logice trójwartościowej, *Ruch Filozoficzny* (Lwów), **5**, 169-171, 1920; see L. Borkowski, ed., *J. Łukasiewicz: Selected works*, North Holland (1970).

[Łukasiewicz, 1938] J. Łukasiewicz. Die Logik und das Grundlagenproblem, in: F. Gonseth, ed., *Les entretiens de Zurich, 1938*, Leemann (1941) (pages 82–100, discussion pages 100–108).

[Łukasiewicz and Tarski, 1930] J. Łukasiewicz and A. Tarski. Untersuchungen über den Aussagenkalkül, *Comptes Rendus Soc. Sci. & Lettres Varsovie*, Classe **III**, 51-77, 1930; translated by J.H. Woodger in: Alfred Tarski, *Logic, Semantics, Metamathematics*, Oxford (1956).

[Mamdami, 1977] E. H. Mamdani. Application of fuzzy logic to approximate reasoning using linguistic synthesis, *IEEE Trans. on Computers*, **26**, 1182–1191, 1977.

[MacColl, 1906] H. MacColl. *Symbolic Logic and its Applications*, London, 1906.

[Miller, 1978] D. W. Miller. On distance from the truth as a true distance, in: J. Hintikka, I. Niinilnuoto & E. Saarinen, eds., *Essays in Mathematical and Philosophical Logic*, Reidel, 1978.

[Miller, 1981] D. W. Miller. A Geometry of Logic, in: H. Skala, S. Termini & E.Trillas, eds., *Aspects of Vagueness*, Reidel, 1981.

[Nilsson, 1986] N. Nilsson. Probabilistic logic, *Artificial Intelligence*, **28**, 71–87, 1986.

[Nilsson, 1993] N. Nilsson. Probabilistic logic revisited, *Artificial Intelligence*, **59**, 39–42, 1993.

[Paass, 1988] G. Paass. Probabilistic logic, in: P. Smets, E.H. Mamdani, D. Dubois & H. Prade, eds., *Non-standard logics for automated reasoning*, Academic Press, 1988.

[Pavelka, 1979] J. Pavelka. On fuzzy logic I, II, III, *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, **25**, 45–52, 119–134, 447–464, 1979.

[Pearl, 1988] J. Pearl. *Probabilistic reasoning in intelligent systems*, Morgan Kaufmann, San Mateo, 1988.

[Peirce, 1902] C. S. Peirce. *Minute Logic*, unpublished, 1902; (see ref. also in Rescher [1969]), in: C.S. Peirce, *Collected Papers*, Harvard (1933)

[Popper, 1959] K. R. Popper. *The Logic of Scientific Discovery*, Hutchinson, 1959; (includes Appendix II, originally appeared in *Mind*, 1938, and Appendix IV, originally appeared in *Br. J. Phil. Sc.*, 1955)

[Popper, 1962] K. R. Popper. Some comments on truth and the growth of knowledge, in: E. Nagel, P. Suppes & A. Tarski, eds., *Logic, Methodology and Philosophy of Science*, Stanford, 1962.

[Popper, 1972] K. R. Popper. Two faces of commonsense, in: K. R. Popper, *Objective Knowledge*, Oxford, 1972.

[Popper and Miller, 1987] K. R. Popper and D. W. Miller. Why probabilistic support is not inductive, *Phil. Trans. R. Soc. Lond.*, A **321**, 569–591, 1987.

[Ramsey, 1926] F. P. Ramsey. Truth and probability, 1926; in: Ramsey, F. P.: 1931, *The Foundations of Mathematics*, Harcourt Brace, NY.

[Reichenbach, 1935a] H. Reichenbach. Wahrscheinlichkeitslogik, *Erkenntnis*, **5**, 37–43, 1935.

[Reichenbach, 1935] H. Reichenbach. *Wahrscheinlichkeitslehre*, 1935; English translation: *The Theory of Probability*, U. of California Press (1949)

[Rescher, 1969] N. Rescher. *Many-Valued Logic*, McGraw-Hill, 1969.

[Rodder, 1975] W. Rodder. On 'And' and 'Or' connectives in Fuzzy Set Theory, T. Report, *I. für Wirtschaftswiß.*, Aachen, 1975.

[Sales, 1982a] T. Sales. Una lògica multivalent booleana, *Actes 1er Congrés Català de Lògica Matemàtica* (Barcelona, January 1982), 113–116, 1982.

[Sales, 1982b] T. Sales. *Contribució a l'anàlisi lògica de la imprecisió*, Universitat Politècnica de Catalunya (Ph. D. dissertation), 1982.

[Sales, 1992] T. Sales. *Propositional logic as Boolean many-valued logic*, TR LSI-92-20-R, Universitat Politècnica de Catalunya, 1992.
[Sales, 1994] T. Sales. Between logic and probability, *Mathware & Soft Computing*, **1**, 99–138, 1994.
[Sales,] T. Sales. Logic of assertions, *Theoria*, **25**, 1996.
[Savage, 1954] L. J. Savage. *Foundations of Statistics*, Dover, NY (1972), 1954.
[Scott, 1973] D. Scott. Does many-valued logic have any use?, in: S. Körner, ed., *Philosophy of Logic*, Blackwell (1976), 1973.
[Scott and Kraus, 1968] D. Scott and P. Krauss. Assigning probabilities to logical formulas, in: J. Hintikka & P. Suppes, eds., *Aspects of Inductive Logic*, North Holland, 1968.
[Shafer, 1976] G. Shafer. *A Mathematical Theory of Evidence*, Princeton, 1976.
[Shafer, 1996] G. Shafer. A unified semantics for logic and probability, *Artificial Intelligence and Mathematics*, Fort Lauderdale, Fla., Jan. 3-5 1996.
[Shortliffe, 1976] E. H. Shortliffe. *MYCIN*, American Elsevier, 1976.
[Smith, 1961] C. A. B. Smith. Consistency in statististical inference and decision, *J. of the Royal Stat. Soc.*, Ser. B **23**, 218–258, 1961.
[Stalnaker, 1970] R. Stalnaker., R.: 1970, Probability and conditionals, *Philosophy of Science*, **37**, 64–80, 1970.
[Suppes, 1968] P. Suppes. Probabilistic inference and the concept of total evidence, in: J. Hintikka & P. Suppes, eds., *Aspects of Inductive Logic*, North Holland, 1968.
[Suppes, 1979] P. Suppes. *Logique du probable*, Flammarion, Paris, 1979.
[Tarski, 1935a] A. Tarski. Wahrscheinlichkeitslehre und mehrwertige Logik, *Erkenntnis*, **5**, 174–175, 1935.
[Tarski, 1935b] A. Tarski. The concept of truth in formalized languages, in: J.H. Woodger in: Alfred Tarski, *Logic, Semantics, Metamathematics*, Oxford (1956), 1935.
[Trillas *et al.*, 1982] E. Trillas, C. Alsina, and Ll. Valverde. Do we need max, min and 1-j in Fuzzy Set Theory?, in: R. Yager, ed., *Recent Developments of Fuzzy Set and Possibility Theory*, Pergamon, 1982.
[Urquhart, 1986] A. Urquhart. Many-valued logic, in: D. Gabbay & F. Guenthner, eds., *Handbook of Philosophical Logic* (Vol. 3), Reidel, 1986.
[van Fraassen, 1968] B. C. van Fraassen. Presuppositions, implication and self-reference, *J. Phil.*, **65**, 1968.
[van Fraassen, 1981] B. C. van Fraassen. Probabilistic semantics objectified I, II, *J. of Phil. Log.*, **10**, 371–394, 495–510, 1981.
[Watanabe, 1969] S. Watanabe. Modified concepts of logic, probability and information based on generalized continuous characteristic function, *Info. and Control*, **15**, 1–21, 1969.
[Wittgenstein, 1922] L. Wittgenstein. *Tractatus Logico-Philosophicus*, London, 1922.
[Woodruff, 1995] P. W. Woodruff. Conditionals and generalized conditional probability, *10th Int. Congress of Logic, Methodology & Phil. of Sc.*, Aug. 19-25, 1995, Florence.
[Yager, 1979] R. Yager. A note on fuzziness in a standard uncertainty logic, *IEEE Trans. Systems, Man & Cyb.*, **9**, 387–388, 1979.
[Zadeh, 1965] L. A. Zadeh. Fuzzy sets, *Information and Control*, **8**, 338–353, 1965.
[Zimmermann, 1977] H. J. Zimmermann. Results of empirical work on fuzzy sets, *Int. Conf. on Applied Gen. Systems Research*, Binghamton, NY, 1977.

INDEX