MAX-PLANCK-INSTITUT FÜR INFORMATIK

Classical vs Non-classical Logics
The Universality of Classical Logic

Dov M Gabbay

MPI–I–93–230 August 1993

Im Stadtwald
W 66123 Saarbrücken
Germany

Author's Address

Dov M Gabbay
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ

Publication Notes

This paper will be published in volume 2 of the Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press.

Acknowledgements

The author is a SERC Senior Research Fellow. I am grateful to Professors Bob Kowalski and H. J. Ohlbach for critical reading of the manuscript and many helpful discussions.

Several of the sections in this paper are based on early published shorter versions, while other sections will be expanded into full papers. The following is the correspondence: Section 2 – [Gabbay, 1993], Sections 3 and 4 – [Gabbay, 1992], Section 5–8 – [Gabbay, 1992d], Section 9 – [Gabbay, 1993b]. Section 10 borrows some text from [Gabbay and Ohlbach, 1992].

Abstract

This report investigates the question of the universality of classical logic. The approach is to show that an almost arbitrary logical system can be translated reasonably intuitively and almost automatically into classical logic. The path leading to this result goes through the analysis of what is a reasonable logic, how to find semantics for it, how to build a labelled deductive system (LDS) for it, how to translate a LDS into classical logic and how to automate the process using SCAN.

This report relies on other papers, published and/or to be published as explained in the acknowledgements.

Keywords

classical logic, nonclassical logic, translation of logics

Contents

1 Introduction—the debate

This paper addresses the problem of whether classical logic can be used as a universal system of logic and whether it can be applied, profitably and sensibly, in all areas where logic needs to be applied. More specifically, we focus on modal and temporal logics and ask whether the use of such special logics, as compared with the use of classical logic, is advantageous. We examine what we loose and what we gain if we use classical logic instead of specialised logics to describe and reason about the same (e.g. temporal) phenomena.

The problem we are facing is not simple.[1] Claims for and against classical and non-classical logics are abundant in the literature. To be able to study the question seriously we must set the background properly and isolate for examination the basic features involved. Further than that, we are going to have to analyse several case studies in detail because we have to see what machinery is involved in the classical vs non-classical representation and reasoning. The issue is so dependent on fine tuning of concepts and methods that no amount of general discussion (without actual technical details) will settle it. In short, to support the claim that classical logic is universal in some sense one has to go ahead and show it in technical details!

What is it that we have to do? A close examination of application areas suggests the study of the following basic features which seem to be central and important for our assessment.

1. We must first have a clear concept of the notion of what is a logical system and what is the place of classical and non-classical logics in the family of logics. The proposed notion of logical systems must take into account the multitude of 'logics' arising in application areas such as software engineering and artificial intelligence. In many such applications there seem to be several approaches of using logics, among them feature some prominent non-classical specialised logics, as well as the direct use of classical logics. An analysis of how classical logic compares in such applications with the use of non-classical logic might give us a clue for our general assessment of classical vs non classical logics.

2. We need to analyse the technical characteristics of classical logic and some non-classical logics and compare them in terms of expressive power and computability.

   This will give us a picture of the 'mathematical' properites of these logics. Once we understand the mathematical properties of what we are dealing with, we can proceed and make judgment about applicability.

3. For this purpose we need to study what conceptual committment we make when operating (a) in classical logic and (b) in non-classical logics. If classical logic is a candidate for being a universal system, this analysis will help us determine in what sense it is universal. For example we may discover that from the point of view of automated deduction (and from this point of view only) classical logic can serve as a universal system.

4. Having understood by now the properties of classical and non-classical logics a bit better, the natural next step is to study translation methods and especially the way non-classical logics can be translated into classical logic.

   Experience with applications shows that translation methods are central to classical logic applicability.

5. The notion of translation needs clarification in itself. Many of the reductions of non-classical logic and/or application areas representations make use of classical logic as a metalevel language. This is a special kind of reduction. We therefore need to study notions of metalevel and the possibilities of using any logic as a metalevel tool.

   We need the notion of 'object level implementation of metalevel features'. Obviously the use of classical logic as a metalevel Turing machine is not going to be an acceptable support for its universality.

[1] The problem is referred to as 'The Debate' by our colleagues at Imperial College.

6. There is another problem we need to clarify. Up to now we used the term of 'classical logic' in our discussion. Do we mean first-order classical logic, or do we mean higher-order classical logic or extensions? Can it be many sorted logic? Or perhaps any system which is classical logic 'look alike' and gives the same 'feel'?

7. We need to develop methodology and theorems showing how (almost) an arbitrary logic can be translated in a natural way into classical logic.

8. The study of all the above mentioned concepts needs to be enhanced and sharpened by diverse case studies from typical application areas.

The following is our plan and strategy for the chapter.

Our overall goal is to show that classical logic can be used as a universal system at least from the point of view of automated deduction. Our main support for this view is the technical result that there exists an algorithmic procedure which can translate (in a meaningful way) an almost arbitrary non-classical logic into classical logic. The core of the Chapter is comprised of the analysis of the concepts involved and the presentation and evaluation of the translation.

Section 2 will examine the notion of what is a logical system. The considerations are driven by the needs of application areas and the needs for automated reasoning. We will progressively refine our notion of logic by catering for more and more reasoning features needed in artificial intelligence and software engineering applications and more and more fine tuning features needed for automated reasoning. We will end up with a notion of a logic which is not at all like classical logic. This process will achieve two objectives.

1. It will progressively persuade us that different kinds of logics with many features are needed in applications.

2. It will show us exactly where classical logic is situated in this panorama of logical systems.

To proceed, we now have two options. The first is to accept the need to work directly in non-classical logics and build an optimal logic for our use and/or develop methodologies for working in a pluralistic view of logics. The second is to claim that although we do concede that there is the phenomena of 'non-classical' types of reasoning and other specialised modes of reasoning commonly 'perceived' as new logics we still maintain that really these should be considered as speicalised systems to be *expressed in* classical logic. Thus there is need for only one universal logic, namely classical logic or a variant in which various forms of reasoning can be expressed. There is no need (indeed it may be ill advised) to construct a new logic for each type of reasoning.

To investigate the merits of each option we need case studies and some theoretical work. The second option, using classical logic or variant as a universal system, presupposes expressive power and naturalness of use. This has to be examined.[2]

At this point we must impress upon the reader that the *only way* to obtain a satisfactory answer to the question of the universality of classical logic is to proceed and show *in detail* how and by what mechanisms and translation methods classical logic can *serve* as a universal language and examine the merit of every step we are doing. No general considerations will do. We must carry out a detailed reduction program which is so naturally and clearly motivated that if it is found unsatisfactory then we will be convinced that no such reduction is of merit!

We thus proceed to Section 3 entitled 'How to construct a logic for your application'. We examine in this Section the needs of two central application areas, temporal reasoning and PROLOG and parsing as deduction, and see how classical logic and the more specialised temporal logic and the Lambek Calculus compare with each other mathematically and conceputally. Unfortunately the case studies have to be done in detail and in full. The details are necessary to appreciate the fine tuning and subtlty of the question we are studying.

Our conclusion from these case studies is that although classical logic (or more precisely a variation, many sorted possibly second-order classical logic) is mathematically adequate, there is still

[2]The social argument, that classical logic is well known and is already established in many communities, can only supplement a scientific argument.

the problem of naturalness of the representation and the compatibility of the reasoning movements in the application areas which stands in contrast with the fixed and limited reasoning machinery options of classical logic. On the other hand the non-classical logic candidates, e.g. temporal and modal logics also have their drawbacks especially when it comes to ease of computation and possibilities of skolemisation and normal form.[3]

At this stage (after Section 3) we have a stalemate. We could advise the reader to use specialised logic if he wants to keep close to the application area (fine tuned jobs) and be prepared to sacrifice computational ease and only use classical machinery for the 'heavy' jobs. We can leave it at that and maybe even support a compromise on hybrid reasoning or maybe stick to classical logic and disregard the problem of naturalness.

We choose to persist and proceed along a different route. We say let us be open minded. We have seen the drawbacks of each approach for these case studies. Let us construct a better, more flexible logical framework for the application areas by closely examining their needs. Let us forget about the classical vs non-classical logic problem. Let us just do what seems to be right and develop the best system which hopefully has the best of the classical and non-classical approaches.

This we do in Section 4, right after the case studies. We simply continue to analyse the case studies and devlop the notion of algebraic Labelled Deductive System, *LDS*.

We now resume the main thread of the argument about the universality of classical logic and argue as follows.

We argue that we have examined some case studies and developed some best systems directly for them. These systems can be translated into classical logic (i.e. in our case *LDS* can be translated). The translation is more or less naturalness preserving. Therefore classical logic can serve as a universal system.

To support this argument we need to:

1. Show how an almost arbitrary logic can be naturally presented as an *LDS*.

2. Show how *LDS* can be translated into classical logic in a natural way.[4]

(1) and (2) show that an arbitrary logic can be technically translated into classical logic. The problem now is the acceptability and quality of the reduction-translation. Obviously a metalevel translation through Gödel numbers is not what we want. Such translation will not convince us of the universality of classical logic. A term translation (i.e. turning all wffs into terms) is not satisfactory either.[5]

So we ask what is then to be considered good translation? We therefore need:

3. A theory of translation and metalevel translation or at least some acceptable ways of translation which will count as satisfactory reductions into classical logic.

Sections 5 and 6 deal with translation problems. Section 7 and 8 deal with the metalevel, in an attempt to propose classical logic (look-alike) extension which can serve as a universal system. Section 9 uses algebraic methods and shows that almost any logic can be translated into classical logic.

By now we can assume that we have successfully shown that a rather general class of logics can be translated (in a satisfactory way) into classical logic. Can we now claim that classical logic can serve as a universal logic. Unfortunately, not yet: We still have one more point to argue. Classical logic itself can be translated into, say, intuitionistic logic. This logic is constructive, is

[3]My personal view is that only from the automated reasoning point of view does *current* knowledge distinguish classical logic as universal.

[4]Personally, I was quite happy with *LDS*. It is a general unifying framework for logics which is also capable of bringing the semantics into the syntax and is very flexible. The simple version of algebraic *LDS* seems a good compromise language for our case studies. I could not help but notice, however, that algebraic *LDS* can easily be translated into many sorted classical logic, though some naturalness of the presentation is lost! Since *LDS* is a general framework it can serve as a vehicle for reducing an almost arbitrary logic into classical logic. For the sake of the universality argument the reader need not necessarily support *LDS*, just accept the reduction to classical logic it generates.

[5]I find the claim that the possiblity of using term translation into classical logic actually establishes its universality unreasonable. See example 10.2.5 for my reasons. The notion of *term translation* is defined in 8.0.5 and 9.0.8.

richer and has a nicer proof theory. Our entire argument can be used now to show that we should use intuitionistic or linear logic as the universal logic. Why classical logic? A historical accident? Is it like FORTRAN, a substandard language which cannot be rid of for social reasons or does classical logic have some special features?

Section 10 shows that certain features are available in classical logic which cannot be obtained in intuitionistic logic. These features are available in linear logic. Should we then adopt linear logic?

I shall not answer this question now. Let us wait until the end of the Chapter, where all will be unveiled!

The following summarises the structure of arguments in the debate.

The Proposal

It is strongly reasonable to regard classical logic as a distinguished universal system of logic among the pluralistic family of logics. Classical logic or some reasonable extension of it, can sensibly and profitably 'stand in' for most proposed non-classical logics, especially when applied in Artificial Intelligence, Logic Programming and Software Engineering. Classical Logic is outstanding when it comes to automated reasoning needs.

The approach for supporting the proposal

- Take typical case studies and show how classical logic (or extensions) can 'stand in' for non-classical logics.
- Clarify, as much as possible, all conceptual ambiguities inherent in the meaning of 'standing in for'.

The argument

The argument relies on two main points

- Almost any logic can be translated into classical logic using good translation methods.
- Other logics seem to have some natural requirements in functionality (automated deduction, interpolation, SCAN) and structure (labelling, bringing semantics into syntax) which are better treated through (which in fact invite) translation into classical logic.
- Some of these functionality requirements can be effectively done at current state of knowledge only through translation.

The presentation of the argument

- Present classical logic within the pluralistic family of logics. Stretch the notion of logic as much as possible, to show that really the notion of logic we need (in view of applications) is very general indeed (Section 2).
- Take two traditional case studies. One syntactical (Lambek calculus) with wide applications in logic and language and one semantical (temporal logic) with wide applications in AI and software engineering. Analyse in detail how classical logic can 'stand in' for them. See the weaknesses and propose a natural improvement, (Labeled Deductive Systems). (Section 3).
- Having proposed the discipline of Labelled Deductive Systems, show that it seems to unify and act as a framework for the variety of logics of section 2. Thus show that perhaps *LDS* can be the universal framework. Show how almost an arbitrary logic can be presented in *LDS* (Section 9).
- Show how *LDS* can be more or less naturally translated into classical logic and then using this experience, show how classical logic or extensions (linked predicate languages, which is really *LDS* in disguise and **HFP**) can serve as a universal target translation, (Section 8).
- Strengthen the argument by fortifying the conceptual foundation of object level and metalevel translation.
- Show why translation into other logics cannot give the same functionality, (Section 11).

2 What is a logical system?—the challenge

This section studies the notion of what is a logical system. It will incrementally motivate a notion of logical system through the needs of various applications and applied logical activity. The aim is to set the scene for the examination of the relationship of classical and non-classical logic, and pose the challenge to the claim of the universality of classical logic. The section proposes an increasingly more detailed image of a logical system. The initial position is that of a logical system as a consequence relation on sets of formulas. Thus any set theoretical binary relation of the form $\Delta \mid\sim \Gamma$ satisfying certain conditions (reflexivity monotonicity and cut) is a logical system. Such a relation has to be mathematically presented. This can be done either semantically, or set theoretically or it can be algorithmically generated. There are several options for the latter. Generate first the $\{A \mid \varnothing \mid\sim A\}$ as a Hilbert system and then generate $\{(\Delta, \Gamma) \mid \Delta \mid\sim \Gamma\}$ or generate the pairs (Δ, Γ) directly (via Gentzen rules) or we use any other means (other proof theories)?

The concepts of a *logical system, semantics* and *proof theory* are not sharp enough even in the traditional literature. There are no clear definitions of what is a proof theoretic formulation of a logic (as opposed to, e.g. a decision procedure algorithm) and what is e.g. a Gentzen formulation. Let us try here to propose a working definition, only for the purpose of making the reader a bit more comfortable and not necessarily for the purpose of giving a definitive formulation of these concepts.

- We start with the notion of a well formed formula of the langauge **L** of the logic.
- A consequence relation is a binary (consequence) relation on finite sets of formulas, Δ, Γ written as $\Delta \mid\sim \Gamma$, satisfying certain conditions, namely reflexivity,monotonicity and cut.
- Such a relation can be defined in many ways. For example, one can list all pairs (Δ, Γ) such that $\Delta \mid\sim \Gamma$ should hold. Another way is to give Δ, Γ to some computer program and wait for an answer (which should always come).
- A semantics is an interpretation of the langauge **L** into some family of set theoretical structures, together with an interpretation of the consequence relation $\mid\sim$ in terms of the interpretation. What I have just said is not clear in itself because I have not explained what 'structures' are and what an interpretation is. Indeed, there is no clear definition of what is semantics. In my book [Gabbay, 1976], following Scott I defined a *model* as a function **s** giving each wff of the language a value in {0,1}. A semantics $\mathcal{S}$ is a set of models, and $\Delta \mid\sim_{\mathcal{S}} \Gamma$ is defined as the condition:

$$(\forall \mathbf{s} \in \mathcal{S})[\forall X \in \Delta(\mathbf{s}(X) = 1) \rightarrow \exists Y \in \Gamma(\mathbf{s}(Y) = 1)]$$

- There can be algorithmic systems for generating $\mid\sim$. Such systems are not to be considered 'proof theoretical systems' for $\mid\sim$. They could be decision procedures or just optimal theorem proving machines.
- the notion of a proof system is not well defined in the literature. There are some recognised methodologies such as 'Gentzen formulations', 'tableaux', 'Hilbert style' but these are not sharply defined. For our purpose, let us agree that a *proof system* is any algorithmic system for generating $\mid\sim$ using rules of the form:

$$\frac{\Delta_1 \mid\sim \Gamma_1; \ldots; \Delta_n \mid\sim \Gamma_n}{\Delta \mid\sim \Gamma}$$

and 'axioms' of the form:

$$\frac{\varnothing}{\Delta \mid\sim \Gamma}$$

The axioms are initial list of $(\Delta, \Gamma) \in \mid\sim$ and the other rules generate more. So a proof system is a particular way of generating $\mid\sim$. Note that there need not be structural requirement on

the rule (that each involves a main connective and some subformulas, etc.).
A Gentzen formulation would be a proof system where the rules are very nicely structured (try to define something reasonable yourself, again, there is no clear definition!) and a Hilbert formulation is a proof system where all the Δ's involved are $\varnothing$.

The central role which proof theoretical methodologies plays in generating logics compells us to put forward the view that a logical system is a pair $(\mid\!\sim, \mathbf{S}_{\mid\!\sim})$, where $\mathbf{S}_{\mid\!\sim}$ is a proof theory for $\mid\!\sim$. In other words, we are saying that it is not enough to know $\mid\!\sim$ to 'understand' the logic, but we must also know how it is presented (i.e. $\mathbf{S}_{\mid\!\sim}$).

The next shift in our concept of a logic is when we observed from application areas whose knowledge representation involves data and assumptions the need to add structure to the assumptions and the fact that the reasoning involved relies on and uses the structure. This view also includes non-monotonic systems. This led us to develop the noton of *Labelled Deductive Systems* and adopt the view that this is the framework for presenting logics. Whether we accept these new systems as logics or not, [6] classical logic must be able to represent them.

The real departure from traditional logics (as opposed to just giving them more structure) comes with the notion of aggregating arguments. Real human reasoning does aggregate arguments (circumstantial evidence in favour of A as opposed to evidence for $\neg A$) and what is known as quantitative (fuzzy) reasoning systems make heavy use of that. Fortunately *LDS* can handle that easily. The section concludes with the view that a proper practical reasoning system has 'mechanisms' for updates, inputs, abduction, actions, etc. as well as databases (theories, assumptions) and that a proper logic is an integrated *LDS* system together with a specific choice of such mechanisms.[7]

2.1 Logical systems as consequence relations

Traditionally, to present a logic $\mathbf{L}$, we need to first present the set of well formed formulas of that logic. This is the *language* of the logic. We define the sets of atomic formulas, connectives, quantifiers and the set of arbitrary formulas. Secondly we mathematically define the notion of consequence, namely for a given set of formulas Δ and a given formula Q, we define the consequence relation $\Delta \mid\!\sim_{\mathbf{L}} Q$, reading '$Q$ follows from Δ in the logic $\mathbf{L}$'.

The consequence relation is required to satisfy the following intuitive properties: (Δ, Δ' abbreviates $\Delta \cup \Delta'$).

Reflexivity

$$\Delta \mid\!\sim Q \text{ if } Q \in \Delta$$

Monotonicity

$$\Delta \mid\!\sim Q \text{ implies } \Delta, \Delta' \mid\!\sim Q$$

[6] Indeed almost any logic can be presented as an *LDS*. This had led me to accept that classical logic can act as a universal system from the point of view of automated deduction, because there are natural ways of translating *LDS* into an extension of classical logic as shown in Section 7, and we can benefit from this reduction by using the computational machinery (e.g. resolution) of classical logic.

[7] My personal view is that this is a logic, i.e. Logic = *LDS* system + several *mechanisms*. In AI circles this might be called *an agent*. Unfortunately, some members of the traditional logic community are still very conservative in the sense that they have not even accepted non-monotonic reasoning systems as logics yet. They believe that all this excitement is transient, temporarily generated by computer science and that it will fizzle out sooner or later. They believe that we will soon be back to the old research problems, such as how many non-isomorphic models does a theory have in some inaccessible cardinal or what is the ordinal of yet another subsystem of analysis. I think this is fine for mathematical logic but not for the logic of human reasoning. There is no conflict here between the new and the old, just further evolution of the subject.

Transitivity (Cut)[8]**:**

$$\Delta \mid\!\sim A;\ \Delta, A \mid\!\sim Q \text{ imply } \Delta \mid\!\sim Q$$

The consequence relation may be defined in various ways. Either through an algorithmic system $\mathbf{S}_{\mid\sim}$,or implicitly by postulates on the properties of $\mid\!\sim$.

Thus a logic is obtained by specifying $\mathbf{L}$ and $\mid\!\sim$. Two algorithmic systems $\mathbf{S}_1$ and $\mathbf{S}_2$ which give rise to the same $\mid\!\sim$ are considered the same logic.

If you think of Δ as a database and Q as a query, then reflexivity means that the answer is yes to any Q which is officially listed in the database. Monotonicity reflects the accumulation of data, and transitivity is nothing but lemma generation, namely if $\Delta \mid\!\sim A$, then A can be used as a lemma to obtain B from Δ.

The above properties seemed minimal and most natural for a logical system to have, given that the main applications of logic were in mathematics and philosophy.

The above notion was essentially put forward by [Tarski, 1936] and is referenced to as Tarski consequence. [Scott, 1974], following [Gabbay, 1969], generalised the notion to allow Q to be a set of formulas Γ. The basic relation is then of the form $\Delta \mid\!\sim \Gamma$, satisfying:

Reflexivity

$$\Delta \mid\!\sim \Gamma \text{ if } \Delta \cap \Gamma \neq \varnothing$$

Monotonicity

$$\frac{\Delta \mid\!\sim \Gamma}{\Delta, \Delta' \mid\!\sim \Gamma}$$

Transitivity (cut)

$$\frac{\Delta, A \mid\!\sim \Gamma; \Delta' \mid\!\sim A, \Gamma'}{\Delta', \Delta \mid\!\sim \Gamma, \Gamma'}$$

Scott has shown that for any Tarski consequence relation there exists two Scott consequence relations (a maximal one and a minimal one) that agree with it (see my book [Gabbay, 1986]).

The above notions are monotonic. However, the increasing use of logic in theoretical computer science and artificial intelligence has given rise to logical systems which are not monotonic. The axiom of monotonicity is not satisfied in these systems. There are many such systems, satisfying a variety of conditions, presented in a variety of ways. Furthermore, some are proof theoretical and

[8]There are several versions of the **Cut Rule** in the literature, they are all equivalent for the cases of classical and intuitionistic logic but are not equivalent in the context of this section. The version in the main text we call **Transitivity** (**Lemma Generation**). Another version is:

$$\frac{\Gamma \mid\!\sim A \quad \Delta, A \mid\!\sim B}{\Delta, \Gamma \mid\!\sim B}.$$

This version implies **monotonicity**, when added to **Reflexivity**.
Another version we call **Internal Cut**:

$$\frac{\Delta, A \mid\!\sim \Gamma \quad \Delta \mid\!\sim A, \Gamma}{\Delta \mid\!\sim \Gamma}.$$

A more restricted version of cut is **Unitary Cut**:

$$\frac{\Delta \mid\!\sim A; A \mid\!\sim Q}{\Delta \mid\!\sim Q}$$

some are model theoretical. All these different presentations give rise to some notion of consequence $\Delta \mid\!\sim Q$, but they only seem to all agree on some form of restricted reflexivity ($A \mid\!\sim A$). The essential difference between these logics (commonly called *non-monotonic logics*) and the more traditional logics (now referred to as *monotonic logics*) is the fact that $\Delta \mid\!\sim A$ holds in the monotonic case because of some $\Delta_A \subseteq \Delta$, while in the non monotonic case the entire set Δ is used to derive A. Thus if Δ is increased to Δ', there is no change in the monotonic case, while there may be a change in the non monotonic case.

The above describes the situation current in the early 1980's. We have had a multitude of systems generally accepted as 'logics' without a unifying underlying theory and many had semantics without proof theory. Many had proof theory without semantics, though almost all of them were based on some sound intuitions of one form or another. Clearly there was the need for a general unifying framework. An early attempt at classifying non-monotonic systems was [Gabbay, 1985]. It was put forward that basic axioms for a consequence relation should be *reflexivity, transitivity (cut)* and *restricted monotonicity*, namely:

Restricted Monotonicity

$$\frac{\Delta \mid\!\sim A; \Delta \mid\!\sim B}{\Delta, A \mid\!\sim B}$$

A variety of systems seem to satisfy this axiom. Further results were obtained [Kraus *et al.*, 1990, Makinson, 1989, Lehmann and Magidor, 1992, Wójcicki, 1988, Wójcicki, to appear, Makinson, 1989] and the area was called 'axiomatic theory of the consequence relation' by Wojcicki.[9]

Although some classification was obtained and semantical results were proved, the approach does not seem to be strong enough. Many systems do not satisfy restricted monotonicity. Other systems such as relevance logic, do not satisfy even reflexivity. Others have richness of their own which is lost in a simple presentation as an axiomatic consequence relation. Obviously a different approach is needed, one which would be more sensitive to the variety of features of the systems in the field. Fortunately, developments in a neighbouring area, that of automated deduction, seem to give us a clue.

2.2 Logical systems as algorithmic proof systems

The relative importance of automated deduction is on the increase, in view of its wide applicability. New automated deduction methods have been developed for non-classical logics, and resolution has been generalised and modified to be applicable to these logics. In general, because of the value of these logics in theoretical computer science and artificial intelligence, a greater awareness of the computational aspects of logical systems is developing and more attention being devoted to proof theoretical presentations. It became apparent to us that a key feature in the proof theoretic study of these logics is that a slight natural variation in an automated or proof theoretic system of one logic (say $\mathbf{L}_1$), can yield another logic (say $\mathbf{L}_2$).

Although $\mathbf{L}_1$ and $\mathbf{L}_2$ may be conceptually far apart (in their philosophical motivation, and mathematical definitions) when it comes to automated techniques and proof theoretical presentation, they turn out to be brother and sister. This kind of relationship is not isolated and seems to be widespread. Furthermore, non monotonic systems seem to be obtainable from monotonic ones through variations on some of their monotonic proof theoretical formulation.

This seems to give us some handle on classifying non-monotonic systems.

This phenomena has prompted us to put forward the view that a logical system $\mathbf{L}$ is not just the traditional consequence relation $\vdash$ (monotonic or non monotonic) but a pair $(\mid\!\sim, \mathbf{S}_{\mid\sim})$, where $\vdash$ is a mathematically defined consequence relation (ie the set of pairs (Δ, Γ) such that $\Delta \mid\!\sim \Gamma$) satisfying whatever minimal conditions on a consequence one happens to agree to, and $\mathbf{S}_{\mid\sim}$ is an algorithmic system for generating all those pairs [Gabbay, 1992a]. Thus according to this definition classical propositional logic $\mid\!\sim$ perceived as a set of tautologies together with a Gentzen system $\mathbf{S}_{\mid\sim}$

[9] In general, the exact formulations of transitivity and reflexivity can force some form of monotonicity.

is not the same as classical logic together with the two valued truth table decision procedure $\mathbf{T}_{\mid\sim}$ for it. In our conceptual framework,($\mid\sim, \mathbf{S}_{\mid\sim}$) is *not the same logic* as ($\mid\sim, \mathbf{T}_{\mid\sim}$).

To illustrate and motivate our way of thinking, observe that it is very easy to move from $\mathbf{T}_{\mid\sim}$ for classical logic to a truth table system $\mathbf{T}^n_{\mid\sim}$ for Łukasiewicz n-valued logic. It is not so easy to move to an algorithmic system for intuitionistic logic. In comparison, for a Gentzen system presentation, exactly the opposite is true. Intuitionistic and classical logics are neighbours, while Lukasiewicz logics seem completely different. In fact for a Hilbert style or Gentzen style formulation, one can show proof theoretic similarities between Łukasiewicz's infinite valued logic and Girard's Linear Logic, which in turn is proof theoretically similar to intuitionistic logic.

This issue has a bearing on the notion of 'what is classical logic', in other words, the question of what is it that we support as a universal system? Given an algorithmic proof system $\mathbf{S}_{\mid\sim_c}$ for classical logic $\mid\sim_c$, then ($\mid\sim_c, \mathbf{S}_{\mid\sim_c}$) is certainly classical logic. Now suppose we change $\mathbf{S}_{\mid\sim_c}$ a bit by adding heuristics to obtain $\mathbf{S}'$. The heuristics and modifications are needed to support an application area. Can we still say that 'classical logic supports the application area' and therefore we have here yet another instance of the universality of classical logic? I suppose we can because $\mathbf{S}'$ is just a slight modification of $\mathbf{S}_{\mid\sim_c}$. However, slight modifications of an algorithmic system may yield another well-known logic. In fact $\mathbf{S}'$ may be linear logic. So is classical logic supporting the application or is linear logic? Or perhaps when we say 'classical logic is universal' we mean 'A certain proof methodology (e.g. resolution) and all its slight modifications are universal', no matter what logic the modification is.

We give an example from goal directed implicational logic. Consider a language with implication only. It is easy to see that all wffs have the form $A_1 \to (A_2 \to \ldots \to (A_n \to q)\ldots)$, q atomic. We now describe a computation with database a multiset Δ of wffs of the above form and the goal a wff of the above form. We use the metapredicate $\Delta \vdash A$ to mean the computation succeeds; i.e. A follows from Δ. Here are the rules:

1. $\Delta, q \vdash q$, q atomic and Δ empty. (Note that we are not writing $A \vdash A$ for arbitrary A. This is not a Gentzen system).

2. $\Delta \vdash A_1 \to (A_2 \to \ldots \to (A_n \to q)\ldots)$ if $\Delta \cup \{A_1, \ldots, A_n\} \vdash q$. Remember we are dealing with multisets.

3. $\Delta' = \Delta \cup \{A_1 \to (A_2 \to \ldots (A_n \to q)\ldots)\} \vdash q$ if $\Delta = \Delta_1 \cup \ldots \cup \Delta_n$, $\Delta_i, i = 1, \ldots, n$ are pairwise disjoint and $\Delta_i \vdash A_i$.

The above computation characterises linear implication. If we relinquish the side condition in (3) and let $\Delta_i = \Delta'$ and the side condition (1) that Δ is empty, we get intuitionistic implication.

The difference in logics is serious. In terms of proof methodologies, the difference is minor.

2.3 Logical systems as algorithmic structured consequence relations

Further observation of field examples shows that in many cases the database is not just a set of formulas but a structured set of formulas. The most common is a list or multiset.[10] Such structures appear already in linear and concatenation logics and in many non-monotonic systems such as priority and inheritance systems. In many algorithmically presented systems much use is made (either explicitly or implicitly) of this additional structure.

A very common example is a Horn clause program. The list of clauses

$$\begin{array}{ll} (a_1) & q \\ (a_2) & q \to q \end{array}$$

does not behave in the same way as the list

$$\begin{array}{ll} (b_1) & q \to q \\ (b_2) & q \end{array}$$

[10]Classical logic cannot make these distinctions using conjunction only. It needs further annotation or use of predicates.

The query $?q$ succeeds from one and loops from the other. This example is studied in more detail in Section 3.

It is necessary to formulate axioms and notions of consequence relations for structures. This is studied in detail in [Gabbay, 1993a]. Here are the main features.

- Databases (Assumptions) are structured. They are not just sets of formulas but have a more general structure such as multisets, lists, partially ordered sets, etc. To present a database formally, we need to describe the structures. Let $\mathcal{M}$ be a class of structures (e.g. all finite trees). Then a database Δ has the form $\Delta = (M, \mathbf{f})$, where $M \in \mathcal{M}$ and $\mathbf{f} : M \mapsto$ wffs, such that for each $t \in M, \mathbf{f}(t)$ is a formula. We assume the one point structure $\{t\}$ is always in $\mathcal{M}$. We also assume we know how to take any single point $t \in M$ out of M and obtain $(M', \mathbf{f}'), \mathbf{f}' = \mathbf{f} \upharpoonright M$.

- A structured-consequence relation $\mid\!\sim$ is a relation $\Delta \mid\!\sim A$ between structured databases Δ and formulas A.

- $\mid\!\sim$ must satisfy the minimal conditions, namely

  Identity

  $$\{A\} \mid\!\sim A$$

  Surgical Cut

  $$\frac{\Delta \mid\!\sim A; \Gamma[A] \mid\!\sim B}{\Gamma[\Delta] \mid\!\sim B}$$

  where $\Gamma[A]$ means that A resides somewhere in the structure Γ and $\Gamma[\Delta]$ means that Δ replaces A in the structure. These concepts have to be defined precisely. If $\Delta = (M_1, \mathbf{f}_1)$ and $\Gamma = (M_2, \mathbf{f}_2)$ then $\Gamma[A]$ displays the fact that for some $t \in M_2$, $\mathbf{f}_2(t) = A$. We allow for the case that $M_2 = \mathbf{f}_2 = \varnothing$ (i.e. taking A out) We need a notion of substitution, which is a binary function $\mathbf{Sub}(\Gamma, \Delta, t)$, meaning that for $t \in M_2$ we substitute M_1 in place of t. This gives us a structure $(M_3, \mathbf{f}_3)$ according to the definition of $\mathbf{Sub}$. $(M_3, \mathbf{f}_3)$ is displayed as $\Gamma[\Delta]$, and $\Gamma[\varnothing]$ displays the case of taking A out.
  Many non monotonic systems satisfy a more restricted version of surgical cut:

  $$\frac{\Gamma[\varnothing/A] \mid\!\sim A; \Gamma[A] \mid\!\sim B}{\Gamma[\Gamma[\varnothing/A]] \mid\!\sim B}$$

  Another variant would be
  Deletional Cut

  $$\frac{\Gamma[\varnothing/A] \mid\!\sim A; \Gamma[A] \mid\!\sim B}{\Gamma[\varnothing/A] \mid\!\sim B}$$

  .

- A logical system is a pair $(\mid\!\sim, \mathbf{S}_{\mid\sim})$, where $\vdash$ is a structured-consequence and $\mathbf{S}_{\mid\sim}$ is an algorithmic system for it.

Of course we continue to maintain our view that different algorithmic systems for the same structured consequence relation define different logics. Still although we now have a fairly general concept of a logic, we do not have a general framework. Monotonic and non-monotonic systems still seem conceptually different. There are many diverse examples among temporal logics, modal

logics, defeasible logics and more. Obviously, there is a need for a more unifying framework. The question is, can we adopt a concept of a logic where the passage from one logic to another is natural, and along predefined acceptable modes of variation? Can we put forward a framework where the computational aspects of a logic also play a role? Is it possible to find a common home for a variety of seemingly different techniques introduced for different purposes in seemingly different intellectual logical traditions?

2.4 Logical systems as labelled deductive systems

To find an answer, let us ask ourselves what makes one logic different from another? How is a new logic presented and described and compared to another? The answer is obvious. These considerations are performed in the metalevel. Most logics are based on modus ponens anyway. The quantifier rules are formally the same anyway and the differences between them are metalevel considerations on the proof theory or semantics. If we can find a mode of presentation of logical systems where metalevel features can reside side by side with object level features then we can hope for a general framework. We must be careful here. In the logical community the notions of object-level vs metalevel are not so clear. Most people think of *naming* and *proof predicates* in this connection. This is not what we mean by metalevel here. We need a more refined understanding of the concept. There is a similar need in computer science. We shall discuss these topics in a later section.

We found that the best framework to put forward is that of a *Labelled Deductive System, LDS*. Our notion of what is a logic is that of a pair $(\mid\!\sim, \mathbf{S}_{\mid\sim})$ where $\vdash$ is a structured (possibly non-monotonic) consequence relation on a language $\mathbf{L}$ and $\mathbf{S}_{\mid\sim}$ is an *LDS*, and where $\vdash$ is essentially required to satisfy no more than *Identity* (i.e. $\{A\} \mid\!\sim A$) and a version of *Cut*. This is a refinement of our concept of a logical system presented in [Gabbay, 1992a]. We now not only say that a logical system is a pair $(\mid\!\sim, \mathbf{S}_{\mid\sim})$, but we are adding that $\mathbf{S}_{\vdash}$ itself has a special presentation, that of an *LDS*.

As a first approximation, we can say that an *LDS* system is a triple $(\mathbf{L}, \mathcal{A}, \mathbf{M})$, where $\mathbf{L}$ is a logical language (connectives and wffs) and $\mathcal{A}$ is an algebra (with some operations) of labels and $\mathbf{M}$ is a discipline of labelling formulas of the logic (from the algebra of labels $\mathcal{A}$), together with deduction rules and with agreed ways of propagating the labels via the application of the deduction rules. The way the rules are used is more or less uniform to all systems.

To present an *LDS* system we need first to define its set of formulas and its set of labels.

For example, we can take the language of classical logic as the formulas (with variables and quantifiers) and take some set of function symbols on the same variables as generating the labels. More precisely, we allow for ordinary formulas of predicate logic with quantifiers to be our *LDS* formulas. Thus $\exists x A(x, y)$ is a formula with free variable y and bound variable x. To generate the labels, we start with a new set of function symbols $t_1(y), t_2(x, y), \ldots$ of various arities which can be applied to the *same* variables which appear in the formulas. Thus the labels and formulas can share variables. We can form declarative units of the form $t_1(y) : \exists x A(x, y)$. When y is assigned a value $y = a$, so does the label and we get $t_1(a) : \exists x A(x, a)$. The labels should be viewed as more information about the formulas, which is not coded inside the formula, (hence dependence of the labels on variables x makes sense as the extra information may be different for different x). A formal definition of an algebraic *LDS* system will be given later, meanwhile, let us give an informal definition of an *LDS* system and some examples which help us understand what and why we would want labels.[11]

[11]The idea of annotating formulas for various purposes is not new. A R Anderson and N. Belnap in their book on Entailment, label formulas and propagate lables during proofs to keep track of relevance of assumptions. Term annotations (Curry–Howard formula as type approach) are also known where the propagation rules are functional application. The Lambek Calculus and the categorial approach is also related to labelling. What is new is that we are proposing that we use an arbitrary algebra for the labels and consider the labelling as part of the logic. We are creating a discipline here of *LDS* and claiming that we have a unifying framework for logics and that almost any logic can be given an *LDS* formulation. We are claiming that the notion of a logic is an *LDS*. This is not the same as the occasional use of labelling with some specific purpose in mind. We are translating and investigating systematically all the traditional logical concepts into the context of *LDS* and generalising them.

I am reminded of the story of the Yuppy who hired an interior decorator to redesign his sitting room. After much

Definition 2.1 [Prototype *LDS* System] Let $\mathcal{A}$ be a first order language of the form $\mathcal{A} = (A, R_1, \ldots, R_k, f_1, \ldots, f_m)$ where A is the set of terms of the algebra (individual variables and constants) and R_i are predicates (on A, possibly binary but not necessarily so) and $f_1, \ldots, f_m$ are function symbols (on A) of various arities. We think of the elements of A as atomic labels and of the functions as generating more labels and of the relations as giving additional structure to the labels. A typical example would be (A, R, f_1, f_2) where R is binary and f_1, f_2 are unary.
A *diagram* of labels is a set M containing elements generated from A by the function symbols together with predicates of the form $\pm R(t_1, \ldots, t_k)$, where $t_i \in M$ and R is a relation of the algebra.
Let $\mathbf{L}$ be a predicate language with connectives $\sharp_1, \ldots, \sharp_n$, of various arities, with quantifiers and with the *same* set of atomic terms A as the algebra.
We define the notions of a declarative unit, a database and a label as follows:

1. An atomic label is any $t \in A$. A label is any term generated from the atomic labels by the functions $f_1, \ldots, f_m$.
2. A formula is any formula of $\mathbf{L}$.
3. A declarative unit is a pair $t : A$, where t is a label and A is a formula.
4. A database is either a declarative unit or has the form $(a, M, \mathbf{f})$, where M is a finite diagram of labels, $a \in M$ is the distinguished label, and $\mathbf{f}$ is a function associating with each label t in M either a database or a finite set of formulas.

Definition 2.4.1 is simplified. To understand it intuitively, think of the atomic labels as atomic places and times (top of the hill, Jan 1st 1992, etc.) and the function symbols as generating more labels, namely more times and more places (*behind*(x), *day after*(t) etc.). We form declarative units by taking labels and attaching formulas to them. Complex structures $(M, \mathbf{f})$ of these units are databases. This definition can be made more complex. Here the labels are terms generated by function symbols from atomic labels. We can complicate matters by using databases themselves as labels. This will give us recursively more complex, richer labels. We will not go into that now. The first simplification is therefore that we are not using databases as labels. The second simplification is that we assume constant domains. All times and places have the same elements (population) in them. If this were not the case we would need a function U_t giving the elements residing in t, and a database would have the form $(A, M, \mathbf{f}, U_t)$. This point will be taken up in a later section.

Example 2.2 Consider a language with the predicate *VS900*(x, t). This is a two sorted predicate, denoting Virgin airline flight London-Tokyo, where t is the flight date and x is a name of an individual. For example *VS*900 (Dov, 15.11.91) may be put in the database, denoted that Dov is booked on this flight scheduled to embark on 15.11.91.
If the airline practices overbooking and cancellation procedures (whatever that means), it might wish to annotate the entries by further useful information such as

- Time of booking;
- Individual/group travel booking;
- Type of ticket
- $\pm$ VIP.

This information may be of a different nature to that coded in the main predicate and it is therefore more convenient to keep it as annotation, or label. It may also be the case that the manipulation of the extra information is of a different nature to that of the predicate.

---

study, the decorator recommended that the Yuppy needed a feeling of space and so the best thing to do is to arrange the furniture against the wall, so that there will be a lot of space in the middle. The cleaning lady, when she first saw the new design was very pleased. She thought it was an excellent idea. 'Yes', said the Yuppy, 'and I paid £1000 for it'. 'That was stupid', exclaimed the cleaning lady, 'I could have told you for free! I arrange the furniture this way every time I clean the floor!'.
Of course she is right, but she used the idea of the new arrangement only as a side effect!

In general, there may be many uses for the label t in the declarative unit $t : A$. Here is a partial list

- Fuzzy reliability value:
  (a number $x, 0 \leq x \leq 1$.) Used mainly in expert systems.
- Origin of A:
  t indicates where the input A came from. Very useful in complex databases.
- Priority of A:
  t can be a date of entry of updates and a later date (label) means a higher priority.
- Time when A holds:
  (temporal logic)
- Possible world where A holds:
  (modal logic)
- t indicates the proof of A:
  (which assumptions were used in deriving A and the history of the proof). This is a useful labelling for Truth Maintenance Systems.
- t can be the situation and A the infon (of situation semantics).

Example 2.3 Let us look at one particular example, connected with modal logic. Assume the algebra $\mathcal{A}$ has the form $(A, <)$, with a set of atomic labels A, no function symbols and a binary relation $<$. According to the previous definition, a diagram of labels, would contain a (finite) set $M \subseteq A$, together with a set of pairs of the form $\{t < s\}$, t, $s, \in M$. A database has the form $(a, M, \mathbf{f})$, where M is a finite diagram and $\mathbf{f}$ is a function, say giving a formula $A_t = \mathbf{f}(t)$, for each $t \in M$.

The perceptive reader may feel resistence to the idea of the label at this stage. First be assured that you are not asked to give up your favourite logic or proof theory nor is there any hint of a claim that your activity is now obsolete. In mathematics a good concept can rarely be seen or studied from one point of view only and it is a sign of strength to have several views connecting different concepts. So the traditional logical views are as valid as ever and add strength to the new point of view. In fact, manifestations of our *LDS* approach already exist in the literature in various forms, they were locally regarded as convenient tools and there was not the realisation that there is a general framework to be studied and developed. None of us is working in a vacuum and we build on each others work. Further, the existence of a general framework in which any particular case can be represented does not necessarily mean that the best way to treat that particular case is within the general framework. Thus if some modal logics can be formulated in *LDS*, this does not mean that in practice we should replace existing ways of treating the logics by their *LDS* formulation. The latter may not be the most efficient for those particular logics. It is sufficient to show how the *LDS* principles specialise and manifest themselves in the given known practical formulation of the logic.

The reader may further have doubts about the use of labels from the computational point of view. What do we mean by a unifying framework? Surely a Turing machine can simulate any logic, is that a unifying framework? The use of labels is powerful, as we know from computer science. Are we using labels to play the role of a Turing machine? The answer to the question is twofold. First that we are not operating at the metalevel, but at the object level. Second, there are severe restrictions on the way we use *LDS*. Here is a preview:

1. The only rules of inference allowed are the traditional ones, modus ponens and some form of deduction theorem for implication, for example.
2. Allowable modes of label propagation are fixed for all logics. They can be adjusted in agreed ways to obtain variations but in general the format is the same. For example, it has the

following form for implications:
Let $LDS_{\twoheadrightarrow}$ be a particular LDS system with labels $\mathcal{A}$, and with $\twoheadrightarrow$ a special implication characteristic to this particular LDS system. Then there exists a fixed set of labels Γ, which can characterise $\twoheadrightarrow$ as follows. For any theory Δ (of labelled wffs) we have:
Δ proves $(A \twoheadrightarrow B)$ with label t iff $\forall x \in \Gamma$ [If A is labelled x then B can be proved from Δ and $x : A$ with labels $t + x$],
where Γ is a set of labels characterising the implication in that particular logic. For example Γ may be all atomic labels or related labels to t, or variations. The freedom that different logics have is in the choice of Γ and the (possibly not only equational) properties of '+'. For example we can restrict the use of modus ponens by a wise propagation of labels.

3. The quantifier rules are the same for all logics.

4. Metalevel features are implemented via the labelling mechanism, which is object language.

The reader who prefers to remain within the traditional point of view of:

assumptions (data) proving a conclusion

can view the labelled formulas as another form of data.

There are many occasions when it is most intuitive to present an item of data in the form $t : A$, where t is a label and A is a formula. The common underlying reason for the use of the label t is that t represents information which is needed to modify A or to supplement (the information in) A which is not of the same type or nature as (the information represented by) A itself. A is a logical formula representing information declaratively, and the additional information of t can certainly be added declaratively to A to form A', however, we may find it convenient to put forward the additional information through the label t as part of a pair $t : A$.

Take for example a source of information which is not reliable. A natural way of representing an item of information from that source is $t : A$, where A is a declarative presentation of the information itself and t is a number representing its reliability. Such expert systems exist (eg Mycin) with rules which manipulate both t and A as one unit, propagating the reliability values t_i through applications of modus ponens. We may also use a label naming the source of information and this would give us a qualitative idea of its reliability.

Another area where it is natural to use labels is in reasoning from data and rules. If we want to keep track, for reasons of maintaining consistency and/or integrity constraints, where and how a formula was deduced, we use a label t. In this case, the label t in $t : A$ can be the part of the data which was used to get A. Formally in this case t is a formula, the conjunction of the data used. We thus get pairs of the form $\Delta_i : A_i$, where A_i are formulas and Δ_i are the parts of the database from which A_i was derived.

A third example where it is natural to use labels is time stamping of data. Where data is constantly revised and updated, it is important to time stamp the data items. Thus the data items would look like $t_i : A_i$, where t_i are time stamps. A_i itself may be a temporal formula. Thus there are two times involved, the logical time s_i in $A_i(s_i)$ and the time stamping t_i of A_i. For reasons of clarity, we may wish to regard t_i as a label rather than incorporate it into the logic (by writing for example $A^*(t_i, s_i)$).

To summarise then, we replace the traditional notion of consequence between formulas of the form $A_1, \ldots, A_n \vdash B$ by the notion of consequence between labelled formulas

$$t_1 : A_1, t_2 : A_2, \ldots t_n : A_n \vdash s : B$$

Depending on the logical system involved, the intuitive meaning of the labels vary. In querying databases, we may be interested in labelling the assumptions so that when we get an answer to a query, we can record, via the label of the answer, from which part of the database the answer was obtained. Another area where labelling is used is temporal logic. We can time stamp assumptions as to when they are true and query, given those assumptions, whether a certain conclusion will be

true at a certain time. Thus the consequence notion for labelled deduction is essentially the same as that of any logic: given assumptions does a conclusion follow.

Whereas in the traditional logical system the consequence is defined using proof rules on the formulas, in the *LDS* methodology the consequence is defined by using rules on both formulas and their labels. Formally we have formal rules for manipulating labels and this allows for more scope in decomposing the various features of the consequence relation. The meta features can be reflected in the algebra or logic of the labels and the object features can be reflected in the rules of the formulas.

The notion of a database or of a 'set of assumptions' also has to be changed. A database is a hierarchical configuration of labelled formulas. The configuration depends on the labelling discipline. For example, it can be a linearly ordered set $\{a_1 : A_1, \ldots, a_n : A_n\}, a_1 < a_2 < \ldots < a_n$. The proof discipline for the logic will specify how the assumptions are to be used. See for example the logic programming case study.

We summarise our current position on what is a logical system. A logical system is a pair $(\mid\!\sim, LDS_{\mid\!\sim})$, where $\mid\!\sim$ is a consequence relation between labelled databases Δ and declarative units $t : A$ and $LDS_{\mid\!\sim}$ is an algorithmic system for $\mid\!\sim$.

We need one more component to the notion of a logical system. In previous subsections, a logical system was presented as $(\mid\!\sim, \mathbf{S}_{\mid\!\sim})$, where $\mid\!\sim$ is a structured consequence relation satisfying *Identity* and *Surgical Cut* and $\mathbf{S}_{\mid\!\sim}$ is an algorithmic proof system for computing $\mid\!\sim$. We are now saying that we need to refine this notion and deal with *Labelled Deductive Systems*, where $\mid\!\sim$ is a consequence relation between labelled databases Δ and declarative units $t : A$ and that $\mathbf{S}_{\mid\!\sim}$ is replaced by some specific *LDS* discipline (algorithm) for computing the above. We need to be able to retrieve the old notion i.e. $(\mid\!\sim_1, \mathbf{S}_{\mid\!\sim_1})$ from the new notion $(\mid\!\sim_2, LDS_{\mid\!\sim_2})$. In other words, we must add into *LDS* the capability of proving and reasoning without labels. To achieve this we can first reason with labels and then strip the labels and give a conclusion without labels. The additional algorithm which we can use to strip the labels is called *flattening*. Thus a labelled theory Δ may prove $t_i : A$ and $s_i : \neg A$, with many different labels t_i and s_i, depending on various labelling considerations and proof paths. The *flattening* algorithm will allow us to decide whether we flatten the pair of sets $(\{t_i\}, \{s_i\})$ to + or - i.e. whether we say $\Delta \mid\!\sim A$ or $\Delta \mid\!\sim \neg A$.

For example, if the labels represent moments of time or priorities, we may say the value is + if $\max\{t_i\} \geq \max\{s_j\}$. Or we may *interlace* the flattening with the deduction itself.

Thus, given a structured theory Δ (without labels) and a candidate A, we can have the following procedure, using *LDS*, of deciding whether $\Delta \mid\!\sim ? A$.

- Label the elements of Δ with completely different atomic labels, representing existing structure in Δ.
- Use the *LDS* machinery to deduce all possibilities $\Delta \mid\!\sim t_i : A$ and $\Delta \mid\!\sim s_i : \neg A$.
- Flatten and get A (or interlace with flattening and get A).

Example 2.4 Here is an example of interlacing. The database has

$$\begin{aligned} &t_1 : A \\ &t_2 : \neg A \\ &t_3 : \neg A \rightarrow B \\ &t_4 : A \rightarrow \neg B. \end{aligned}$$

Assume priority is $t_1 < t_2 < t_3 < t_4$, and assume a flattening process which gives higher priority rules superiority over low priority rules and similarly for facts but gives lexicographic superiority for rules over facts. Thus $t_4 t_1$ is stronger than $t_3 t_2$. If we deduce and then flatten, we get

$$\begin{aligned} &t_4 t_1 : \neg B \\ &t_3 t_2 : B \end{aligned}$$

The flattening process would take $\neg B$.
If we pursue an interlace argument, we first flatten the premisses and take $\neg A$ and then perform the modus ponens and get B.

2.5 Aggregated systems

So far all our logical systems either proved or not proved a conclusion A. There was no possibility of aggregating arguments in favour of A as against arguments in favour of negation of A. The lack of aggregation is a basic characteristic which currently separates the symbolic qualitative school of reasoning from the numerical, quantitative one.

There are many systems around (many are recognised as probabilistic systems, expert systems, fuzzy systems) which attach numerical values to assumptions and rules, use various functions and numerical procedures to manipulate these values and formulas and reach conclusions.

In many cases we get systems which give answers which seem to make sense, which can be very successfully and profitably applied but which cannot be recognised or understood by traditional logic. The main feature common to all of these numerical systems (which is independent of how they calculate and propagate their values) is that their 'proofs' aggregate. They can add the numbers involved and thus aggregate arguments. The spirit is: *Five good rumours are better than one proof.*

To further illustrate, consider the following example

Example 2.5 The assumptions are:

$$\begin{array}{l} t_1 : A \to C \\ t_2 : B \to C \\ t_3 : A \\ t_4 : B \\ t_5 : D \to \neg C \\ t_6 : D \end{array}$$

Here we can conclude C in two different ways and conclude $\neg C$ in one way.
Non monotonic systems like defeasible logic will not allow us to draw any conclusion unless one rule defeats all others. If we had a numerical evaluation of the data, say t_i are numbers in [0 1], then we could aggregate our confidence in the conclusion. Thus we get:

$$\begin{array}{c} (t_1 \cdot t_3 + t_2 \cdot t_4) : C \\ t_5 \cdot t_6 : \neg C \end{array}$$

the two numbers can be compared and a conclusion reached.

If we operate in the context of *LDS*, we can use the labels to aggregate arguments. Any conclusion is proved with a label indicating its proof path. These can be formally (algebraically) added (aggregated) and an additional process (called *flattening*) can compare them.

In consequence relation terms, the property of aggregation destroys the cut rule. The reason is as follows:

Assume $\Delta, A \mid\!\sim B$. This now means that the aggregated proofs in favour of B are stronger than the aggregated proofs in favour of $\neg B$. Similarly $\Gamma \vdash A$ would mean the balance from Γ is in favour of A.

If we perform the cut we get

$$\Delta, \Gamma \mid\!\sim ? B$$

Δ and Γ may interact, forming new additional proofs of $\neg B$, which outweigh the proofs for B.

Cut is a very basic rule in traditional logical systems and can be found in one form or another in each one of them. Thus it is clear that aggregation of arguments is a serious departure from traditional formal logic. Yet, it cannot be denied. In practical reasoning we do aggregate arguments and so logic, if it is to be successfully applied and be able to mirror human reasoning, must be able to cope with aggregation. Classical logic, if it is to be a universal language, must also be able to deal naturally with aggregation.

One form of cut is still valid. The unitary cut.

$$\frac{\Delta \mid\!\sim A; A \mid\!\sim B}{\Delta \mid\!\sim B}$$

This holds because there is nothing for Δ to interact with.

We thus require from our reasoning system to satisfy only *Identity* ($A|\!\sim\!A$) and *Unitary Cut*.

To show how real and possibly destructive aggregation can be, consider the example of Prince Karlos and Princess Laura.

Example 2.6 The prince and princess are separated. Both made it clear to the press that no third parties were involved and the separation was purely due to a personality clash. However, the editor Mr Angel of the *Daily Tabloid*, thought otherwise. First he observed that after her separation the princess moved to a house very near the Imperial Institute of Logic, Language and Computation. This in itself did not mean much, because both the Institute and the residence were in the centre of town. However, Mr Angel further found out that in the past two years, whenever the princess went on a European holiday, there was an Esprit project meeting in the same hotel, and surprisingly all projects involve a certain professor from the Institute. Again, this could be a coincidence, because it is a well known fact that Esprit project consortia find it most inspiring to be in the most expensive holiday resorts in Europe and it is equally well known that certain dynamic professors participate in many such projects.
However the plot thickens when the princess, as part of her general social activity seems to actively support the new logics for computation. This could also be a coincidence because after all, this subject is going to transform the nature of our society. The various little arguments do seem to be aggregating, though not conclusively enough to risk an article in such a responsible paper as the *Daily Tabloid*. The situation changed when it became known that the princess actively supports Labelled Deductive Systems and the Universality of classical logic. Under this aggregation of arguments an obvious conclusion could be drawn!

2.6 Practical reasoning systems

Our discussion so far generalised the notion of a *deductive system*, namely, given a database Δ and a formula Q, we ask the basic question, does Δ prove Q? The various concepts we studied had to do with what form do Δ and Q take and what kinds of consequence relations $\mid\!\sim$ and algorithmic systems $\mathbf{S}_{\mid\sim}$ are involved.

In practical reasoning systems, the *deductive* question is but one of many which interest us. Other operations such as *updating, abduction, action, explanation* are also involved. If we rethink of Q as an *input*, we can partially list the kind of of operations which may be involved. These operations are performed using algorithms which accompany the deductive component. We refer to them as *mechanisms*.

- The input Q is a query from Δ. We are interested in whether $\Delta \mid\!\sim Q$ and possibly ask what proofs are available.
- The input Q is an update. We want to insert Q into Δ to obtain Δ'. We may possibly have to deal with inconsistency and restructuring of Δ.[12]
- The input Q is an abductive stimulus (goal). We are interested in Δ' such that $\Delta + \Delta' \mid\!\sim Q$. Where $+$ is a symbol (to be precisely defined) which 'adds' or 'joins' Δ and Δ' to 'combine' their declarative information.

  The $+$ operation may or may not be the same as update. The abductive question is to find (possibly the minimal) Δ' which helps prove the input.[13]
- The input Q may be a stimulus for action on the database outputting a new database or outputting an explanation or any other output of interest.

[12] Such a view has been presented in Chapter 13 of Bob Kowalski's book [1986]. The systematic study of updates and theory change was initiated in [Alchourrón *et al.*, 1985].

[13] For a given system $(\mid\!\sim, \mathbf{S}_{\mid\sim})$, the abductive mechanism is usually dependent on $\mathbf{S}_{\mid\sim}$, the particular algorithmic proof system involved. Different applications might require different abductive procedures.

The new possibilities of a formula Q interacting with a database Δ, (via action or abduction or other mechanisms.) allow for a new way of answering queries from Δ. To see this, consider the query $\Delta?Q$. In the declarative aspect, we want an answer, namely we are asking whether $\Delta \mid\!\sim Q$. This can be checked via $\mathbf{S}_{\mid\sim}$,or semantically. Q does not act or change Δ in any way. In the interactive case, we trigger an action. Q acts on Δ to produce a new Δ' Q is read imperatively. We can write $\Delta!Q$ to stress this fact. The result of the interaction is Δ'. Thus $\Delta!Q = \Delta'$. Thus given a Δ and a Q, we have two options. We can ask whether $\Delta \mid\sim Q$ (written $\Delta?Q$, with $\mid\sim$ implicit) or we can let Q act on Δ, written $\Delta!Q$, where ! denotes the action. When an action ! is given, it is possible to derive a consequence relation $\mid\sim$ dependent on the action ! (really we should write $\mid\sim_!$). The new $\mid\sim$ is defined by $\Delta \mid\sim Q$ iff $\Delta!Q = \Delta$. This view was particularly put forward by F. Veldman and pursued by J. van Benthem.

To summarize this section, our current last word about the question of what is a logical system is the following:

Definition 2.7 [Current tentative view of logical system]
A logical system has the form $(\mid\sim, \mathbf{S}_{\mid\sim}, \mathbf{S}_{\text{abduce}}, \mathbf{S}_{\text{update}}, \ldots)$ where $\mid\sim$ is a labelled consequence relation, $\mathbf{S}_{\mid\sim}$ is a *Labelled Deductive System* with *Flattening* procedures and $\mathbf{S}_{\text{abduce}}, \mathbf{S}_{\text{update}}$ etc are *mechanisms* which are parasitic (make use of) $\mathbf{S}_{\mid\sim}$.

3 How to construct a logic for an application; the case studies

Let us see at what stage of The Debate we now stand. We have discussed the notion of a logical system and have seen the place of classical logic in the family of logics. We must now examine some case studies in detail to see how classical and non-classical logics compare. For a reader who agrees with our view of logic (Logic = *LDS* + *mechanisms*), the question is whether classical logic can do the job that *LDS* + *mechanism* can? For other readers the question is whether classical logic can successfully handle the case studies which motivated us to introduce *LDS* + *mechanism*. One way or another, we want to see classical logic in action! This brings us to the present section.

The purpose of this section is to present and evaluate the options available to a practically minded researcher wishing to use logic for the representation, reasoning and computation in his application area. This evaluation will provide us with a case study for comparing the merits in using classical logic vs a specialised non classical logic for the application. It will demonstrate for a specific application both the mathematical and conceptual options that are involved.

We begin by listing the properties of classical logic against the (non-classical representation and reasoning) needs of a typical case study.

The basic notions involved in classical logic are the following:

1. The notion of a well formed formula, which is the basic declarative unit available for the representation of the knowledge of the case study.

2. The notion of a database Δ or a theory, which is the aggregation of declarative units. In this case it is a set of wffs.

3. The consequence relation $\mid\sim$ of the form $\Delta \mid\sim A$ between databases and declarative units. This can be presented in various forms, semantically, proof theoretically, etc. Some systems are formulated using consequence relations between two databases $\Delta \mid\sim \Gamma$. This notion for arbitrary Γ can be defined in some cases from the fragment consequence where there is a single declarative unit in Γ.

Different logics, such as intuitionistic or many valued logics, share with classical logic the notions of a declarative unit and a database, but differ on the consequence relation.

In contrast to the above, a typical case study may indeed have identifiable basic semantic declarative units, but these declarative units are naturally organised in a structure. This structure is external to the declarative units. Thus the notion of a database as a set is not found in

applications, but rather the notion of a database as a structured constellation/network of formulas seems more appropriate. The following are examples of sources of structures imposed natrually on declarative units of application areas:

- Time stamps, earlier-later relations among data items.
- Sources of the data. The structure is inherited from the social relationships among the sources.
- Causal relations among data (in e.g. medical networks).
- Accessibility relations among formulas holding in different possible worlds.
- Priorities among data.
- Artificial structuring among data clauses for the purpose of efficient computation.
- Numerical stamps, giving reliability or plausibility. The structure is induced from numerical relationships.

The above are enough widespread examples to justify focussing our discussion on a typical case study where the declarative units are structured, say in a graph form with nodes and several types of edges. Fig 1 illustrates such a structure.

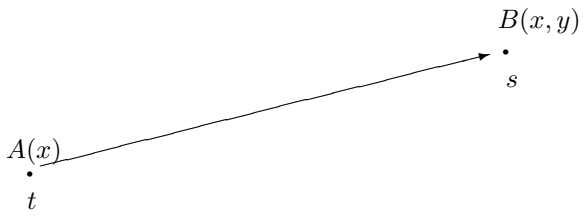


Figure 1:

t and s are two nodes and the arrow is one relation between them. $A(x)$ and $B(x,y)$ are semantic declarative units associated with t and s, involving the objects x and y. We are 'semantic' because we are not yet committed to how we formalise this situation.

We need a logic to describe and reason about the above structure. Suppose we choose to use classical logic to represent the knowledge in A and B. So assume that A and B themselves are already in classical logic. How do we represent the structure of t and s? In the case of classical logic we need to be able to talk directly about t and s. Thus we need to add a new slot to any predicate $P(x_1,\ldots,x_n)$ of classical logic, for the sort of t,s, turning it into $P^*(t,x_1,\ldots,x_n)$ and represent the nodes $t:A(x)$ and $s:B(x,y)$ as $A^*(t,x)$ and $B^*(s,x,y)$ respectively. We need to represent by a special predicate tRs the relation 'arrow' between the nodes. Thus the situation in Fig 1 becomes the following database: $\{tRs, A^*(t,x), B^*(s,x,y)\}$. We need the following features:

1. Augment classical logic into two sorted logic. A new first coordinate is added to each atom to allow for the node elements.
2. Introduce new predicates to express relationships among nodes.
3. Extend classical proof theory to such two (or many) sorted systems.

The many sorted logic is no longer classical logic but a classical logic look-alike. It has more convenient expressive power[14] and seems ideal for our purposes. It has many advantages:

[14] I suppose the sorts can be expressed by unary predicates and domain axioms and so mathematically we remain within first order logic. However sublanguages of classical logic become more expressive when sorts are allowed, while their 'spirit' is maintained. The full generality of the many sorted translation is studied in a later section on Linked Predicate Languages.

- We can use the power of the quantifiers to express relationships among the nodes such as

$$\forall s(tRs \rightarrow \exists x(A^*(s,x)) \wedge \forall u(tRu \wedge sRu \rightarrow \exists y B^*(u,x,y))).$$

- We can use existing powerful classical theorem provers to get (either directly or after modification) automated deduction for the new two sorted logics. Special unification algorithms can also be used.
- The logic is familiar in principle, being a variant of classical logic.
- The method can act as a universal method for many application areas.

At this stage, it looks like that we have answered the question in the title of this section and we can therefore conclude the paper and proceed to give the references. Unfortunately this is not the case. To explain the flaw in this rosy picture, let us examine and analyse a particular temporal example.

3.1 Case study 1: temporal logic in two-sorted classical logic

We consider a linear flow of time $(T, <)$ with predicates of the form $A(v_1)$, $B(v_1, v_2)$. $<$ is irreflexive, transitive and linear. We pass into two sorted predicate logic with $A^*(t, v_1), B^*(t, v_1, v_2)$. We translate the basic declarative statements of temporal logic as follows:

1. $A(v_1)$ holds at t is translated into $A^*(t, v_1)$.
2. Relations among points t, s are expressed directly using $<$ and the quantifiers and connectives of classical logic.
3. Properties of $(T, <)$ are expressed in classical logic.
4. Reasoning is done in classical logic.

The case study considers two databases. The database of Figure 2 which is a pure temporal database and is studied in detail and the database of Figure 3 which is a modal temporal database and is not studied in detail, but is presented as another example to illustrate the complexity of the translation into classical logic. We use the letters t, s as variables of the 'possible world' sort and $v_0, v_1, \ldots$ as variables of the 'element inside possible worlds' sort. x, y, z are general metavariables which can be either sort. So there is a little abuse of notation here.

Figure 2 represents a typical temporal database.

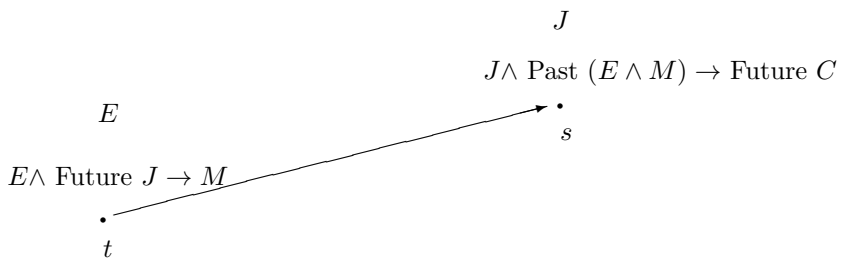


Figure 2:

Dictionary

E = Institute doing well in publications
J = John McCarthy visits Institute
M = Institute gets lots of funds
C = Institute holds an international conference

The translation of the above into two sorted classical logics yields the following in Horn clauses (variables are universally quantified, u_0 is a Skolem constant):

1. $E^*(t)$
2. $E^*(t) \wedge t < y \wedge J^*(y) \rightarrow M^*(t)$
3. $J^*(s)$
4. $J^*(s) \wedge x < s \wedge E^*(x) \wedge M^*(x) \rightarrow s < u_0$
5. $J^*(s) \wedge x < s \wedge E^*(x) \wedge M^*(x) \rightarrow C(u_0)$
6. $t < s$
7. $x < y \wedge y < z \rightarrow x < z$
8. $\sim (x < x)$
9. $x < y \vee x = y \vee y < z$

Clauses 1–6 describe the database and clauses 7–9 are axioms for the linearity of time.

Consider the query ? Future C, asked at time t. In classical logic (Horn clause) we are asking

$$?\exists s(t < s \wedge C(s))$$

The computation will proceed along Prolog lines and will probably terminate. The computation follows a machine oriented depth first strategy.

How would a human reason in such a situation?

Let us write $t : A$ to mean A holds at time t.

1. $t :$ Future J (because $t < s$ and $s : J$)
2. $t : E$ (as given)
3. $t : E \wedge$ Future $J \rightarrow M$ (as given)
4. $t : M$ (by modus ponens)
5. $t : E \wedge M$ (by Adjunction)
6. $s : J$ (as given)
7. $s :$ Past $(E \wedge M)$ (because $t < s$ and $t : E \wedge M$)
8. $s : J \wedge$ Past $(E \wedge M) \rightarrow$ Future C (as given)
9. $s :$ Future C (by modus ponens)
10. So for some u_0 such that $s < u_0$ we must have $u_0 : C$.
11. Since $t < s < u_0$ and $u_0 : C$, we get $t :$ Future C.

Note that we reason locally and separately in each node, together with the mechanism of passing formulas (information) from one node to the other.

Another major approach to temporal and modal reasoning recognises the need for a more intuitive compatibility of the reasoning process. It allows for the use of temporal connectives representing patterns relative to a temporal point where the reasoning is located. Such patterns are expressed in natural language by words like *Tomorrow, Since, Until, Will, Was*, etc.

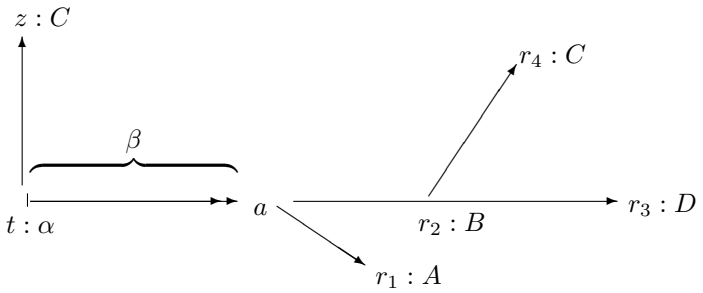


Figure 3:

The pattern of Figure 2 can be expressed as:

$$E \wedge (E \wedge FJ \to M) \wedge F(J \wedge (J \wedge P(E \wedge M) \to FC))$$

from the point of view of the point t. The query at t is $?FC$.

If we adopt the view of doing temporal reasoning in the actual world (i.e. the world where the reasoner is located) using the above patterns[15], we need a proof system in that language. Such systems exist., A Hilbert system for linear temporal logic for F and P has the following axioms and rules, (with $G =\sim F \sim$ and $H =\sim P \sim$):

1.

$$\frac{\vdash A}{\vdash GA}, \quad \frac{\vdash A}{\vdash HA}, \quad \frac{\vdash A, \vdash A \to B}{\vdash B}$$

2.

$$GA \to GGA$$
$$HA \to HHA$$

3.

$$A \to GPA$$
$$A \to HFA$$

4.

$$FA \wedge FB \to F(A \wedge B) \vee F(A \wedge FB) \vee F(FA \wedge B)$$
$$PA \wedge PB \to P(A \wedge B) \vee P(A \wedge PB) \vee P(PA \wedge B).$$

[15]Such patterns can be quite complex. Figure 3 represents a structure around the world which the reasoner inhabits, involving modality and time.

The double arrow connection is temporal and the single arrow is modal. The notation $t : A$ means that A holds at t and the notation $t \overset{A}{\longrightarrow} s$ means that A holds at all points between t and s. If the actual world (point where the reasoner stands) is a, then the above pattern can be expressed as:

$$S(\alpha \wedge \Diamond C, \beta) \wedge \Diamond(B \wedge \Diamond C \wedge \Diamond D) \wedge \Diamond A$$

If modal connection $t \rightsquigarrow s$ is expressed by tRs and temporal connection by $t < s$, then the patterns $F, P, \Diamond, S$ are as follows:

- FA holds at t means: $(\exists s > t)A^*(s)$
- PA holds at t means: $(\exists s < t)A^*(s)$
- $\Diamond A$ holds at t means: $\exists s(tRs \wedge A^*(s))$
- $S(A, B)$ holds at t means:
$$\exists s < t[A^*(s) \wedge \forall u(s < u < t \to B^*(u))]$$
- A (atomic) holds at t is represented by $A^*(t)$.

5.

$$G(A \wedge B) \leftrightarrow GA \wedge GB$$
$$H(A \wedge B) \leftrightarrow HA \wedge HB.$$

6.

$$FFA \rightarrow FA$$
$$PPA \rightarrow PA$$

The deduction of the case study is valid if we can prove in the above system that

$$\vdash E \wedge (E \wedge FJ \rightarrow M) \wedge F(J \wedge (J \wedge P(E \wedge M) \rightarrow FC)) \rightarrow FC$$

We need to show that $1 \wedge 2 \wedge 3 \vdash 4$ below, where

1. E
2. $E \wedge FJ \rightarrow M$
3. $F(J \wedge (J \wedge P(E \wedge M) \rightarrow FC))$
4. FC

To prove this we need several lemmas

- $\vdash F(\alpha \wedge \beta) \rightarrow F\alpha$
- $\vdash FF\alpha \rightarrow F\alpha$
- $\vdash G\alpha \wedge G\beta \rightarrow G(\alpha \wedge \beta)$
- $\vdash \alpha \rightarrow \beta$ implies $\vdash F\alpha \rightarrow F\beta$

Thus from 3 we can get FJ which together with 1 and 2 yields $E \wedge M$.
Since $E \wedge M \rightarrow GP(E \wedge M)$ we get $GP(E \wedge M)$
Also since $\vdash J \wedge (J \wedge P(E \wedge M) \rightarrow FC) \rightarrow P(\wedge M) \rightarrow FC$ we get from 3 that

$$F(P(E \wedge M) \rightarrow FC).$$

From the last two we get

$$F(P(E \wedge M) \wedge (P(E \wedge M) \rightarrow FC))$$

which yields FFC which yields FC.

Again the above proof is not completely intuitive, since all reasoning steps have to be done at the actual world and propagated to far away worlds. Serious difficulties arise in predicate logic where Skolemization is not possible in the actual world for formulas with modalities governing nested quantifiers. Such an example is $\forall xG\exists y(A(x, y))$. The quantifier '$\exists y$' depends not only on the '$\forall x$' but also on the 'G', because for different worlds there may be a different y.

It is obvious that we naturally think in terms of worlds and event time points (for example at the hospital when my first baby was born) and imagine at any given step of reasoning some layout or graph of such points, with different formulas holding at these points (for example, on the way to the hospital, after the baby was born but before he had the operation). Thus some events (time points) are memorable enough to be used as milestones, to have certain predicates holding at some temporal patterns around them, and some have no patterns, but are simply ordinary dates. Natural language is very revealing of how we perceive time. A sentence like 'since 1981 I have not had a holiday' uses both temporal patterns ('since') and specific date points.

In terms of patterns around the actual world, we can say that we allow several worlds to act as 'local' actual worlds and express patterns around them. An additional graph shows the temporal relationship of these local actual worlds.

A typical pattern is three worlds d, r, s with $r < s, d < s$. r, s are variables. d a constant. Consider them as possible worlds. Call this pattern D. With each $t \in D$, we associate a formula A_t which is supposed to hold in the world. Formally this is given by a function $\mathbf{f}(t) = A_t$. These worlds have elements, say people 'living' in them. Suppose c lives at world s (c for Carl). It is important to know where Carl was born. Suppose he was born at world t. We write this as $c(t)$. So all elements living anywhere are represented by functions of where they were born. If $c(t)$ now lives at s, we write $c(t) \in V_s, V_s$ is the set of all elements residing at s. Of course $c(t)$ may reside in several places, e.g. $c(t) \in V_d$, etc.

So we are adopting a compromise position. We accept the local point of view of describing patterns around an actual world, except that we want to allow for several 'local' actual worlds. The relationships between the local actual worlds is described in classical logic. This is a compromise between the two approaches. If we have only one actual world we have the modal approach, but if we have enough 'actual' worlds with only atomic (patterns) formulas describing them, we would be back in the classical logic approach. It is hoped that reasoning with such mixed representation will have the benefits of both approaches without their problems.

The following is a possible definition.

Definition 3.1 1. A Future-past temporal language **KV** (**K** for the fact that the underlying temporal logic is $\mathbf{K}_t$. **V** for the fact that we use variables and non constant domain) has its well formed formulas built up from the classical connectives and quantifiers, predicates, variables and constants and the temporal connectives P and F (with $G =\sim F \sim$ and $H =\sim P \sim$).

2. Let D be a finite directed graph i.e. D is a set of points connected by directed edges. Let $c_1(t), c_2(t), \ldots$ be a list of one variable function symbols. t ranges over D. If $s_1, s_2 \in D$ are connected, we can write $s_1 \rho s_2$. ρ a formal binary predicate.

3. A temporal **KV** theory is a tuple $\tau = (D, \mathbf{f}, d, U)$ where D is a graph, $d \in D$, $\mathbf{f}$ a function associating with each $t \in D$ a formula of the temporal logic $\mathbf{K}_t, \mathbf{f}(t) = A_t$, and U is a function associating with each $t \in D$ a set of closed terms U_t. $c(s) \in U_t$ means that the constant $c(s)$ exists at the node t in the graph D. the label s in $c(s)$ is supposed to mean that c was 'created' at label (world) s.

   So for example at time t John may love a woman called $c(s)$. This woman was born at time s. She may or may not exist at t. If she exists at t (she may have died but John still loves her) we write $c(s) \in U_t$.

   John may love an imaginary woman whom he introduces at time s. This means that we reserve the right to allow that $c(s) \notin U_s$. We have no committment at this stage.

   d is the global actual world, as opposed to the other elements of D which are local actual worlds.

4. Let (S, R, a, V, h) be a Kripke model, that is S is a non empty set, $a \in S, R \subseteq S \times S$ and for each element $s \in S, V_s$ is a non empty domain. h assigns to each atomic predicate Q of arity n and each t a subset $h(t, Q) \subseteq V_t^n$, and for each variable or constant x an element $h(x) \in \bigcup_{t \in S} V_t$. h can be used to define satisfaction for all wffs in the usual way:

$$s \vDash \exists y A(y) \text{ iff } \exists y \in V_s \text{ such that } s \vDash A(y)$$
$$s \vDash FA \text{ iff } \exists s'(sRs' \text{ and } s' \vDash A)$$
$$s \vDash PA \text{ iff } \exists s'(s'Rs \text{ and } s' \vDash A).$$

   We say $(D, \mathbf{f}, d, U)$ holds at a Kripke model (S, R, a, V, h) iff there exists functions $g_1 : D \to S$ and $g_2 : U \to V$ such that (g_1 says which worlds in the Kripke possible world model each element of D is supposed to be, while g_2 says which element of the universe V of the Kripke model each name from U is supposed to be)

   (a) g_1 is one to one with $g_1(d) = a$

(b) If x and y are directionally (not) connected in D, i.e. $\sim x\rho y$ in D, then (not) $g_1(x)Rg_1(y)$

(c) If $\mathbf{f}(x) = A(u_1 \ldots, u_k)$ then $g_1(x) \vDash A(g_2(u_1), \ldots, g_2(u_k))$ (i.e. if A is the wff labelled by $x \in D$ then $g_1(x) \vDash A$ in the model).

The meaning of the definition is that we want the 'configuration' τ to be realised in the model.

5. Let $\tau_1 = (D_1, \mathbf{f}_1, d_1, U_1)$ and $\tau_2 = (D_2, \mathbf{f}_2, d_2, U_2)$ be two configurations. We write $\tau_1 \vDash \tau_2$ iff for all Kripke models (S, R, a, V, h) and all $g_1^1 : D_1 \to S, g_2^1 : U_1 \to V$ for which τ_1 holds at (S, R, a, V, h) we have that for some extensions $g_1^2 : D_1 \cup D_2$, $g_2^2 : U_2 \to V$ the database τ_2 holds in $(S, R, a, V, h, g_1^2, g_2^2)$.

   In other words, in any model in which the configuration τ_1 holds the configuration τ_2 can be made to hold.

6. We say τ is valid iff $\{d : \top, U_d\} \vDash \tau$, where d is the actual world of τ.

Example 3.2 Let τ_1 be as in Fig. 4. and let τ_2 be $d_2 : FA$

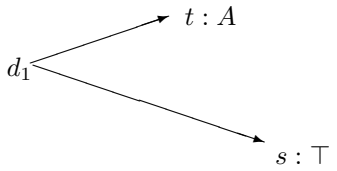


Figure 4:

Clearly $\tau_1 \vDash \tau_2$.
However there is no way in which any wff B can have $d_2 : B \vdash \tau_1$.
If we choose B to be $FA \wedge F\ \top$, we cannot force $s \neq t$. What we need is a 2nd order wff B of the form

$$B = \exists Q[F(A \wedge Q) \wedge F \sim Q].$$

It seems that modal logic and labelled databases can express second order notions. One example of that is the present one. Another is any modal logical system complete for non first order class of frames. For example, the logic **G** of provability, being the extension of modal **K** with

$$\Box A \to \Box\Box A$$
$$\Box(\Box A \to A) \to \Box A$$

This logic is complete for the class of finite partially ordered sets. This class is not first order describable. Thus we cannot directly fully translate this logic into first order logic. Another such logic is the extension of **K** with the McKinsey axiom

$$\Box\Diamond q \to \Diamond\Box q.$$

This is complete for a second order condition on the possible worlds and not first order. This has bearing to the universality of first order logic. The fact is however, there are functional translations into classical logic which can capture second order properties. This happens in the case of the McKinsey axiom, though we need an infinite first order theory of axioms to ensure the correctness of the translation. So the situation is not so simple. The basic mechanisms operating here are the following:

- What is higher order in one language can become lower order in a language with more symbols (e.g. function symbols)

- Translations into (first order) classical logic may use additional symbols. The application areas of logic as well as the automated reasoning machinery is not sensitive (stringent, puts costs on) to additional symbols.

Remark 3.3 [Consequence and proof theory] The reader may wonder what is the consequence relation associated with the previous definitions. In ordinary modal logic, a theory Δ is a set of formulas, for example the formula expressing the pattern of Figure 3. The consequence relation is between a theory Δ and single wffs A. $\Delta \models A$ means in every Kripke model in which all wffs of Δ hold at the actual world, A also holds in the actual world.
Our new notion of 'theory' is a diagram $(D, \mathbf{f}, d, U)$. This is a diagram of wffs which can hold in a Kripke model if it can be 'embedded' in it as in definition 3.1.1. The notion of conseqence can then be defined as in 3.1.1. The reader may ask, is a 'new theory' a set of such diagrams? The answer is that there is no need for such sets because any finite set of diagrams can be incorporated into one big diagram (in fact in traditional logic any finite set of formulas can be incorporated into one big conjunction formula).
What about proof theory? In classical logic and modal logic we have rules for manipulating formulas. In our case we have rules for manipulating diagrams. These are described in the next section, on algebraic *LDS*.

3.2 Case study 2: priority logic and PROLOG

Our second case study involving structure is propositional Horn clause computation without negation by failure. This case study is not semantically based. It is based on prioriized proof theory and is a typical challenge to translate to classical logic. Here *LDS* can help. If Δ is a set of Horn clauses and q an atom then the non-deterministic procedural interpretation of $\Delta?q$ should mean $\Delta \vdash_c q$, where $\vdash_c$ is provability in classical logic. Thus since the database $\{q, q \to q\}$ proves q, the computation of $?q$ from this database should succeed. In practice, i.e. in any PROLOG implementation, for this simple example, the database is represented as a list, i.e. either Δ_1

1. q
2. $q \to q$

or Δ_2

1. $q \to q$
2. q

and the computation is deterministic, searching the list either top down or bottom up, unifying with the head of a clause asking for the body. Thus if the implemenation scans the clauses top down, we have that $\Delta_1?q$ succeeds while $\Delta_2?q$ loops.

Can we characterise the notion of $\Delta?q$ = success for the top down interpretation?

We can enumerate the database Δ_2 but obviously a translation into classical logic like the one of the modal logic case is not easily available.

It is no use translating Δ_2 into $\{q(1) \to q(1), q(2)\}$ and trying to figure out a truth table for $\to$ much in the same way we did for $\Diamond$ in the previous case study. I don't think we can find some natural translation. Of course one can always translate into classical logic using meta predicates like **database** ('Δ'), **succeed** ('Δ', 'A') etc. but this is not a direct translation. To make our case study more tractable let us change slightly the way the computation works. This will no longer yield computation but another familiar logic. The reason for doing so is simply that at this stage I do not know how to handle the PROLOG case. There is, however, a Gentzen system for it by J. van Benthem [1992]. The change we make is that we ask the pointer to continue moving in one direction only. Thus $?q$ from

1. $q \to q$
2. q

succeeds, because after clause 1 the pointer continues to clause 2.

On the other hand $?q$ from

1. $p \to q$
2. r
3. $r \to p$

fails, because the pointer, having passed clause 2, cannot go back to it.

I am using the word 'pointer' without due explanation. Use your own intuitions. In more complex PROLOG programs execution is much more complex using stacks and several pointers. This database and query has only one goal done in sequence. So it is easy to explain the 'pointer.

Let Δ be a database in the form of a list of clauses $\Delta = (A_1, \ldots, A_n)$. The pointer is just a number $1 \leq k \leq n$. We query the pair (Δ, k) i.e. $(\Delta, k)?q$. To search for a solution we try and resolve with heads of the clauses $(A_k, A_{k+1}, \ldots)$. Once we resolve with clause j, we ask for the body of the clause. The computation must tell us for each k and j what is the new pointer. This we call the strategy of the pointer. For example if the pointer always goes back to $r = j - k$, we get that $((q \to q, q), 1)?q$ loops, while $((q \to q, q), 2)?q$ succeeds.

The above discussion presented an example where the database was a list. The list was needed for technical reasons, having to do with the implementation strategies of PROLOG interpreters. There is a rich family of systems which require priorities among the data. Among them are defeasible logics, cumulative defaults, Lambek Calculus, truth maintainance and belief revision. They all share the feature that they have no semantics but the data is organised according to priority or some network hierarchy and computation depends on this structure. The structure may come from the application area (defeasible logics, default) or be purely technical (maintaining consistency). This subsection deals with list structure. We would like our case study to be based on an application where the list structure in the database comes intuitively and naturally and has an obvious meaning. We find such a case in legal reasoning.

Consider the area of Esprit projects with which we are all familiar. There are rules for claiming expenses following an Esprit project meeting. These rules may vary slightly from country to country and from university to university. A typical rule could be of the form

$$C(x, d) \to S(x, d, 50)$$

which reads:

$$\begin{array}{c} x \text{ spent the day } d \text{ at a conference} \to \\ x \text{ gets subsistence of DM 50 for } d \end{array}$$

This is a time dependent rule in the sense that it is valid at any time s *after* it was introduced. Thus if the rule was introduced at time t, we can put it at the database as

$$t : C(x, d) \to S(x, d, 50)$$

If Dov has been to a conference at time s then the database will contain

$$s : C(\text{Dov}, d)$$

To ask the query $?S(\text{Dov}, d, 50)$ we resolve with the first clause and get $?C(\text{Dov}, d)$. However we can use the clause $s : C(\text{Dov}, d)$ only if $t < s$.

The above control mechanism is not *temporal control* (as was the case in the previous case study) but rather *prioritized control*,[16] and although the priorities come from temporal considerations, the time does not enter into the computation, only the control strategy of the 'PROLOG pointer'. In

[16] Again, we are not being precise. What is a 'prioritized' system? As a first approximation, say that a labelling system is a *priority system* if the atomic labels form a partially ordered set $(T, \leq)$ and the label generating function is concatenation. i.e. the labels are finite sequences of elements of T. The ordering $\leq$ on T is used to define some priority ordering on the sequences. This priority ordering is then used in all proof theoretical rules.

fact there are cases where exceptions are made by the commission allowing to claim for conferences before the start time of the rule. In terms of labelled deduction, the rule is that to perform modus ponens on $t : A \to B$ with $s : A$, we must have $t < s$, i.e. the 'fact' must come after the 'rule', where 'after' is not necessarily real 'after' but virtual 'after' including special permission.

Turning back to the application at hand, obviously we are supposed to claim expenses only once for each conference we attend. Thus if there is another rule, say university rule, introduced at time t' which says:

$$t' : C(x, y) \to UP(x, y, 150)$$

where $UP(x, y, 150)$ means university participation of DM 150, we are not expected to use both rules and get a total of DM 200. Either we claim from Esprit or we claim from the university.

The system may be flexible enough to allow us to claim DM 50 from Esprit and get the rest (DM 50) from the university, but not all systems are like that. The important point is that the assumption $s : C(x, d)$ can be used *only once* in the computation.

Let us summarise the properties we have so far:

- The database is structured with labels which are ordered.
- Modus ponens is allowed only when the ticket (i.e. the implication) is earlier than the minor premiss (i.e. the fact)
- Minor premisses (facts) can be used at most once.

Let us now consider another complication arising in the application area. The above rules are only valid for conferences held in European Community countries. For a conference in America, one needs permission from the project coordinator in Brussels. So we can write a more accurate rule:

$$t'' : AC(x, d) \wedge \ \mathrm{Per}(x, d) \to S(x, d, 100)$$

x participates at an American conference at day d and x has permission then x can get DM 100.

The problem with the above is that Brussels insists that permission be asked *before* conference participation, and not *after* the event. Thus to represent the rule we cannot use ordinary conjunction, but we need the 'first A then B' connective, which we denote by $A \otimes B$. Our rule becomes

$$t'' : \ \mathrm{Per}(x, d) \otimes C(x, d) \to S(x, d, 100)$$

Consider the query $?S(a, d, 100)$. We resolve with the above rule and get the query $?\ \mathrm{Per}(a, d) \otimes C(a, d)$. To succeed with that query we need to look at facts after t, succeed with $\mathrm{Per}(a, d)$ first and then succeed with $C(a, d)$ with label bigger than that of $\mathrm{Per}(a, d)$.

We thus get the following additional property for our system

- To show $A \otimes B$ we must succeed with A and with B, but must show A from an earlier part of the database than B.

So far, we made a distinction between 'facts' and 'rules'. In a full scale logic (not Horn clause) a rule can serve as a 'fact' for another rule. Take for example, the following:

t_2: If subsistence per day is only DM 50, we must appeal to the commission.

This has the form

t_2: $\forall x[C(x, d) \to S(x, d, 50)] \to q$

If $t < t_2$ then the rule can be used to derive q. At time r, how do we know whether the antecedent of t_2 holds? In our case it is explicitly stated as a rule, but in general it may be derivable from a group of rules. How do we check the implicational goal $r : \forall x[C(x, d) \to S(x, d, 50)]$? We have to use hypothetical reasoning. We add $C(x_0, d_0)$ to the database and try and derive $S(x_0, d_0, 50)$, with x_0 and d_0 a new Skolem constants. We thus get an additional principle.

There are two questions to be settled. First is when we add $C(x_0, d_0)$, i.e. with what priority (time) label? Intuitively the answer is at r, i.e. we add $r : C(x_0, d_0)$. This is not so simple. In

general we may be checking $r : A \to B$ and so we want to add $r : A$, but A may have the form $A_1 \otimes A_2$. Do we add $r : A_1 \otimes A_2$ or do we add $r_1 : A_1, r_2 : A_2$, with r_1, r_2 new priority points with the restriction $r < r_1 < r_2$. It makes sense to choose the latter, in which case, if $r < s$ do we also require $r_2 < s$. (i.e. do we put r_2 immediately after r but before anything which comes after r?)

The second question is how do we compute with the additional $r : A$? Do we use the part of the database after $r : A$? or do we say that since our purpose was to determine $r : A \to B$ then 'future data' is not relevant. The validity of $r_1 : A \to B$ now means that data up to now together with r must yield B?. These issues we leave for later.

Meanwhile note that rules can be used at any time after they are introduced, and as many times as necessary, while facts (in this particular Esprit example, though not necessarily in general) can be used only once. Thus the rule $A(x) \to B(x)$ can be used as often as needed, but the fact $A(d)$ can be used only once. This is a bit misleading because we can regard the rule as a family of rules for each instantiation of x. Thus we could write a family of rules.

$$\begin{array}{l} A(d) \to B(d) \\ A(e) \to B(e) \\ \vdots \end{array}$$

If $A(d)$ as a fact can be used only once and $A(d) \to B(d)$ can be fired only with $A(d)$ then the rule can be used only once anyway. We get

- Without a real loss of generality we can assume that propositional instances of rules can be used at most once

We are now ready for an example

Example 3.4 [Esprit Travel Expenses]
Database

$$\begin{array}{l} t_1 : \forall x, d[\ \mathrm{Per}(x, d) \otimes C(x, d) \to S(x, d, 200)] \\ s_1 :\ \mathrm{Per}(\mathrm{Dov},\ \mathrm{date}) \\ s_2 : C(\mathrm{Dov},\ \mathrm{date}) \\ t' : \forall x, d[C(x, d) \to\ \mathrm{UP}(x, d, 150)] \\ t_2 : \forall x, d[C(x, d) \to S(x, d, 50)] \to q \end{array}$$

Priority

$$t_1 < s_1 < s_2 < t' < t_2.$$

Computation Pointer
At any time the pointer resides at some priority label r, i.e. the basic computation structure is $(\Delta, r)?t : G$. The label r is the pointer. The compuation rules tell us which rules we can use to resolve with G. In our particular example we can resolve with any rule with label s. The computation then continues with the new pointer label $\max(r, s)$. Formally (we leave the pointer implicit):

- $\Delta?t : q$ succeeds if $s : q \in \Delta, t = s$.
- $\Delta?t : q$ succeeds if for some $s : A \to q \in \Delta, s < t$ and $\Delta?t : A$ succeeds.
- $\Delta?t : A \to q$ succeeds if $\Delta \cup \{t : A\}|?t : q$ succeeds.
- $\Delta?t : A \otimes B$ if for some $s_2 > s_1 > t, \Delta?s_1 : A$ and $\Delta?s_2 : B$ succeeds.

Let us ask the query $?s : q$, with $t_2 < s$. We can thus unify with rule t_2 and ask $?C(x_0, d_0) \to S(x_0, d_0, 50)$. We add to the database $s : C(x_0, d_0)$ and ask $?s : S(x_0, d_0, 50)$. This unifies with t_1 and we ask $?s :\ \mathrm{Per}(x_0, d_0) \otimes C(x_0, d_0)$. The last query fails.
If at some future time a general permission is given, then of course the computation will succeed.

It is possible to develop the model further. One can add integrity constraints, to make the database more realistic. Once we have integrity constraints we can deal with conditions $?t : A \to B$ as a goal is computed by letting $t : A$ into the database, which may violate integrity constraints, so some truth maintenance has to be done. We will not go into the details of all our options. A Martelli, L Giordano and I have a paper on this [Gabbay *et al.*,]. Our purpose here is just to illustrate a realistic use of labels which is purely syntactical. For this purpose our discussion so far is quite sufficient.

Let us give a simplified version of the computation so far in the form of propositional directional N-Prolog. The version is a stylised, simplification of the above legal database problem, but it also happens to be a fragment of the Lambek calculus, exactly suited for linguistic application and implemented by Esther Köning [1992].

Definition 3.5 [Directional N-Prolog] Let our language contain $\otimes$ and $\to$.

1. Clauses and Goals
   (a) A is a clause if A is an atom
   (b) A is a body if A is an atom
   (c) If A_i are clauses and q an atom then $(A_1 \otimes \ldots \otimes A_n) \to q$ is a clause with body $A_1 \otimes \ldots \otimes A_n$ and head q
   Note that $\otimes$ need not be commutative
   (d) A database is a list of clauses. We present a database Δ as
   $t_1 : A_1, \ldots, t_n : A_n$, $t_1 < t_2 < \ldots < t_n$. A_i are clauses. t_i are priorities and hence $<$ is strict. In case $t_1 = t_2$, we can simply write $t_1 : A_1 \wedge A_2$.
2. Let Δ be a database and B a goal. We recursively define the notion of $\Delta \vdash \alpha : B$ where α is a sequence of elements from $\{t_1, \ldots, t_n\}$.
   (a) $\Delta \vdash \alpha : B$ if B is atomic and $\alpha : B \in \Delta$
   (b) $\Delta \vdash \alpha : B_1 \otimes \ldots \otimes B_k \to q$ iff the database
   $$\Delta; x_1 : B_1, \ldots x_k : B_k \vdash \alpha * \beta : q$$
   where $*$ is concatenation and where $t_1 < \ldots t_n < x_1 < \ldots < x_k$ and β is a subsequence of $(x_1, \ldots, x_k)$.
   (c) $\Delta \vdash \alpha : B$ iff B is an atomic q and for some
   $$t_i : A_i \in \Delta, \text{ we have } A_i = C_1 \otimes \ldots \otimes C_k \to q$$
   and $\alpha = (t_i) * \alpha'$ and
   $$\{t_{i+1} : A_{i+1}, \ldots, t_n : A_n\} \vdash \alpha' : C_1 \otimes \ldots \otimes C_k$$
   (d) $\Delta \vdash \alpha : C_1 \otimes \ldots \otimes C_k$ iff Δ can be partitioned into k segments $\Delta = \Delta_1 * \ldots * \Delta_k$ and $\Delta_j \vdash \alpha_j : C_j$ and $\alpha = \alpha_1 * \ldots * \alpha_k$

Example 3.6

$$\begin{array}{l} t_1 : A \to B \\ t_2 : C \\ t_3 : A \\ t_1 < t_2 < t_3 \end{array}$$

proves $t_1 t_3 : B$
Note that we require the pointer to continue to go in the same direction. Thus

$$\begin{array}{l} s_1 : A \\ s_2 : A \to B \\ s_1 < s_2 \end{array}$$

does not prove B.
We need not use all clauses.

Example 3.7 [Translation into classical logic] It is possible to represent the above databases and reasoning structures in classical logic by introducing a new sort for the priority labels and describe the computation in classical logic. It might be rather unnatural. The translation is the following. We need predicate logic with two sorts of variables. The first sort, the $t_1, t_2, s_2, s_2, \ldots$, type of variables, range over an algebra $(A, *, <)$, where $*$ is concatenation, and $<$ is a irreflexive and transitive ordering. The other sort are $x, y, x_1, x_2, \ldots$, type of variables, which are the ordinary predicate calculus variables. Atomic predicates have the form $Q^*(t, x_1, \ldots, x_n)$ where $n \geq 1, t$ is the algebra sort variable and x_i are ordinary predicate sort variables. We now translate into this two sorted predicate logic, the Directional N-Prolog clauses as follows (τ is the translation):

- We associate with each atom $Q(x_1, \ldots, x_n)$ of Directional N-Prolog a predicate $Q^*(t, x_1, \ldots, x_n)$ of the two sorted predicate logic.
- $\tau(\alpha : Q(x_1, \ldots, x_n)) = Q^*(\alpha, x_1, \ldots, x_n)$ where α is a label from the algebra.
- $\tau(\alpha : A \otimes B) = \exists t_1 t_2(\alpha = t_1 * t_2 \wedge t_1 < t_2 \wedge \tau(t_1 : A) \wedge \tau(t_2 : B))$
- $\tau(\alpha : A \to q) = \forall t[\tau(t : A) \to \exists s\ (s \textit{ subsequence of } t \wedge \tau(\alpha * s : q))$.

The notion of subsequence must be definable in the two sorted classical logic.

The following can be proved

Lemma 3.8 1. If $t : A \in \Delta$ then $\Delta \vdash t : A$

2. If $\Delta \vdash t : A$ and $\Delta \subseteq \Delta'$ then $\Delta' \vdash t : A$
3. $$\frac{\begin{array}{c} s < t \\ \Delta \vdash t : A \\ \Delta' \vdash s : A \to B \end{array}}{\Delta' + \Delta \vdash s * t : B}$$

Lemma 3.9 Let **DH** be the Hilbert system with the following axioms and rules

$$\begin{array}{l} A \to A \\ (A \to B) \to ((C \to A) \to (C \to B)) \\ A \to (B \to A) \end{array}$$

MP $$\frac{\vdash A, \vdash A \to B}{\vdash B}$$

RT $$\frac{\vdash A \to B}{\vdash (B \to C) \to (A \to C)}$$

Then $t_1 : A_1, \ldots, t_n : A_n \vdash \alpha : B$ for some α, which is a subsequence of $(t_1, \ldots, t_n)$ iff

$$\mathbf{DH} \vdash A_1 \to \ldots \to (A_n \to B).$$

4 Algebraic LDS: a unifying solution:

The previous section presented two case studies, one strongly semantical and one strongly syntactical, where non-classical logic was used. We saw that classical logic can handle the case studies but the representation was not the most natural. In both cases it seems that a discipline for reasoning and propagating structures with labels is helpful. Let us develop such a discipline, which will be useful for many case studies and then check its relationship with classical logic. Thus our strategy with respect to the universality of classical logic is that we develop what seems to us to be the best system for the job (*LDS*) and then show that it can be translated to classical logic.

The reader may have his own opinions about the best logic for the case studies. Our argument for the universality of classical logic is not affected. *LDS* is complex enough that if it can be reduced to classical logic, the chances are that the reader's system can as well. The point is that some reasonably good system should be reduced in full technical details to classical logic.

This section will formally define the notions of algebraic *LDS*, its semantics, and labelled proof rules.

Definition 4.1 1. A labelling algebra is a first order theory in a language with function symbols $f_1, \ldots, f_k$ with respective arities and relation symbols $R_1, \ldots, R_m$ with respective arities, and with constants $t_1, t_2, t_3, \ldots$, and variables $x_1, x_2, \ldots$.

We call such a language an 'algebra' because its role is mainly algebraic in our conceptual framework.

2. A (finite) diagram D of the algebra is a set containing terms of the algebra and (finite) atomic expressions of the form $\pm R_i(t_1, \ldots, t_k)$ where t_i are terms.[17]

3. Let D be a diagram and ψ a closed formula of the language of $\mathcal{A}$. We say that ψ holds at D iff $D \vdash \psi$ in classical logic.

4. A logical language $\mathbf{L}$ is a predicate language with variables and predicates, a set of connectives $\sharp_1, \ldots, \sharp_n$ with appropriate arities and the quantifiers $\forall$ and $\exists$. The notion of a wff with free variables is defined in the traditional manner. The language may also contain function symbols $e_1, e_2, \ldots$ of various arities, and constants. The language shares the variables $x_1, x_2, \ldots$ with the labelling algebra. The constants of the langauge $\mathbf{L}$ are not atomic constants but are generated from special unary function symbols in the following manner. We assume we always have a sequence $c_1(x), c_2(x), \ldots$ of terms made of function symbols with one variable (for elements for the domains associated with labels). If t, s are labels $s : c_1(t)$ means that $c_1(t)$ is an element created (Skolemized) at label t but now residing at s. Thus the terms of $\mathbf{L}$ are generated by first generating the labels from the atomic labels and the labelling algebra function symbols $f_1, \ldots, f_k$. Then we apply $c_1, c_2, \ldots$ to generate the constants of $\mathbf{L}$ and then we apply the function symbols of $\mathbf{L}, e_1, e_2, \ldots$ to generate the full range of terms of $\mathbf{L}$. Such terms we denote by $\alpha_1, \alpha_2, \ldots$.

5. The language of an algebraic labelled deductive system is a pair $(\mathcal{A}, \mathbf{L})$ where $\mathcal{A}$ is an algebra and $\mathbf{L}$ is a logical language. The algebra and the language share the set of free variables and may share some of the constants and function symbols.

6. A declarative unit is a pair $t : A$, where t is a term of the algebra and A is a wff of the logic. t and A may share free variables and constants. A labelled term has the form $s : \alpha$, where s is a label and α is a term of $\mathbf{L}$.

7. A database is a tuple $\tau = (D, \mathbf{f}, d, U)$ where D is a diagram of the labelling algebra, and $\mathbf{f}$ and U are two functions, associating with each term in D a wff $A_t = \mathbf{f}(t)$ and a set of terms U_t. U_t is the set of $\mathbf{L}$ terms residing at label t. The functions $\mathbf{f}$ and U can also be displayed by writing $\{t : A, t : c_i(x)\}, t \in D$. Note that D may contain also some relations $\pm R(t_1, \ldots, t_n)$. $d \in D$ is the 'actual' world of D.

Example 4.2 A possible world *LDS* language can be as follows. The language is the first order language of modal logic with connectives $\Box$ and $\Diamond$. The labelling algebra is the first order language of a binary relation R. We have no function symbols in the algebra but we do have an infinite stock of constants. The modal language contains the functions $c_1(t), c_2(t), \ldots, t$ ranges over labels, generating the terms of the modal language. A diagram D has the form $\{t_1, \ldots, t_n, \pm R(s_i, s_j)\}$ where s_i, s_j are from among $t_1, \ldots, t_n$ which are terms in D. This can be graphically displayed as a proper geometric diagram such as where arrows display the relation R. Thus D in this case is

[17] Compare with Definition 3.1.1, where these concepts were given for modal logic. Since in modal logic the relation symbol R is used for the posisble world relation, we used the symbol ρ. We shall here abuse the notation and use R.

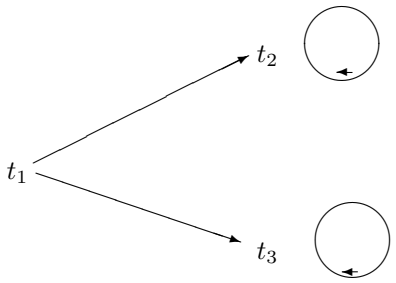


Figure 5:

$$\{t_1, t_2, t_3, R(t_1, t_2), R(t_1, t_3), R(t_3, t_3), R(t_2, t_2)\}.$$

A database is a tuple $(D, \mathbf{f}, d, U)$, where $d \in D$ and $\mathbf{f}$ associates with each term $t \in D$ a wff $\mathbf{f}(t) = A_t$ of predicate modal logic, and U_t is a set of terms of the modal language of the form $\{c_i(s_j)\}$, for $t \in D$. The following Figure 6 displays a database based on the diagram of Fig. 5.

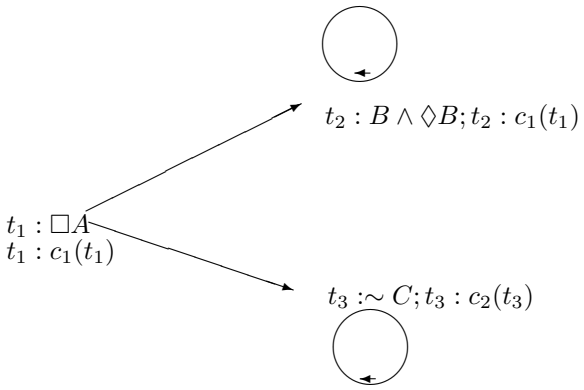


Figure 6:

Definition 4.3 Let X be a first order language. Let *Monadic*($\mathbf{X}$) be the language $\mathbf{X}$ enriched by an infinite number of new monadic predicate symbols. If ψ is a formula of the monadic language with exactly the new monadic predicates $Q_1, \ldots, Q_k$ and exactly the free variables $x_1, \ldots, x_m$, we write $\psi(x_1, \ldots, x_m, Q_1, \ldots, Q_k)$ to indicate this fact.

Definition 4.4 [Semantics for algebraic LDS] Let $(\mathcal{A}, \mathbf{L})$ be an *LDS* language. Let $\sharp_1, \ldots, \sharp_n$ be the connectives of the language, with arities $r(i), i = 1, \ldots, n$ respectively. A possible world semantical interpretation for $(\mathcal{A}, \mathbf{L})$ has the form $\mathbf{I} = (\tau, \psi_1, \ldots, \psi_n, \psi)$ where τ is a theory (axioms) in the language of $\mathcal{A}$ and each ψ_i is a wff of the monadic extension of the language of $\mathcal{A}$, containing $r(i)$ new monadic predicates and one free variable x, and ψ is a closed wff of the language with one monadic predicate and the constant a. Recall that $\mathcal{A}$ is based in first order logic.
A structure of the semantics has the form $(\mathbf{M}, a, V, h)$ where $\mathbf{M}$ is a classical model of the theory τ of the algebra $\mathcal{A}$, $V_m, m \in M$ is a nonempty domain and h is an assignment associating with each n-place atomic P of $\mathbf{L}$ and each $m \in M$ a subset $h(m, P) \subseteq V_m^n, a \in M$ and where M is the domain of $\mathbf{M}$.
Satisfaction $\vDash_h$ can be defined for arbitrary wffs of $\mathbf{L}$ via the inductive clauses and the usual 'abuse' of notation, $(V = \bigcup_{m \in M} V_m)$

0. $m \vDash P(b_1 \ldots, b_n)$ iff $(b_1, \ldots, b_n) \in h(m, P), b_i \in V$

1. $m \vDash \neg A$ iff $m \nvDash A$
2. $m \vDash A \wedge B$ iff $m \vDash A$ and $m \vDash B$
3. $m \vDash \sharp_i(A_1, \ldots, A_k)$ iff $\mathbf{M} \vDash \psi_i(m, Q_1, \ldots, Q_k)$, with $Q_i = \{n \mid n \vDash A_i\}$
4. $m \vDash \exists y A(y)$ iff for some $y \in V_m, m \vDash A(y)$.
5. We say that the structure $(\mathbf{M}, a, V, h)$ satisfies a formula A of $\mathbf{L}$ iff $\mathbf{M} \vDash \psi(Q_1, \ldots, Q_k)$, with $Q_i = \{m \mid m \vDash A\}$.
6. A database $\tau = (D, \mathbf{f}, d, U)$ is said to hold at a structure $(\mathbf{M}, a, V, h)$ iff there exist functions $g_1 : D \to M$ and $g_2 : U \to V$ such that g_1 validates the diagram D in $M, g_1(d) = a$, and for every $t : A(x_1, \ldots, x_n)$ in D we have $g_1(t) \vDash A(g_2(x_1), \ldots, g_2(x_n))$.
7. We say $\tau_1 \vDash \tau_2$ iff for every structure and every g_1^1, g_2^1 which validate τ_1 in the structure, there exists extensions g_1^2 and g_2^2 of g_1^1 and g_2^1 respectively, which validate τ_2.
8. We say the model validates τ iff $\mathrm{f}\{d : U_d\} \vDash \tau$, for $d \in \tau$.

We are now in a position to translate any algebraic *LDS* into many sorted classical logic. The definition is straightforward but has consequences for the question of universality of classical logic. An algebraic *LDS* has the form $(\mathcal{A}, \mathbf{L})$, where $\mathcal{A}$ is the algebra of labels and $\mathbf{L}$ a predicate language with non-classical connectives. To be specific, assume the labelling algebra is generated by one binary function symbol * and that the logic is generated by one binary connective $\Rightarrow$ and has no function symbols.

Thus the declarative units have the form $t : A$ where t is an expression built up from atomic labels $c, d, \ldots$ variables $x, y, z, \ldots$ and the operation *, and A is a formula built up from atomic formulas $Q(x), P(x), R(x, y)$ .... The terms and the formulas share the variables. Here is an example:

$$y * ((c * d) * z) : (Q(y) \Rightarrow P(x)) \Rightarrow R(x, y).$$

To translate the above into classical logic we associate with each atomic predicate $Q(x), P(x), R(x, y)$, an atomic formula $Q^*(t, x), P^*(t, x), R^*(t, x, y)$ of many sorted logic with one more sort t for terms.

An atomic declarative unit of the form say

$$(y * c) * d : R(x, y)$$

can be translated as

$$R^*((y * c) * d, x, y).$$

The first coordinate of R^* is for the label and it accepts terms from the labelling algebra $\mathcal{A}$. The rest of the coordinates are for the variables and constants of R from $\mathbf{L}$.

The problem arises when we want to translate a more complex formula such as

$$(y * c) * d : Q(y) \Rightarrow Q(x).$$

If we abbreviate $Q(y) \Rightarrow Q(x)$ as $E(x, y)$ then we would like to translate

$$(y * c) * d : E(x, y)$$

as

$$E^*((y * c) * d, x, y).$$

We cannot do that becasuse E is not atomic and we do not know what E^* is. We thus need a translation for '$\Rightarrow$' which will allow us to find $[A \Rightarrow B]^*$ from A^* and B^* by structural induction.

This is possible to do if we know what $\Rightarrow$ means. If $\Rightarrow$ has a semantical interpretation, then the semantics can help and if $\Rightarrow$ has a proof theoretic presentation then the proof rules might help. For example if we read $\Rightarrow$ as concatenation implication then we can translate

$$(\sharp) \quad [A \Rightarrow B]^*(t) = \text{ (definition) } \forall y(A^*(y)) \to B^*(t * y)$$

where * is the algebra function symbol, and $\rightarrow$ is classical implication.

According to ($\sharp$) the translation of

$$(y * c) * d : Q(y) \Rightarrow Q(x)$$

is

$$\forall z(Q^*(z, y)) \rightarrow Q^*(((y * c) * d) * z, x)$$

which is a formula of two sorted classical logic.

The translation of

$$y * ((c * d) * z) : (Q(y) \Rightarrow P(x)) \Rightarrow R(x, y)$$

is

$$\forall u[[Q(y) \Rightarrow P(x)]^*(u) \rightarrow R^*((y * (c * d) * z) * u, x, y)]$$

which is

$$\forall u[\forall w(Q^*(w, y) \rightarrow P^*(u * w, x)) \rightarrow R^*((y * (c * d) * z) * u, x, y)]$$

This should be compared with Example 3.2.4.

If we have a semantical interpretation for $\Rightarrow$, say as strict implication, then we can use that in our translation. If $<$ is the possible world relation we get

$$(\sharp\sharp) \quad [A \Rightarrow B]^*(t) = \text{definition } \forall y[t < y \wedge A^*(y) \rightarrow B^*(y)]$$

We can understand $<$ as

$$x < y \text{ iff } \exists z(x * z = y)$$

and assume * is associative.

In which case

$$[A \Rightarrow B]^*(t) = \forall y[A^*(t * y) \rightarrow B^*(t * y)]$$

and the translation of

$$y * c * d * z : (Q(y) \Rightarrow P(x)) \Rightarrow R(x, y)$$

is

$$\forall u[\forall w(Q^*(y * c * d * z * u * w, y) \rightarrow P^*(y * c * d * z * u * w, x)) \rightarrow R^*(y * c * d * z * u, x, y)]$$

In summary, we need to know how to translate the connectives. Also note that although we introduced labels to help us with structuring the data, when we translate, we lose the structure because we have to translate the connective.

This has a bearing on the debate. Take the case of modal logic. We compromised between the pure modal language and the pure translation into classical logic by allowing for labelled wffs $t : A$, where t is a world. When we translate $t : A$ as $A^*(t)$ and translate the modality:

$$[\Box B]^*(t) = \text{def } \forall s(tRs \rightarrow B^*(t))$$

we lose the structure of the labels and we are back to the option of expressing the modal phenomena inside classical logic. The difference is in use. We translate only when we need to interface modal formulas with non modal formulas or we want to perform heavy duty automated deduction!

We now give a general definition of semantic translation.

Definition 4.5 [Semantic translation of algebraic LDS into classical logic] Let $(\mathcal{A}, \mathbf{L})$ be an *LDS* as in definition 4.0.1. Let $\mathbf{I} = (\tau, \psi_1, \ldots, \psi_n, \psi)$ be a semantical interpretation for $(\mathcal{A}, \mathbf{L})$ as in definition 4.0.4. Note that ψ_i, ψ are wffs in the monadic extension of the language of $\mathcal{A}$. Thus from the point of view of two sorted predicate logic, the ψ_i and ψ are acceptable as formulas with pure sort variables and so are the $\mathcal{A}$ predicates appearing in ψ. In a sense the languages of $\mathcal{A}$ and $\mathbf{L}$ become *linked* by adding a sort in $\mathbf{L}$ for the terms of $\mathcal{A}$. This will be done systematically in Section 7.

We define a translation * into two sorted classical logic of $(\mathcal{A}, \mathbf{L})$, relative to $\mathbf{L}$ as follows:

With each atomic predicate $Q(x_1, \ldots, x_n)$ of $\mathbf{L}$ associate a two sorted classical predicate $Q^*(t, x_1, \ldots, x_n)$, where the first coordinate accepts the sort of terms from the algebra $\mathcal{A}$.

We now need the inductive definition of how to translate declarative units and databases.

1. The translation of $t : Q(x_1, \ldots, x_n)$, Q atomic is $Q^*(t, x_1, \ldots, x_n)$.
2. $[t : \neg A]^* = \neg[t : A]^*$
3. $[t : A \wedge B]^* = [t : A]^* \wedge [t : B]^*$
4. $[t : \exists y A(y)]^* = \exists y[t : A(y)]^*$, y not in t.
   $[t : \forall y A(y)]^* = \forall y[t : A(y)]$, y not in t.
5. Let $\sharp$ be any connective of $\mathbf{L}$ and ψ be its truth table:

$$[t : \sharp(A_1, \ldots, A_k)]^* = \psi(t, \lambda y[y : A_i]^*)$$

   where $\lambda y[y : A]^*$ is the predicate $[y : A]^*$ as a predicate of y, where y does not appear in A.

We now translate *LDS* theories into many sorted logic. A database can be displayed as a set of expressions of the form $\{\pm R(t_1, \ldots, t_n), t : c_i(x), t : A\}$ where R is a relation of the language of $\mathcal{A}$, t a label, x a variable or a term (label), c_i a term function(as in 4.0.1 item 1). We need a predicate $E(t_1, t_2)$ of the algebra sort in classical logic to translate $t : c_i(x)$ meaning the term $c_i(x)$ created at label x exists at label t. Thus we have:

$$[t : c_i(x)]^* = E(t, c_i(x))$$

A database τ is translated as $\bigwedge E(t, c_i(x)) \wedge \bigwedge t : [A]^* \wedge \bigwedge \pm R(t_1, \ldots, t_n)$, for $t : c_i(x), t : A$, and $\pm R(t_1, \ldots, t_n)$ in τ.

The above definitions of semantics for *LDS* and a semantical translation into classical logic cover the modal and temporal case study. The second case study, deals with a priority *LDS* which has only (syntactical) proof theoretic operational semantics (definition 3.2.2). We now need to define what kind of proof rules are available for *LDS* and how we can translate a proof theoretic *LDS* into classical logic.

The definition of semantics for *LDS* also defined logical consequence between two databases τ_1 and τ_2 (see Definition 4.0.4 and compare with Remark 3.1.3). We could define syntactical operations π on τ_1 which can transform it to τ_2 (notation $\tau_1 \mid\sim_\pi \tau_2$) and possibly prove a completeness theorem, for any τ_1, τ_2.

$$\tau_1 \models \tau_2 \text{ iff } \tau_1 \mid\sim_\pi \tau_2.$$

Presenting such a set of rules π would be considered as a proof system. The discipline of such rules is a proof theory for algebraic *LDS*. The basic declarative database is an algebraic constellation of labelled formulas. The rules allow us to manipulate one constellation into another constellation. The general form of such rules if given in definition 4.0.8 and discussed there. We shall see later that having a convenient proof theory for an *LDS*, allows us to translate it into second-order classical lgoic, without using any semantics. Thus for an arbitrary logic $\mathbf{L}$, if it can be formulated as an algebraic *LDS* system and provided with a proof theory, then it can be translated into classical logic.

We begin our presentation of the proof theory discipline with the case study of modal logic. This case can help clarify all concepts. Recall that a modal logic constellation might have the form in Fig. 7.

Figure 7:

The modal axioms and the meaning of $\Box$ dictate to us that in the constellation displayed in Fig. 7,, A must hold at s. Further the meaning of $\Diamond$ tells us that there should exist a point r with $s < r$ such that $r : B$.

We can thus state two rules for manipulating modal databases.

$$(*1) \qquad \frac{t : \Box A; t < s}{s : A}$$

and

$$(*2) \qquad \frac{s : \Diamond B}{\text{create } r, s < r \text{ and } r : B}$$

using the first rule we manipulate the constellation displayed in Fig. 7 into the one of Fig. 8 and using the second rule we further manipulate it into that displayed in Fig. 9 and Fig. 10.

Figure 8:

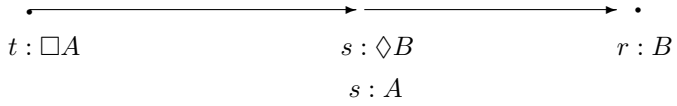


Figure 9:

The second rule is good for modal logics like **K, S4**, etc.

The axiom of Löb:

$$\Box(\Box A \to A) \to \Box A$$

corresponds to the modification rule

$$(*3) \qquad \frac{s : \Diamond B}{\text{create } r; s < r \text{ and } r : B \wedge \Box \sim B}$$

thus in the logic with the Löb axiom we get from the configuration of Fig. 7 to the configuration in Fig. 10.

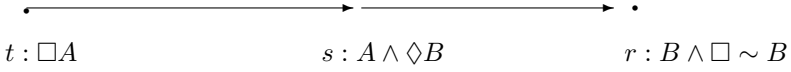


Figure 10:

It is clear now how the rules work. They allow us to move from one configuration to another and the consequence relation is between configurations. For example, we have Fig. 7 $\vDash$ Fig. 9, in modal **K** and with Löb's axiom we have Fig. 7 $\vDash$ Fig. 10.

The above rules are elimination rules. We still need introduction rules

$$(*4) \qquad \frac{s : A; t < s}{t : \Diamond A}$$

Table 1:

Axioms	*LDS* **Features**
K axioms $\Box(A \to B) \to (\Box A \to \Box B)$ $\vdash A \Rightarrow \vdash \Box A$ $\Diamond A =$ def$\neg\Box\neg A$	The notion of basic constellation or a diagram, as in Definition 4.0.1 (2.) Note that in the modal case the relation R in the diagram is binary. The *LDS* formulation contains also some simple rules for $\Box$ and $\Diamond$ some of which were shown in the figure above, rules (*1), (*2)
$\Box A \to \Box\Box A$	Transitivity of R in the constellation
$\Box(\Box A \to A) \to \Box A$	In modal semantics the axiom has no first order condition. It corresponds to the finiteness of the frame. In LDS it corresponds to the modification rule (*3).
$\Diamond A \wedge \Diamond B \to$ $\Diamond(A \wedge B) \vee \Diamond(A \wedge \Diamond B)$ $\vee\Diamond(B \wedge \Diamond A)$	Corresponds to the linearity of the relation R. This affects the basic rule (*2) as explained in Remark 4.0.6

$$(*5) \qquad \frac{\begin{array}{c}\text{create an arbitrary } s; t < s \\ \text{and show } s : A\end{array}}{t : \Box A}$$

Example for $\Box$ introduction:

Given $t : \Box(A \to B) \wedge \Box A$
Create $s, t > s$
Show $s : B$
Deduce $t : \Box B$

The picture however is not as simple as it seems. In the usual formulations of modal logics, axioms correspond to conditions on the possible world relation.

In our presentation, axioms correspond to any one of a variety of features. Table 1 below offers a selection.

We see here how a second order axiom, i.e. the axiom of Löb, which corresponds to a second order semantical condition, can become a simple movement in *LDS*. When *LDS* is translated into two sorted classical logic, the function symbols generating the labels may allow us to reduce the second order condition into first order, as is the case with McKinsey axiom $\Diamond\Box q \to \Box\Diamond q$, when added to **K** without transitivity.

Remark 4.6 Suppose we deal with the modal logic for linear frames. Then the configuration in Fig. 11. can be expanded in three ways.
By rule (*2) we can create a point $u : A$, with $t < u$. In a non linear modal logic such as **K, S4**, etc. this would lead us to the configuration of Fig. 12.
one more step would allow us to have $u : A \wedge B$ and hence by $\Diamond$ introduction we get $t : \Diamond(A \wedge B)$. However in the case of linear modal logic, Fig. 12. is not allowed. We need to consider five possibilities.

1. $t < u < r < s$
2. $t < u = r < s$

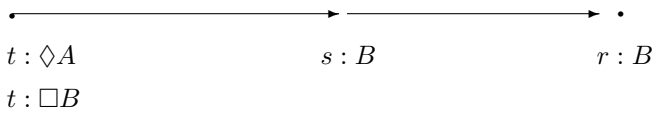


Figure 11:

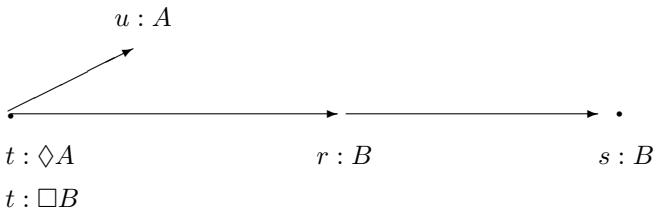


Figure 12:

3. $t < r < u < s$
4. $t < r < u = s$
5. $t < r < s < u$

If, as a result of *each* of these possibilities, we end up with $t : \Diamond(A \wedge B)$ then we can conclude $t : \Diamond(A \wedge B)$.
We have to do that because our databases are linear and the above five configurations are all the minimal possible extensions in which u can be accommodated.

We thus have to modify all the rules with 'create' in them to mean:

Given initial configuration

Split proof into n branches according to all minimal allowed extensions in which the created u can be accommodated.

(*3) becomes (**3)

$t : \Diamond A$ in a configuration

$(**3)$ create u, consider all allowed minimal extensions of D with u in them. Put $u : A$ in and branch the proof. The ultimate goal of the overall proof must succeed in all branches

The above is computationally very expensive. In the example previously given, we need to go to five configurations in order to make the simple move

$$(*6) \qquad \frac{t : \Box A \wedge \Diamond B}{t : \Diamond(A \wedge B)}$$

However our *LDS* proof discipline does not stop us from adopting (*6) as a rule. Recall that the *LDS* discipline tries to enjoy both worlds – the classical world through the labels and the special

non-classical world through the language of the formulas in the labels. For each appliction, desired balance can be sought.

We now come to quantifier rules. We have already assumed that different labels will have different sets of elements in them. To appreciate what this means, we take our clue from modal logic. Consider Fig. 13.

Figure 13:

At the label t, an x exists such that $t : \Diamond A(x, y)$ holds. This x depends on t and on y. We therefore need a Skolem function $c^t(y)$. The index t is read to mean that c^t was created at t.[18] We thus get $t : \Diamond A(c^t(y), y)$. Hence we can create a node $s : A(c^t(y), y)$. We also must indicate whether $c^t(y)$ 'exists' at the node s. If it does exist at s (probably because of some rules) then we write $s : c^t(y)$. The difference comes out in existential introduction. Suppose we have $s : E(c^t(y))$, can we infer $s : \exists x E(x)$? The answer depends whether $c^t(y)$ exists at s or not. Here are some rules

$$(*7) \qquad \frac{s : c; s : E(c)}{s : \exists x E(x)}$$

$$(*8) \qquad \frac{t : \exists x A(x, y_1, \ldots, y_n)}{t : A(c^t(y_1, \ldots, y_n), y_1, \ldots, y_n); t : c^t(y_1, \ldots, y_n)}$$

$$(*9) \qquad \frac{t : \forall x A(x)}{t : A(u^t); t : u^t,} \; u^t \text{ is a universal constant}$$

$$(*10) \qquad \frac{s : u^t; s : A(u^t); s : c^r}{s : A(c^r)}$$

u^t is a new universal constant, r is arbitrary

$$(*11) \qquad \frac{t : c^r; s : A(u^t)}{s : A(c^r)} \; u^t \text{ a universal constant}$$

$$(*12) \qquad \frac{s : u^t; s : A(u^t)}{s : \forall x A(x)} \; u^t \text{ a universal constant}$$

Rule (*9) is analogous to the classical logic rule which allows us to replace $\forall x A(x)$ by $A(u)$, where u is a universal constant, i.e. u is arbitrary. At any stage later in a classical logic proof, we can pass from $B(u)$ to $\forall u B(u)$ provided we discharged all additional assumptions. We can certainly pass from $B(u)$ to $B(c)$, c any constant. The same considerations apply to the labelled case except that we have to watch for the added complication that elements created in one label (world) (e.g. c^r, u^t) may not exist in another label (world). Imagine we have $t : \forall x A(x)$, this means A holds for all elements existing at t. We use rule (*9) and represent $t : \forall x A(x)$ by a universal constant u^t, i.e. we have now $t : u^t$ and $t : A(u^t)$. Suppose for some proof theoretical reason,

[18] To be consistent with the notation of Definition 3.1.1 and Definition 4.0.1, we should write $c(t)$. It is more convenient for the case of Skolem functions which involve other elements y, to push the t to be superscirpt. y itself may be a c^s. Thus $c^t(c^s)$ is clearer than $c(t, c(s))$, especially when embedded inside formulas.

$s : A(u^t)$ is obained. Thus we really have that A holds at s for an arbitrary element existing at t. Suppose now that we know that the element c^r created at r, exists at t. This is written as $t : c^r$. Then we can deduce $s : A(c^r)$. This is rule (*11).

We now explain rule (*10). Start with $t : \forall x A(x)$, this by rules (*9) and (*12) is equivalent to having $t : u^t$ and $t : A(u^t)$. Suppose that by some proof manipulation we end up with $s : u^t$ and $s : A(u^t)$. This means that the universal constant u^t is in label s and so is $s : A(u^t)$. We understand that as a proof of $s : \forall x A(x)$ from $t : \forall x A(x)$ and so we allow ourselves to deduce $s : \forall x A(x)$. Therefore for any c^r, which exists at s displayed as $s : c^r$, we get $A(c^r)$ at s, i.e. $s : A(c^r)$. This entire chain is summarised as rule (*10).

So far these rules assume that somehow an element c^t created at t ends up available at label s, i.e. $s : c^t$ holds. How do elements move around? We need special rules for that and they differ from system to system. In other words the logic must tell us how elements skolemised in one label can be transported to another label. These are called *visa rules*. Here are two sample rules corresponding to the Barcan and converse Barcan formulas:

$$(b1) \qquad \frac{t : x^r, t < s}{s : x^r} \; x \text{ either a constant } c \text{ or a universal constant } u$$

$$(b2) \qquad \frac{t : x^r, s < t}{s : x^r} \; x \text{ either constant } c \text{ or a universal constant } u$$

Example 4.7 [Barcan Formula] Use (b1) to show that

$$t : \forall x \Box A(x) \vdash t : \Box \forall x A(x)$$

1. Start $t : \forall x \Box A(x)$
2. $\forall$-Elimination at t yields $t : \Box A(u^t), t : u^t$
3. Create an arbitrary $s, t < s$.
4. $s : A(u^t), s : u^t$ by $\Box$ elimination rule and visa rule (b1).
5. $s : \forall x A(x)$, by (*12)
6. $t : \Box \forall x A(x)$, since s was arbitrary.

The discussion so far presented proof rules for the modal case study of Section 3. Here the labels clearly mean possible worlds and so the Skolem constants $c^t(x)$, had the natural meaning of elements existing at t, for example *friend of x at time t*. Our other case study was prioritised logic. It has no semantics. The proof theory and Skolemization rules and visas apply equally well to this case. It is interesting to see what meaning we can give to our Skolem and visa system when the labels are priorities.

Consider $t : \exists x A(x)$. We Skolemise and get $A(c^t)$ with $t : c^t$. If we allow c^t to exist in $s : c^t$, for s higher priority, we will get $s : \exists x A(x)$ by $\exists$ introduction and the priority of $\exists x A(x)$ will go up.

We thus have

- In prioritised logics, all elements are dependent on a priority (or labelled with priority). Elements of lower priority do not exist at higher priority.

So for example the statement

t: Every homeless person is to have a home in an Estate

Write it as

$$t : HL(x) \rightarrow \exists y[\text{Home}(x, y) \wedge \text{ Estate}(y)]$$

Skolemise and get $t : c^t(x)$, where c^t is the home of x. Its existence depends on the priority factor of the statement above.

Suppose we have

$$s : \text{Estate}(z) \rightarrow \exists u \ \text{Janitor}(z, u)$$

meaning:
with priorty s, every Estate home has a Janitor.

Skolemising we get $d^s(z)$. Thus by putting both arguments together (assuming x is homeless), we get $d^s(c^t(x))$. This means that we have people walking around not with passports, indicating as to where they were born (as is the case in modal logic) but with a priority label indicating to which priority they exist (or if we take a clue from modern society, how important they are).

We have finished our preliminary case study examples of how to introduce proof rules into *LDS*. We are now ready for the formal definition of proof theory for *LDS*.

Definition 4.8 [Proof Theory for Algebraic *LDS*] Let $(\mathcal{A}, \mathbf{L})$ be an algebraic LDS.

1. *Elimination Rules*
   An elimination rule for a connective $\sharp(A_1, \ldots, A_n)$ has the form

   $$\frac{\varphi; t_1 : B_1; \ldots, t_n : B_n; s : \sharp(A_1, \ldots, A_n)}{\psi; r_1 : C_1, \ldots; r_m : C_m}$$

   The terms $r_1, \ldots, r_m$ are new atomic constants.

   Where $\varphi(t_1, \ldots, t_n, s)$ is a formula of $\mathcal{A}$ called the pre-condition for the (firing of the) rule and $\psi(t_1, \ldots, t_n, s, r_1, \ldots, r_m)$ is a formula of A called the post condition. ψ may contain equality.

   The rule is to be understood as saying:

   > If we have proved the wffs above the line with labels satisfying φ then we can create new labels and deduce the formulas below the lines and the new and old labels satisfy ψ. ψ may contain equality just in case we want to say the new labels are equal to some old ones.

2. *Introduction Rules*
   Introduction rules in *LDS* are defined in terms of elimination rules and hence need not be introduced separately. Their use will be properly defined when we give the notion of a proof. Intuitively, when we are in the middle of a proof and we want to introduce a connective $t : \sharp(A_1, \ldots, A_n)$ with label t, we are really saying 'we already have this connective'. If this is indeed true, then for an aritrary elimination rule of this connective (e.g. like in (1) above), if we assume the antecedent, without the connective, we must be able to prove the consequent of the rule. If we can demonstrate this capability for each elimination rule, then we can introduce the connective. Take for example

   $$\frac{A; A \rightarrow B}{B}$$

   we show we already have $A \rightarrow B$ (i.e. introduce $A \rightarrow B$) by showing we can assume A and get B. Another example is $\frac{A \wedge B}{A}$ $\frac{A \wedge B}{B}$. To introduce $A \wedge B$ we must show we can get the conclusions of each rule, without the connective, namely Assume $\varnothing$ and get A and assume $\varnothing$ and get B.

3. *Quantifier Rules*
   These include the usual classical quantifier rules and visa rules

   $$(a) \quad \frac{t : \forall x A(x)}{t : A(u^t); t : u^t, u^t \text{ a new universal constant}}$$

   $$(b) \quad \frac{t : \forall x A(x); t : c^s}{t : A(c^s)}$$

$$(c)\quad \frac{t:c^s; t:A(c^s)}{t:\exists x A(x)}$$

$$(d)\quad \frac{t:\exists x A(x,y)}{t:A(c^t(y),y); t:c^t(y)}$$

$t:c^t(y)$ is optional in (d). It may not be adopted in modal logic but may be adopted in numerical or priority logics.

(e) To introduce $t:\forall x A(x)$ we prove $t:A(u_0^t)$, for any arbitrary new constant u_0^t. Some important side conditions may be involved.

4. *Visa Rules*
These have the form

$$\frac{t_i:A_i, t_i:c^{s_i}, r_j:B_j, \psi(t_i,s_i,r_j)}{r_j:c^{s_j'}}\ j=1\ldots k \text{ and } i=1,\ldots,n$$

where $s_j' \in \{s_i \mid i=1,\ldots n\}$, for $j=1,\ldots,k$..

The meaning of the visa rule is that if c^{s_i} exist at t_i where A_i holds and if ψ holds and B_j holds at r_j then $c^{s_j'}$ exist at r_j.

- Sample visa rule from modal logic:

$$\frac{t:c^s; t<r, r:\Box\bot}{r:c^s}$$

The rule says that if c^s exists at t then it exists at any endpoint above t. It happens to correspond to the following refinement of the Barcan formula.

$$\forall x \Box A(x) \to \Box(\Box\bot \to \forall x A(x)).$$

5. A proof π_0 of level 0 from a database $(D,\mathbf{f},d,U)$ is a sequence of labelled databases $(D_n,\mathbf{f}_n,d,U)$ and justification function $\mathbf{J}^0$ satisfying the following:

4.1 $(D_0,\mathbf{f}_o,d,U_0)=(D,\mathbf{f},d,U)$ and $\mathbf{J}^0(0)=$ 'assumption'.

4.2 $(D_{n+1},\mathbf{f}_{n+1},d,U_{n+1})$ is obtained from $(D_n,\mathbf{f}_n,d,U_n)$ by applying an Elimination rule and adding the consequent of the Elimination rule to $(D_n,\mathbf{f}_n,d,U_n)$ to obtain $(D_{n+1}$, $\mathbf{f}_{n+1}$, $d,U_{n+1})$. The elimination rule is applicable iff $D_n \vdash \varphi$ in which case $\mathbf{J}^0(n+1)=$ name of the elimination rule.

We write $(D,\mathbf{f},d,U) \vdash_0 (D',\mathbf{f}',d,U')$ iff there exists a proof π_0 of level 0 leading from one to the other.

6. A proof of level $\leq n$ has the form of linked sequences of databases with a main sequence π, and justification function $\mathbf{J}$. The first element of the main sequence π is $(D,\mathbf{f},d,U)$. Each element of the sequence is obtained from a previous one either according to one of the cases of a proof of level 0 or according to the following case

Introduction Case
For some connective $\sharp(A_1,\ldots,A_n)$ let

$$\frac{\varphi_j; t_1^j:B_1^j,\ldots,t_{n(j)}^j:B_{n(j)}^j; s_j:\sharp(A_1,\ldots,A_n)}{\psi_j; r_1^j:C_1^j,\ldots; r_{m(j)}^j:C_{m(j)}^j}$$

$j=1\ldots k$, be all the elimination rules involving $\sharp$. Suppose we also have that for some s, we have that for each j there is a proof π_j of level $\leq n-1$ of each $(D_n,\mathbf{f}_n,d,U_n)+\{\psi_j, r_i^j:C_i^j\}$

from $(D_n, \mathbf{f}_n, d, U_n) + \{\varphi_j, t_i^j : B_i^j\}$ (i.e. we can prove the consequent of each rule from the antecedent of the rule without the use of $s : \sharp(A_1, \ldots, A_n)$, as if we already have it) then the introduction step allows us to move onto $(D_{n+1}, \mathbf{f}_{n+1}, d, U_{n+1}) = (D_n, \mathbf{f}_n, d, U_n) + \{s : \sharp(A_1, \ldots, A_n)\}$, we link the proofs π_j into the proof π at line n, via the justification function, $\mathbf{J}(n+1) = \{\text{proofs } (\Pi_j, \mathbf{J}_j)\}$.

We write $(D, \mathbf{f}, d, U) \vdash_n (D', \mathbf{f}', d, U')$ if there is a proof of level $\leq n$ of the consequent from the antecedent. We write $(D, \mathbf{f}, d, U) \vdash (D', \mathbf{f}', d, U')$ if there is a proof of any level of the consequent from the antecedent.

7. For the sake of tractability we can also assume that in the elimination rule the complexity of $B_1, \ldots, B_n$ and $C_1, \ldots, C_m$ is strictly less than the complexity of $\sharp(A_1, \ldots, A_n)$. We should assume some suitable complexity function. So for example the $\Rightarrow$ elimination rule, modus ponens

$$\frac{t : A; s : A \Rightarrow B}{f(t,s) : B}$$

is acceptable because A, B are subformulas of $A \Rightarrow B$. However, the classical disjunction elimination rule, written in traditional form

$$\frac{\begin{array}{c} A \vee B \\ A \ldots C \\ B \ldots C \end{array}}{C}$$

where $X \ldots Y$ means there exists a (inductively simpler!) proof from X to Y, is not in acceptable form. We can rely, however, on the deduction theorem for $\Rightarrow$, when we do have it and write the rule as follows

$$\frac{t_1 : A \Rightarrow C; t_2 : B \Rightarrow C; s : A \vee B}{f(t_1, t_2, s) : C}$$

where C is a wff with less nested disjunctions than $A \vee B$.

In this case the complexity function is lexicographic on two parameters: the maximal number of nested disjunctions and the ordinary structural complexity of wffs. This is still unsatisfactory! On pure proof theoretical grounds we want C to be a subformula of A or B.

Later on in this section we are going to translate any proof theoretical *LDS* into classical logic. For that purpose we need assume a stronger control on complexity. Let us say at this point that we want any elimination rule to reduce some global complexity.

We are now in a position to define a proof theoretic translation of any *LDS* into classical logic. The definition is the same as in the semantical case except that we use the eliminaton rules for any connective $\sharp$ to reduce the translation of $t : \sharp(A_1, \ldots, A_n)$. Also compare with example 7.0.7 Section 7, which studies the general framework for this type of translation.

Definition 4.9 [Proof theoretical translation of Algebraic *LDS* into classical logic] Let $(\mathcal{A}, \mathbf{L})$ be a proof theoretic *LDS* as in definition 4.0.8. We define a translation * into two sorted predicate logic, relative to the elimination rules of the *LDS* as follows:
With each atomic $Q(x_1, \ldots, x_n)$ we associate a two sorted classical predicate $Q^*(t, x_1, \ldots, x_n)$. The first coordinate accepts terms of the sort of the algebra $\mathcal{A}$. The inductive definition of the translation $[t : A]^*$, for an arbitrary wff A and label t is as follows:

1. $[t : Q(x_1, \ldots, x_n)]^* = Q^*(t, x_1, \ldots, x_n)$, Q atomic.
2. $[t : \neg A]^* = \neg[t : A]^*$
3. $[t : A \wedge B]^* = [t : A]^* \wedge [t : B]^*$

4. $[t : \exists y A(y)]^* = \exists y[t : A(y)]^*$, y not in t.
   $[t : \forall y A(y)]^* = \forall y[t : A(y)]^*$, y not in t.

5. Let $\sharp$ be any connective of **L** whose elimination rules are

$$\frac{\varphi_j, t_1^j : B_1^j, \ldots, t_{n(j)}^j : B_{n(j)}^j; s_j : \sharp(A_1, \ldots, A_n)}{\psi_j, r_1^j : C_1^j, \ldots, r_{m(j)}^j : C_{m(j)}^j}$$

   Then

$$[t : \sharp(A_1, \ldots, A_n)]^* = \text{(definition)}$$

$$\bigwedge_j [\forall t_1^j, \ldots, t_{n(j)}^j \{\bigwedge_{i=1}^{n(j)} [t_i^j : B_i^j]^* \wedge \varphi_j \rightarrow$$
$$\exists r_1^j, \ldots, r_{m(j)}^j (\psi_j(s_j/t) \wedge \bigwedge_{i=1}^{m(j)} [r_i^j : C_i^j]^*)\}]$$

   where (s_j/t) means substitute t for s_j.

Note that we need to assume in the introduction rules that some complexity goes down. Unlike the semantic translations, B_i^j and C_i^j are not necessarily subformulas of $\sharp(A_1, \ldots, A_n)$ and so if some complexity goes down we end up with a finite number of steps with several formulas of the form $s : Q$, Q atomic.
The translation of a database is done as in definition 4.0.5.

Remark 4.10 [Concluding Remarks] We now summarise and evaluate the evolution steps in our thinking.
1. The need for Labelled Deductive Systems

- The first step was the realisation that practical applications which lend themselves to possible logical analysis, contain several independent and related structures. These structures must be recognised by any logic which we use to describe and reason about the application.

- The second step is that we have an option, either to use classical logic to represent these structures or to use specialised logics. Careful analysis of two case studies shows conceptually we might wish to use specialised logics, but computationally we are better off with classical logic.

- The third step is to look for a good framework which could give us the kind of logics we want for applications. The proposed framework was Labelled Deductive Systems, where the declarative units are of the form Labels: Formulas. The technical details are developed in a book on the subject [Gabbay, 1992d] . The important intuitive point is that we agree to manipulate the existing natural structures of the application together, side by side, without reduction or translation.

Our case studies showed that additional structure can come from semantics (bringing semantics into the logic, as in the case of temporal logic) or from resource or proof theoretic considerations (bringing priorities as labels). In either case the formalism (Algebraic *LDS*) is very similar. In fact, some successful wide spread formalisms such as the A-Box reasoning of KL-one is already a example for an *LDS* in our sense. Therefore we propose to use this formalism (*LDS*) in application areas where logic is needed. The practical usefulness and applicability of such a program can be demonstrated. We are currently trying to do exactly that. The reader can reflect upon his own personal experience to see how useful the labels can be. The relevance of *LDS* to the question of the universality of classical logic is that any *LDS*, either syntactically presented or semantically presented can be translated into classical logic.
The perceptive and critical reader may have already asked himself why do we need to present and use *LDS* as an intermediate step in the translation of a logic **L** into classical logic. Semantical translations are known, and given a semantics for the logic, we can translate the semantics directly into classical logic and thus obtain a translation. This can certainly be done for many modal and temporal systems. We need not formulate these systems as *LDS* and then translate the

LDS into classical logic. However, *LDS* is necessary for the translation of logics which have no known semantics, for example, any logic formulated proof theoretically through Elimination and Introduction rules. We need to take the following steps:
Step 1: Formulate the logic proof theoretically as an algebraic *LDS* of the kind described in this section (especially note that the introduction rules depend definitionally on the elimination rules, which may not be the case in the original logic).
Step 2: Translate the *LDS* formulation into classical logic.
The perceptive reader might continue to ask that once we manage to produce the *LDS* in Step 1 above, do we not now have a semantics? I.e. a term semantics arising from the rules? The answer is that it is not necessarily the case. In a proper semantics the semantical value of '$\sharp(A_1, \ldots, A_n)$' is reduced to the values of 'A_i'. We allow Elimination rules which reduce a formula to other formulas, not necessarily subformulas of it, as long as some general measure of complexity is reduced and a very long path involving all connectives may be involved in the reduction. It is not at all clear that any semantics can be extracted out of this reduction. More on these points will be given in Section 10.

2. Advantages of LDS

The advantage of *LDS* is that it is a natural and adaptable way of doing logic. We consider a logical system as basically a discipline for databases and labels, giving structures and mechanisms for deduction. The mechanisms could be either proof theory or other mechanisms, such as abduction or circumscription, or explanation, etc. This scenario is very natural and can be adapted to a variety of application areas. To use *LDS* in an application area we need to

1. Recognise underlying structures in the application. These include
   (a) the declarative information to be manipulated and reasoned with.
   (b) The units (objects) to be manipulated and their relative structure. (e.g. semantic objects such as worlds or individuals, syntactic objects such as priorities or probabilities).
   (c) Recognise any compatibility or conflict in natural manipulative movement between (a) and (b).
2. Having recognised in (1) the natural components of the system, we devise an *LDS* to represent and reason about them. Note that *LDS* is not a single logic, but a family of logics.

The above process needs to be applied anyway for any use of traditional logic. What I am saying to the reader is that he should *not* approach the problem with a pre-determined pre-fabricated logic (such as the logic of his youth) and therefore be compelled to force all representation into it. In fact, what many researchers often do in parctice is to use labels as side effects (implementation tricks) to *retain* distinctions that their (favourite) logic forces them to abandon.[19]
Once we adopt the discipline of *LDS*, we see that there are other advantages.

- The labels can be formally used to bring into the object level metalevel features of the logic. A simple example would be to use the labels to trace the proof of the current goal. Conditions on the label can restrict the next move. These conditions are basically metalevel, but such metalevel features can be incorporated into an object level algebraic labelled proof rule.
- *LDS* allows for a uniform method for bringing the semantics into the syntax. See my paper [Gabbay, 1992d] as a striking example of that, with a connection with situation theory.
- *LDS* is a unified framework for monotonic and non-monotonic logics.

[19] My favourite analogy is that of a person who has a wife (husband) and a mistress (lover). Obviously they are there for a reason and may have considerable influence (good or bad)! There may be a need for them much in the same way that there is a need for the label. Rather than supress them and treat them as a side effect, I am proposing that we bring them forward into the open, and recognise their influence. We should openly declare that a declarative unit is a formula and a label (or several labels) and analogously in some societies it may be better to admit (the unfortunate fact) that a 'family' unit is a husband/wife and a mistress/lover (read Balzac!). We should recognise the structure involved so that we can handle it better.

- *LDS* is more computationally transparent (through the labels) and hence more receptive to meaningful optimisations.
- The proof theory of *LDS* can be more flexible. In classical logic, there is proof theory and there are tableaux systems which can be considered as model building systems. One is syntax based and the other is semantically based. In *LDS*, a tableaux procedure is just another *LDS*. The basic notion of consequence in *LDS* is that of a database proving a labelled formula for example $t_1 : A_1, t_2 : A_2 \ldots \vdash s : B$. A tableaux refutation for an *LDS* consequence relation will start with

  $$\text{True } [t_i : A_i], \text{ False } [s : B]$$

  and manipulate that. But such a system is just another *LDS*!

  Thus we have: $\Delta_1 \vdash_{LDS_1} \alpha : A$ iff in the tableaux system $\Delta_2 \vdash_{LDS_2}$ '*closed*' where LDS_2 is an *LDS* system dealing with *signed* labelled formulas.
- The devices of labelled Skolemization and visas across worlds are very powerful tools of *LDS*.
- Different worlds can have different reasoning systems (time dependent reasoning).
- We can reduce higher order properties to lower order.

3. Limitations of LDS

The main limitation is that an *LDS* logic needs to commit itself. Take for example **S4**. This logic has one modality which is reflexive and transitive. Presented as a Hilbert system, we write a certain number of axioms:

$$\Box A \to A$$
$$\Box A \to \Box\Box A$$
$$\Box(A \wedge B) \leftrightarrow (\Box A \wedge \Box B)$$
$$\vdash A \Rightarrow \vdash \Box A$$

The above system is not committed to any interpretation. $\Box A$ means any of the following:

- A holds in all accessible worlds.
- A is a set in the Euclidean plane and $\Box A$ is its topological interior.
- A is provable
- $\Box$ represents the progressive of English (e.g. if A is 'John walks', then $\Box A$ is 'John is walking').
- $\Box$ can mean some algebraic operation.

In any *LDS* interpretation, we need to identify the labels, in order to state what the declarative units are. The nature of t, the algebra of t, will commit us to some - if not exactly one - of the possible interpretations. We thus loose generality. We gain power, but we have to sacrifice our semantic options. When we are applying *LDS*, we may not mind that because the application area already dictates the interpretation. Thus we must be careful to choose the right level of labelling. Not too detailed to be able to remain within the realm of logic, avoiding turning the system into an implementation.

This argument can be used against the universality of classical logic. To translate a specific logic into classical logic, such as the **S4** necessity $\Box$, we need to commit its interpretation. Some application areas, such as legal reasoning, may not allow us to commit the interpretation. Furthermore, such an application area may not be particularly interested in the computational advantages offered by the translation and may be more sensitive to the naturalness of the representation.

By the way, this limitation is equally valid when we translate into or use directly classical logic. $\Box$ has to be translated into classical logic and the translation requires commitment, (unless we use the *term translation*).

4. Conclusion, Relation to The Debate

We conclude with further comparison with classical logics.

- Any *LDS* system can be translated into classical logic, in a technical sense. Take two sorted classical logic with one sort for the algebra of labels and turn $t : A$ into $A^*(t)$. This is the same process as the translation of modal logic. Thus the support for the universality of classical logic comes from the generality and unifying role of *LDS* and the fact that we need not have a semantical interpretation for the translation. The argument also inherits its limitation. To translate into classical logic, we need to commit the interpretation, even in the proof theoretic case, because the commitment is in the choice of the algebra of labels. Of course there is always the *term translation*, which does not commit to anything, but this translation is not conceptual and is good mainly for automated deduction purposes. More on this in Section 10.

- It is an interesting mental exercise to realise that classical logic itself can be turned into an *LDS* in a non trivial way. Take any predicate of classical logic, say $Q(x, y, z)$ and imagine a classical theory τ for Q. Examining the classical models of τ, we might find that most of the models of τ may have some distinguished elements in them in terms of which Q can be described. It may be worth our while perception-wise to write $x : Q^*(y, z)$ instead of $Q(x, y, z)$ and bring out *semantically* this special structure. This process is inverse to the translation method of (1) and intends to stress the semantical properties of the x coordinates. The x-labels will be manipulated separately, according to the model theory, and may result in a proof theory for τ which is much clearer. This would be an *LDS* formulation of τ.

  This construction shows how to bring (even in classical logic) the semantics into the syntax. Use the elements of the model as labels, or as a special sort. I think we can dispense with model theoretic semantics altogether. The correct notion of *semantics* in my view is *sound translation*. Given two system $\mathbf{L}_1$ and $\mathbf{L}_2$, $\mathbf{L}_2$ can serve as *semantics* for $\mathbf{L}_1$ if $\mathbf{L}_1$ can be soundly translated into $\mathbf{L}_2$. What corresponds to a completeness theorem is the faithfulness (also called completeness) of the translation. There are such semantics in the literature. Famous among them are the Gödel Dialectica interpretation, Boolos translation of modal G into Peano arithmetic and the multitude of 'operational semantics' in theoretical computer science.

5 Reductions to classical logic: the options

There exist in the literature several translations of modal and temporal logic into the classical predicate calculus. The previous two sections applied these methods to several case studies and discussed the reductions both conceptually and mathematically. These translations can be used not only as mathematical and conceptual reductions but also as computational reductions into the Horn clause fragment of classical logic.[20] Through these reductions, classical logic was playing the role of a metalanguage. Although this role is intuitively clear, there is a lack of proper analysis of the language - metalanguage capabilities of classical logic and there is a need in general for a framework for clarifying the notion of what we might call *object language implementation of meta language concepts*.

The purpose of this section is to fill this gap, and clarify the fundamental concepts involved in the relationship between metalanguage and object language features. We will also study in

[20] If a logic $\mathbf{L}$ can be translated into classical logic, then we can identify the sublanguage $\mathbf{L}_{\text{Horn}}$ of all formulas of $\mathbf{L}$ whose target translation is a Horn clause. This fragment can be expected to be the 'computable part' of $\mathbf{L}$ and perhaps algorithmic procedures can be developed directly for this fragment. Consdier for example modal logic. Consider the modal labelled formula $t : Q(x) \wedge \Diamond R(x, y) \rightarrow P(y)$. When translated into classical logic semantically it becomes

$$Q^*(t, x) \wedge \exists s(t < s \wedge R^*(s, x, y)) \rightarrow P^*(t, y)$$

which is equivalent to

$$Q^*(t, x) \wedge t < s \wedge R^*(s, x, y) \rightarrow P^*(t.y)$$

The above is a classical many sorted Horn clause.

Thus we know that we can apply $\Diamond$ into 'bodies' of clauses and remain in the 'Horn fragment'. Similarly $\Diamond A \rightarrow \Box B$ is in the 'Horn fragment' if B is a 'Horn clause' and A is a 'Horn body'. This approach is systematically investigated in [Gabbay, 1987] and [Gabbay, 1990].

more detail a metalanguage **HFP**, presented in a later section, and examine the claim that it can function as a general purpose metalanguage. Our conceptual framework will revolve around two pure notions. The notions of a *proper metalanguage* and that of an object language $\mathbf{L}_1$ *implementing metalanguage features* for a language $\mathbf{L}_2$.

Given a language $\mathbf{L}$, we understand by a proper metalanguage $\mathbf{M}$ to the language $\mathbf{L}$, any language which is able to name every well formed symbol of $\mathbf{L}$ and to describe all the logical operations of $\mathbf{L}$. Thus formulas φ of $\mathbf{L}$ become terms t_φ of $\mathbf{M}$ and logical relations of $\mathbf{L}$ become predicates in $\mathbf{M}$. $\mathbf{M}$ must have axioms to ensure that every interpretation of $\mathbf{M}$ gives rise, through the embedding of $\mathbf{L}$ in $\mathbf{M}$, to an interpretation of $\mathbf{L}$. For a given $\mathbf{L}$, $\mathbf{M}$ is not unique. Many languages can serve as a metalanguage for $\mathbf{L}$, provided they are strong enough. In computer language terms, any strong enough language, eg BASIC, can be used to write an interpreter for any other language. In such a case one *simulates* one language in another. The other pure notion of one language, $\mathbf{L}_1$, serving as an object language, $\mathbf{L}_2$, is more difficult to explain. Consider the relationship between $\mathbf{L}$ and $\mathbf{M}$. $\mathbf{L}$ is a sublanguage of $\mathbf{M}$ in a very strong sense. $\mathbf{M}$ has names for everything in $\mathbf{L}$ and axioms about $\mathbf{L}$. Imagine a more equal relationship between two languages. Suppose $\mathbf{L}_1$ and $\mathbf{L}_2$ are both sublanguages of a language $\mathbf{L}_{1,2}$. $\mathbf{L}_{1,2}$ is not a metalanguage to $\mathbf{L}_1$. It is just a richer language. For example, classical predicate logic, $\mathbf{L}_1$, augmented with additional dummy predicates being the language $\mathbf{L}_2$ gives us $\mathbf{L}_{1,2}$ which is certainly not a metalanguage. Consider $\mathbf{L}_{1,2}$ and consider some mixed axioms $\Delta_{1,2}$ in $\mathbf{L}_{1,2}$ affecting both $\mathbf{L}_1$ and $\mathbf{L}_2$. Any interpretation of $\mathbf{L}_{1,2}$ satisfying the mixed axioms, will induce an interpretation on $\mathbf{L}_1$ alone. Similarly, we can consider a proper metalanguage $\mathbf{M}_1$ of $\mathbf{L}_1$, with names for $\mathbf{L}_1$ symbols, etc. as discussed before. In the language $\mathbf{M}_1$ we can write axioms $\Delta_{\mathbf{M}_1}$ about $\mathbf{L}_1$, restricting its possible interpretations. It may be the case that the family of possible pure interpretations of $\mathbf{L}_1$ allowed by $\Delta_{1,2}$ (as induced from interpretations of $\mathbf{L}_{1,2}$ satisfying $\Delta_{1,2}$) are the same as the possible interpretations of $\mathbf{L}_1$ allowed by $\Delta_{\mathbf{M}_1}$ (as induced from interpretations of $\mathbf{M}$ satisfying $\Delta_{\mathbf{M}_1}$). In this case we say that $\mathbf{L}_{1,2}$ and $\Delta_{1,2}$ is an object language inplementation of $\Delta_{\mathbf{M}_1}$. The justification for 'object language' is that $\mathbf{L}_{1,2}$ is not meta to $\mathbf{L}_1$, it is just an extension. Consider for example two interpreters $\mathbf{L}_1$ and $\mathbf{L}_2$ which are linked in some way. Thus each interpteter has its allowable options of how to run. However, since they are linked, not all of these individual options can be realised. $\mathbf{L}_2$ can choose to run in certain ways which will *limit* the options of $\mathbf{L}_1$ because they are linked. Suppose we use a proper metalanguage $\mathbf{M}_1$ to talk and describe the options of $\mathbf{L}_1$ and suppose further that we express, through a wff φ_1 of $\mathbf{M}_1$ our wish to have $\mathbf{L}_1$ run only in certain ways. So φ_1 (if true) limits the runs (or options) of $\mathbf{L}_1$. There is another way of limiting the runs of $\mathbf{L}_1$, through its linkage with $\mathbf{L}_2$.

It may be that $\mathbf{L}_2$ can be restricted by condition δ_1 so that $\mathbf{L}_1$ (when linked with $\mathbf{L}_2$ satisfying δ_1) can run exactly in a way which satisfies φ_1. If this is the case, then we say that $\mathbf{L}_2$, through its object language restriction δ_1 and its linkage with $\mathbf{L}_1$, can *implement* φ_1, ie implement the metastatement φ_1. It may be that for each φ_n of $\mathbf{M}_1$, there exists a way of restricting $\mathbf{L}_2$ by δ_n which through the link with $\mathbf{L}_1$, allows $\mathbf{L}_1$ to run only in a way which satisfies φ_n. In that case we have object language implementation of the set $\{\varphi_n\}$ of the language $\mathbf{M}_1$.

To make these concepts more concrete, think of an *LDS* system. The logic $\mathbf{L}$ of the formulas is $\mathbf{L}_2$ and the algebra $\mathcal{A}$ of labels is $\mathbf{L}_1$. When linked together they form an *LDS* system. The metalanguage $\mathbf{M}$ is a language capable of talking about the proof theory of the logic $\mathbf{L}$. Here is a concrete example: let $\mathbf{L}$ be intuitionistic implicational logic. Let the proof theory for $\mathbf{L}$ be given using modus ponens ($A, A \to B \vdash B$) and $\to$ Introduction (to show $A \to B$, assume A and show B). $\mathbf{M}$ will be a metalevel language capable of talking about proofs of $\mathbf{L}$, containing predicates which can describe how many times an assumption A was used in a given proof of B. Let φ be a formula of $\mathbf{M}$ saying that each assumption is used at most once. Some proofs satisfy φ, some do not. We can add a labelling algebra $(A, \cup)$ where A is a set of atomic labels and $\cup$ is a multiset union of atomic labels and use the labels to keep trace exactly which assumptions are used in the proof. The system becomes:

- Label all assumptions by different atomic labels

- Formulate modus ponens as

$$\frac{\alpha : A; \beta : A \to B \text{ and } \alpha \cap \beta = \varnothing}{\alpha \cup \beta : B}$$

- Formulate $\to$ introduction as the following:
  to show $\alpha : A \to B$ assume $x : A$ with a new atomic label x and show $\alpha \cup \{x\} : B$.

- Say $\Delta \vdash B$ if we label all wffs of Δ with different atomic labels we can prove $\alpha : B$ with α a subset of the set of these labels.

This *LDS* system implements the metalevel condition φ on the proofs.

Another example is from resolution in classical logic. Suppose we have a classical language $\mathbf{L}_1$ and a set Δ of clauses which is inconsistent. The resolution machinery may derive the empty clause but may not be equipped to give a set of substitutions into the clauses of Δ which can yield propositional inconsistency. Let $\mathbf{M}_1$ be a metalanguage in which we demand this set. This demand can be implemented by adding a dummy predicate $Q(x_1, x_2, \ldots)$ with the appropriate variables (language $\mathbf{L}_2$) as a disjunct to all clauses of Δ. The resolution machine can stop when unable to eliminate Q and from the various instantiations of Q the substitutions can be obtained.

The reader should note that the concept of $\mathbf{L}_2$ acting as an object level implementation of some metalevel statements $\{\varphi_n\}$ of $\mathbf{L}_1$ is a relative one. Because $\mathbf{L}_1$ an $\mathbf{L}_2$ are linked, we can equally say that $\mathbf{L}_1$ is an object level implementation of some metalevel statements $\{\varphi'_n\}$ of $\mathbf{L}_2$ in some language $\mathbf{M}_2$. Thus the concept is relative to a metalevel set of sentences Δ.

Of course the above pure concepts have to be formally defined. This may not be easy. We prefer at first, to illustrate the two pure concepts for one specific example, namely a temporal logic $\mathbf{L}$, and examine in depth the possibilities of using a metalanguage $\mathbf{M}$ for talking about time and about the formulas of $\mathbf{L}$ which hold at each moment of time. Such a metalanguage must be able to name the moments of time and the earlier later relation $<$, and we also need names and proof predicates for formulas of the temporal logic $\mathbf{L}$.

Let 't' name the moment of time t and '$<$' name $<$. Let 'φ' be the name of φ. Let **Hold**('$<$', 't', 's') be the metapredicate saying $t < s$. Let **Data**('t', 'φ') be the metapredicate saying φ holds at t. Let **Prove**('t', 'Ψ_1', 'ψ_2') be the predicate indicating that $\Psi_1 = \bigwedge_{s,\varphi}$ **Data** ('s', 'φ') $\vdash$ **Data** ('t', 'ψ_2'). Note that Ψ_1 is a conjunction of sentences of the form **Data**('s', 'φ') and therefore **Prove** is meta to **Data**.

The following is a formal metalanguage system containing the above predicates.

Definition 5.1 The metalanguage M

1. Let $\mathbf{M}$ be a predicate meta-language with predicates **Hold**(x, y, z), **Data**(x, y) and **Prove**(x, y, z). The language is sorted. The sorts are as follows:
   (a) x in **Hold** is a relation name ($<$).
   (b) y, z in **Hold** and x in **Prove** and x in **Data** are time points names.
   (c) y in **Data** is an object language relation name.
   (d) y and z in **Prove** are **Data** relation names. y is 'higher' than z but in $\mathbf{M}$ they are all just terms for data. Thus **Prove** is a metapredicate to **Data**.

2. The language $\mathbf{M}$ also has function symbols $\mathbf{f}_{\mathbf{Prove}}$, $\mathbf{f}_{\mathbf{Hold}}$, $\mathbf{f}_{\mathbf{Data}}$, $\mathbf{f}_\wedge, \mathbf{f}_\vee, \mathbf{f}_\sim, \mathbf{f}_\to, \mathbf{f}_F$ and $\mathbf{f}_P$ satisfying the following:
   $\mathbf{f}_\wedge$ ('φ', 'ψ') = '$\varphi \wedge \psi$'
   $\mathbf{f}_\vee$ ('φ', 'ψ') = '$\varphi \vee \psi$'
   $\mathbf{f}_\to$ ('φ', 'ψ') = '$\varphi \to \psi$'
   $\mathbf{f}_\sim$ ('φ') = '$\sim \varphi$'
   $\mathbf{f}_F$ ('φ') = '$F\varphi$'
   $\mathbf{f}_P$ ('φ') = '$P\varphi$'

$\mathbf{f_{Data}}$ ('t', 'A') = '**Data** ('t', 'A')'

$\mathbf{f_{Hold}}$ ('$<$', 't', 's') = '$t < s$'
$\mathbf{f_{Prove}}$ ('t', 'Ψ', 'φ') = '**Prove** ('t', 'Ψ', 'φ')'
The following axioms hold in **M**.

(a) **Prove**('t', 'φ', '$A \wedge B$') $\leftrightarrow$ (**Prove**('t', 'φ', 'A') $\wedge$ **Prove**('t', 'φ', 'B'))

(b) **Prove** ('t', 'φ', '$A \vee B$') $\leftrightarrow$ **Prove**('t', 'φ', '$(A \to \bot) \to B$')

(c) **Prove** ('t', 'φ', '$A \to B$') $\leftrightarrow$ **Prove**('t', $\mathbf{f}_\wedge$ ('φ', $\mathbf{f_{Data}}$('t', 'A')), 'B')

(d) **Prove** ('t', 'φ', '$\bot$') $\to \forall x$ **Prove** ('t', 'φ', x)

(e) **Prove** ('t', 'φ', 'FA') $\leftrightarrow$
$\exists x$**Hold** ('$<$', 't', x) $\wedge$ **Prove** (x, 'φ', 'A')

(f) **Prove** ('t', '**Data**' ('t', 'A'), 'A')

(g) **Prove** $(x, y, z) \to \forall y'$ **Prove** $(x, \mathbf{f}_\wedge(y, y'), z)$

(h) **Prove** $(x, y, z) \wedge$ **Prove** $(x, y, \mathbf{f}_\to(z, z')) \to$ **Prove** (x, y, z')

(i) **Prove** $(x, y, \mathbf{f}_\sim(z)) \leftrightarrow$ **Prove** $(x, y, \mathbf{f}_\to(z, \text{'}\bot\text{'}))$

(j) **Prove**$(x, \mathbf{f}_\sim(\mathbf{f}_\sim(y)), y)$.

Example 5.2 Consider the following system Fig. 14. At time 1 A holds, at time 2 $P(A \wedge FB) \to$

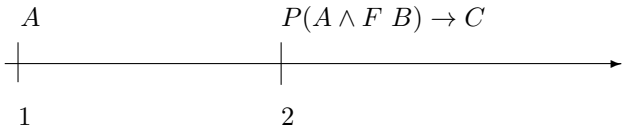


Figure 14:

C holds. We query at time 1 the formula $Q = FB \to FC$. We will argue informally. The actual proof is the formalization, in the metalanguage, of our informal proof.

1. Assume FB at time 1 and below we show FC at time 1.
2. If FB holds at time 1 then $P(A \wedge FB)$ holds at time 2 and hence C holds at time 2.
3. If C holds at time 2 then FC holds at time 1.
4. From (1) - (3) it follows that $FB \to FC$ holds at time 1.

Note that the above reasoning is proof theoretic and not model theoretic. In other words we assume we know that certain formulas hold at a certain configuration of points and we reason using the proof theoretic properties of the connectives that more formulas hold at some other points. This is done in **L**. The metalanguage **M** can represent or formalise this reasoning.

The next question to ask ourselves is, can classical logic itself, which is used as the underlying logic of **M**, be used as the metalanguage **M** itself. This means that we do not use naming functions but represent the system, directly in classical logic. Actually one needs exact definitions of the concept 'using classical logic directly', which is not easy to give.

Our task in this section is to examine to what extent predicate logic itself, without any additional connectives (for time) can handle adequately all temporal phenomena within itself. This we want to accomplish through explicit reference in predicate logic to time points and intervals and events by special time variables and predicates.

We must analyse the temporal expressive power of first order predicate logic, and indicate how it can be used effectively and correctly.

This problem has two aspects. First is the mathematical aspect, namely, are there temporal logics defined using new special connectives and axioms, which cannot mathematically be presented in first order predicate logic? Are there such logics which are intuitive and arise in applications? One such candidate is the modal logic of provability which is characterised by finite flows of time. There are other such logics. The second aspect is not mathematical, but the aspect of style and of the naturalness of the representation. Can the presentation within classical logic cater for all the natural features and non-mathematical properties of temporal logics arising in application areas?

One needs to make the above vague notion of 'naturalness' more precise. We know for example, that in classical logic with $\sim$ and $\vee$, implication $A \to B$ is definable as $\sim A \vee B$. Thus the logic of $\sim$ and $\to$ is reducible to (or can be represented within) the logic of $\vee$ and $\sim$ by a simple translation. It can also be described in predicate logic by first Gödel numbering all wffs with $\to$ and $\sim$ then using a proper proof predicate. This is a roundabout way of doing a simple reduction of $\to$ and $\sim$ to $\vee$ and $\sim$ but it is mathematically valid. Yet, we intuitively know that the reduction

$$A \to B \equiv \sim A \vee B$$

is a more direct one and indeed a better one.

There are more examples where the representation may be on the borderline of being natural. Our task is therefore to attempt to make our intuitions more precise. In algebra for example, it is generally accepted, that although a finite dimensional vector space V is isomorphic both to its dual V^* and to its dual V^{**}, it is the latter isomorphism $V \leftrightarrow V^{**}$ which is the 'natural' one. The attempts to make this notion more precise led to the development of category theory.

The best way of dealing with our problems is first to translate some temporal logics into the predicate calculus and then, in the light of what we have done, examine the general notions of reduction and translation involved. We need to perform some translations first, to see some of our mathematical options. For this purpose we have to build up some extensions of the predicate calculus i.e. by adding temporal connectives to predicate logic. We also look at temporal logics built (or simulated) within predicate logic. We need a notation system which will help distinguish between these two approaches. We thus write **TL1, TL2**, ... to name temporal logics using temporal connectives. Technically these are proper extensions of predicate logic and write **PC1, PC2**, ... to name two sorted systems of predicate logic with a special sort for time. Schematically we want to check whether, and how, we can say that: **PC=TL**.

In the terminology of this section, **TL** is the object language **L** and **PC** is a corresponding object language implementation of metaproperties of **L**, ie **PC** is meta to **TL** in the second object level sense.

Let us first see how the **PC** languages can be built. We begin by isolating from among the predicates and variables of the general predicate calculus some special symbols which will have temporal meaning. These symbols become distinguished in the same way that the equality symbol $=$ is distinguished in predicate logic. These include:
a unary predicate T for the flow of time,
a transitive binary relation $<$ for the earlier-later ordering of time.

We thus have a two sorted predicate logic with special predicates, terms, and variables for each sort.

Example 5.3 (a) The property that time is linear can be written as:
$(\forall t, s)(t < s \vee t = s \vee s < t)$

(b) 'John loves Mary at time t' can be written as:
Love(t,John,Mary)

(c) The truth table for 'A will be true' is represented by the formula $\psi_{will}(t, Q_1) =$ definition $\exists s > t\ Q_1(s)$
Where Q_1 is supposed to be $\{s' \mid A \text{ is true at } s'\}$

(d) The truth tables for *Since* and *Until*, ψ_S and ψ_U are
$\psi_S(t, Q_1, Q_2) =$ (definition) $\exists s < t(Q_1(s) \wedge \forall u(s < u \wedge u < t \to Q_2(u)))$
$\psi_U(t, Q_1, Q_2) =$ (definition) $\exists s > t(Q_1(s) \wedge \forall u(t < u \wedge u < s \to Q_2(u)))$

Example 5.4 Assume we want to formalise the properties of the English progressive tense, say in the context of a natural language processing model which can recognise certain temporal features, such as the difference between:

1. 'John walks to school',
   and

2. 'John is walking to school'.

The **TL** (temporal logic) way of talking about the progressive is to add a new unary connective Nq to the language of predicate logic and represent 'John is walking' as:
N Walk(John).
The properties of the progressive can be captured by the axioms below:

1. $Nq \to q$
   If John is walking around now, then he is actually walking now.

2. $Nq \to NNq$
   The above axiom really says that if John is walking now, then he has been walking a little bit before now, and will be walking a little bit after now.

3. $N(p \wedge q) \leftrightarrow Np \wedge Nq$
   John is walking and drinking now, if and only if John is walking now and is drinking now.

4. $\vdash q \Rightarrow \vdash Nq$
   For example, If we are walking at all times then we are walking around at any time.

Axiom 2 is needed because we want to force Nq to mean what it means and we want to use axioms involving only N.
If we allow the connectives *Since* and *Until*, we can define N in terms of S and U by:
$Nq =$ (definition) $q \wedge S(\top, q) \wedge U(\top, q)$.
But then of course we need axioms on S and U.
The **PC** representation of the progressive is to associate with it the formula $\mathbf{N}(t, Q)$ of predicate logic where:
$\mathbf{N}(t, Q) =$ (definition) $\exists t_1 \exists t_2 (t_1 < t < t_2 \wedge \forall s(t_1 < s < t_2 \to Q(s)))$
The nature of the progressive is determined by this formula. This formula **N** is said to be the *table* for N in the terminology of of the definitions in section 6 below.
The axioms for N given in the **TL** style of presentation are all valid and follow in predicate logic from **N**. In other words, the interpretation from the modal language into the language of classical logic which translates $\Box q$ into $\mathbf{N}(t, Q)$ is sound relative to all axioms of the modal logic, namely, all of these axioms are translated into theorems of classical logic. In fact the translation is also complete, namely if a modal formula is translated into a classical logic theorem then the modal formula is provable from the modal axioms. However the condition of completeness of the axioms does not determine the translation formula **N** uniquely. Imagine reading Nq as $N_1 q$ below:
$N_1 q = q$ is true from now onwards.
N_1 satisfies the same axioms as N. The **PC** presentation of N_1 is $\mathbf{N}_1$, namely:
$\mathbf{N}_1(t, Q) =$ (definition) $\forall s(t < s \to Q(s))$.
It can be mathematically proved that over e.g. rational numbers (or real numbers) flow of time, N and N_1 satisfy the same logical axioms. Over the integers time, q becomes q i.e. $\mathbf{N}(t, Q) = Q(t)$ while $\mathbf{N}_1(t, Q)$ is not $Q(t)$. Thus we see that the two methods of representation, i.e. **TL** and **PC** are really stylistically different. There are more examples in the next section.

The moral of the N, N_1 example is that the axiomatic presentation of the connective N (or N_1) does not admit a unique meaning and allows for a family of meanings in one axiomatic presentation. To present N or N_1 within classical predicate logic we have to use one of its meanings. We thus lose the other meanings. Of course one can always use predicate logic as a proper metalanguage (use Gödel numbers for naming) or just term translation and simulate the axiom system for N (or

N_1). This will indeed yield the axiomatic system for N (or N_1) exactly, but it is wrong to say that we have 'expressed' N or N_1 in predicate logic (in the object level). We use predicate logic instead of English to describe the system for N_1. We use no properties of predicate logic to give meaning to the modal connectives. Although such term translations are used, they are useful mostly for automated reasoning purpose, where the classical resolution machine is helpful.

We are going to translate the metalanguage predicates defined in 5.0.1 into classical logic. We associate for each n-place atomic predicate formula $A(x_1, \ldots, x_n)$ of the temporal logic, a 2 sorted formula $A^*(t, x_1, \ldots x_n)$ of classical logic, where t ranges over a new sort for time. We also allow for the predicate $<$ to be binary over the time sort. Other formulas of this two sorted logic are built as usual from atoms of the form $t < s, A^*(t, x_1, \ldots, x_n)$ the connectives and the quantifiers on each sort.

Definition 5.5 We now translate the metapredicates **Data**('t', 'A'), **Hold**('$<$', 't', 's') and **Prove** as follows:

1. **Data**('t', '$A(x_1, \ldots, x_n)$')$^* = A^*(t, x_1, \ldots, x_n)$ where $\bot^* = \bot$.
2. **Data**('t', '$A \sharp B$')* = **Data**('t', 'A')$^* \sharp$ **Data** ('t', 'B')* where $\sharp$ is any of the connectives $\wedge, \vee, \rightarrow$.
3. **Data**('t', 'GA')$^* = \forall s(t < s \rightarrow$ **Data** ('s', 'A')$^*)$
4. **Data**('t', 'HA')$^* = \forall s(s < t \rightarrow$ **Data** ('s', 'A')$^*)$
5. **Data**('t', 'FA')$^* = \exists s(t < s \wedge$ **Data** ('s', 'A')$^*)$
6. **Data**('t', 'PA')$^* = \exists s(s < t \wedge$ **Data** ('s', 'A')$^*)$
7. **Hold** ('$<$', 't', 's')$^* = (t < s)$
8. **Prove** ('t','Ψ', 'φ')$^* = (\Psi^* \rightarrow \varphi^*(t))$

Note that in (8) $\varphi^*(t)$ is really **Data**('t', 'φ')*.

The importance of (8) is that the translation of '**Prove**' is the classical implication '$\rightarrow$'.

It is easy to show that the translation '$\rightarrow$' of '**Prove**' satisfies the properties listed under Definition 5.0.1.

We have seen above two ways of representing our temporal logic **L** in another language **M** (functioning as a metalangauge). One was pure meta, through the predicates **Hold**, **Data** and **Prove** and one was an object level 'meta', through classical predicate logic. We need to systematise and formally develop machinery for both approaches. For the first approach, the proper metalanguage approach, we offer the language **HFP**, to be studied later in this paper. For the predicate logic approach, we must give the proper definitions of how predicate logic can be used. We begin with predicate logic in the next section.

6 Translations into classical logic: technical case study

We saw in the previous section that predicate logic can be used to represent temporal connectives by dedicating a special sort variable t for moments of time and using it to either represent the truth table of the temporal connectives or to represent the proof theory of the connective. This section gives precise definitions of how this is done. The basic idea of the translation can be explained by example as follows.

Let **L** be any logic with connectives $\sharp_i$, $i = 1, \ldots, n$ which is complete for a possible world semantics. Assume the basic models have the form (see 6.0.12 for details) $(S, R_i(t_1, \ldots, t_{r(i)}), a, h, D)$ where $a \in S$ is the actual world, $R_i \subseteq S^{r(i)}$ is the possible world relation, $D \neq \varnothing$ is a domain and for each k-place atomic $Q, h(Q) \subseteq S^{r(i)} \times D^k$ and where R_i, $i = 1, \ldots, n$ satisfy some conditions and the assignment h satisfies some restriction. For example modal propositional logic has semantics of the form (S, R, a, h), with $R \subseteq S^2, R$ transitive and h possibly satisfying some restriction such as, the condition

- for all atomic $q, t \in h(q)$ and tRs imply $s \in h(q)$.

We consider the classical theory of a general (S, R_i, a, h) without any restrictions. Then a non-classical logic can be characterised by the following parameters

- The restrictions on R_i (reflexivity, seriality, etc) and S via a formula $\mathbf{R}$. $\mathbf{R}$ is a formula involving the predicates $S(t)$ and $R_i(t_1, \ldots, t_{r(i)}), i = 1, \ldots, n$. It expresses properties of these predicates. Given any classical model of $\mathbf{R}$ with domain M, the extension ofthe predicate S gives us the set of possible worlds and the extensions of R_i restricted to S give us the relations on S. In case of temporal logic, where the possible world relation is already agreed to be just a partial order, the main use of $\mathbf{R}$ is to characterise the domain S. For example, $\mathbf{R}$ may say $\forall t[t \in S \rightarrow t \geq 0]$.
- Restrictions on h (via a formula $\mathbf{H}$).
- Truth tables for the connectives i.e. $t \vDash \sharp_i(A_1, \ldots, A_{r(i)})$ iff some formula $\psi_i(t, Q_1, \ldots, Q_{r(i)}, R_1, R_2, \ldots)$ holds, where Q_i is intended to denote the set $\{s \mid s \vDash A_i\}$. The table ψ_i is written using S, $R_1, \ldots, R_n$. Usually only R_i, and S are used for $\psi_i, i = 1, \ldots, n$.

So given a logic $\mathbf{L}$ with semantics characterised by $\mathbf{H}$ and $\mathbf{R}$ and a φ of $\mathbf{L}$ we can write $\mathbf{H} \wedge \mathbf{R} \rightarrow \varphi^*$ in classical logic, where φ^* is the translation of φ. This says 'φ holds in all models' i.e. $\mathbf{L} \vdash \varphi$.

This section defines the above for temporal logic, where (S, R_i) is obtained from a partial order, denoted by $(T, <)$. This is a simplification but not mathematical weakening, since arbitrary relations can be coded in partial orders and thus R_i can be suitably retrieved. However, instead of evaluation in one dimension $(t \vDash \varphi)$ we have m dimensions $((t_1, \ldots, t_m) \vDash \varphi)$.

Definition 6.1 [General form of the language PC] We define the fragment **PC** of classical logic capable of handling time as follows. We augment classical logic into two sorted language. The pure flow of time sort, containing the binary predicate $t < s$, the unary predicate $T(s)$ and $S(s)$. The variables t, s will belong to the flow of time sort. We also have mixed predicates of the form $Q(t_1, \ldots, t_m, x_1, \ldots, x_n)$. $k \geq 1$. the first m co-ordinates take the time sort, the others take the object (other) sort. (In the propositional case the object sort is not needed, i.e. we have predicates of the form $Q(t_1, \ldots, t_m)$).

(a) The atomic predicates of the fragment have the form given by a_1 or a_2 but not a_3, below.

(a_1) $Q(t_1, \ldots, t_m, x_1, \ldots, x_n)$, where Q is an atomic symbol, $x_1, \ldots, x_n$ are object (non -time) variables and t_i are time variables. t_1 must appear in Q. $x_1, \ldots, x_n$ may not appear in Q (i.e. $Q = Q(t_1)$ is allowed). Q is a two sorted predicate.

(a_2) $t < s, t = s$, for t, s time variables

(a_3) $t < x, t = x, x < y, x < t$,are not allowed, for x, y object variables.

(b) the fragment is closed under $\wedge, \vee, \sim, \rightarrow$ and under quantification $\forall$ and $\exists$.

Example 6.2 [An m-dimensional propositional table with restriction $\mathbf{R}$] Consider a k place connective $\sharp(q_1, \ldots, q_k)$. Since we are dealing with m-dimensional semantics, the possible world models for $\sharp$ have the form $(S, R_1, \ldots, R_n, a, h)$, where n is the number of connectives of the language and R_i are relations with the appropriate number of places on S. m-dimensional satisfaction is defined for sequences $(t_1, \ldots, t_m)$ from S^m. Thus the extension of $\sharp(q_1, \ldots, q_k)$ is a set $Q \subseteq S^m$, comprising all tuples $(t_1, \ldots, t_m)$ such that $(t_1, \ldots, t_m) \vDash \sharp(q_i, \ldots, q_k)$. This set must be obtained in some first order way from the extensions of $q_1, \ldots, q_k$.
If we let

$$Q_i = \{(s_1, \ldots, s_m) \mid (s_1, \ldots, s_m) \vDash q_i\}$$

then Q is obtained from Q_i via a formula $\psi_\sharp$.

The formula $\psi_\sharp$ has m free variables and is built up from m-place predicates $Q_1, \ldots, Q_k$, the domain predicate $S(t)$, the relations R_i. S, R_i must satisfy the restriction formula $\mathbf{R}$. In the case of temporal logic, S and R_i are all generated from the temporal flow $(T, <)$, and $\psi_\sharp$ can be directly built up from $Q_1, \ldots, Q_k, <$. The restriction formula $\mathbf{R}$ need only define S (the part relating to R_i is absorbed into $\psi_\sharp$). We can thus view $\mathbf{R}$ as a one place formula restricting S. We can thus define:
Any Wff built up from atoms of the form $S(s)$, where s is a time variable, and $Q_i(t_1, \ldots, t_m), i = 1 \ldots k$, using quantifiers over $t \in T$ only, and containing m free time variables is called an m-dimensional table for a k-place connective with a parameter S.

Definition 6.3 (the general form of the language TL) Let $\sharp_1, \ldots, \sharp_k$ be symbols for $m_1, \ldots m_k$ - place connectives. The syntactical temporal extension of predicate logic with the connectives $\sharp_1, \ldots, \sharp_k$ is defined as follows. It is called $\mathbf{TL}(\sharp_1, \ldots, \sharp_k)$.

(a) Any predicate Wff is a temporal Wff.

(b) If $A_1, \ldots, A_{m_i}$, are temporal Wffs, so is $\sharp_i(A_1, \ldots, A_{m_i})$.

(c) The set of temporal wffs is closed under the classical connectives and quantifiers.

Example 6.4 The first-order semantical presentation of any propositional **TL** logic of the syntactical form $\mathbf{TL}(\sharp_i)$ (as defined above) is obtained by associating with each $\sharp_i$ a table ψ_i with a parameter S in the sense of Example 6.0.2. The tables should all be of the same dimension m and for a connective $\sharp_i$ the table is m_i place. The semantics we are giving in this case is m-dimensional. Thus with each $\sharp_i$ we associate a formula $\psi_i(t_1, \ldots, t_{m_i}, S, Q_1, \ldots Q_{m_i})$.

Example 6.5 Consider the following connective F^* . The dimension is 2. the number of places is 1. The table for F^* has the form:

$$\psi_{F^*}(t, s, Q_1) = [t = s \wedge (\exists t' > t) Q_1(t', s)] \vee [t > s \wedge Q_1(t, t)]$$

$$\vee [t < s \wedge \forall t' (t < t' < s \rightarrow Q_1(t', t'))]$$

Here we do not use the parameter S.

Definition 6.6 (a) The axiomatic presentation of an $\mathbf{TL}(\sharp_i)$ logic is obtained by writing axioms and rules for the connectives $\{\sharp_i\}$.

(b) In case we can prove that an axiomatic **TL** presentation and a semantical **TL** presentation define the same logic (i.e the same set of valid Wffs) then we say we proved the completeness of the axioms for the semantics. Note that we haven't yet precisely defined the notion of valid sentences.

Example 6.7 Consider the example Nq of the progressive from the previous section. There are four temporal logics involved with this connective:

(a) An axiomatic presentation:

(1) $Nq \rightarrow q$

(2) $Nq \rightarrow NNq$

(3) $N(p \wedge q) \leftrightarrow Np \wedge Nq$

(4) $\vdash q \Rightarrow \vdash Nq$

(b) A first-order semantical presentation via the table:

$$\mathbf{N}(t, Q) = \exists t_1, t_2 (t_1 < t < t_2 \wedge \forall s (t_1 < s < t_2 \rightarrow Q(s)))$$

(c) A first-order semantical presentation via the table:

$$\mathbf{N}_1(t, Q) = \forall s(t < s \to Q(s)).$$

(d) A topological semantical presentation as follows:
Let each atom q get a set in a topological space. Let Nq mean
Nq=(definition) the topological interior of q.
Thus the table $\mathbf{N}_2$ for N is:
$\mathbf{N}_2(t, Q)$iff $t \in$ Interior(Q).
The table is not even first order and involves a topology over sets. This topology may be connected with some natural ordering on the topological space.

I don't think there is any sense in saying that the above four logics for the connective N are the same. It is true that the set of valid sentences in all four logics is the same, but this is all they have in common.

Thus to represent the axiomatic presentation (a) above in first order logic, we have to use one of its interpretations (eg (b) or (c)). We are thus forced to commit ourselves to one interpretation, while the **TL** axiomatic presentation gives no committment. This is a similar situation to what we have in logical representation of legal rules. The presentation (translation) of the rules into logic already imposes a reading on them and thus depriving the rules from any other interpretation. One's tendency in the legal context is not to translate.[21]

To further see the difference between the logics, try and see what is the meaning of some extensions of each logic in terms of the other logics. Consider the following:

Example 6.8 (a') Extend the logic (a) of the previous example with the axiom of Dummett as simplified by Geach:

$$N(N(q \to Nq) \to q) \to (\sim N \sim Nq \to q)$$

This axiom makes sense in terms of axioms and rules but what does it mean in terms of tables or topologies?

(b', c') Add the condition that the flow of time is dense and linear, to (b) and (c) of the previous example. Does this correspond to an axiom? How does it affect the table $\mathbf{N}_2$ in case the topology is obtained from some natural ordering.

(d') Add the condition that the topological space of (d) is discrete. Does this have any counterpart in the other systems?

(e') Suppose we restrict our atomic formula q to being true only at a finite set of points or its complement (i.e. give them finite sets or complements of finite sets in the topological interpretation). What is the effect of this restriction on the various logics?

Exercise 6.9 The reader may wish to figure out what is the effect of conditions (a') ... (e') on the respective logics.

Given the topological interpretation of N, it is obviously not first order. It has the same valid theorems as the logic (6.0.7b) and (6.0.7c), which obviously can be represented as first order. Do we choose one of them, say (6.0.7b) and claim that we have represented the logic (d') within the framework of the predicate calculus? Haven't we represented something completely different?

We need standards and protocols for translations from one language to another. The definitions below give a standard translation of semantically presented temporal logics whose table is first order. Such examples are (b) and (c) of the previous example 6.7.

We saw that an $\mathbf{TL}(\sharp_i)$ logic can be presented semantically via a table. The table does not yet define the logic uniquely, it just defines the meaning of the connectives. Thus the table $\mathbf{N}$ for N

[21] Legal texts have (1) syntactic ambiguity and (2) open textured terms. We are disregarding (1) and referring to (2).

defines one logic for the case that time is the rational numbers but reduces to $Nq = q$ in the case that time is the integers. Thus to completely define the set of valid sentences of a logic presented to us we must also specify exactly the class of temporal models for this logic. The relevant notions will be defined later in this section. We have enough information in our disposal to be able to reduce semantically defined **TL** logics to **PC**, provided the **TL** tables are first order formulas.

Definition 6.10 We define a translation $*$ from any m-dimensional semantical (1st order) presentation of an $\mathbf{TL}(\sharp_i)$ logic for $\sharp_i$ into **PC** as follows:

(a) Any atomic $Q(x_1, \ldots, x_n)$ of **TL** is reduced to $Q^*(t_1, \ldots t_m, x_1, \ldots, x_n)$ of **PC**. (i.e. time co-ordinates are added). In the propositional case, the atom q will translate into $q^* = Q$.

(b) The translation $*$ commutes with the classical connectives and quantifiers.
$(A \wedge B)^* = A^* \wedge B^*$
$(A \vee B)^* = A^* \vee B^*$
$(\sim A)^* = \sim A^*$
$(\forall x A(x))^* = \forall x A^*(x)$
$(\exists x A(x))^* = \exists x A^*(x)$

(c) The m-dimensional connective $\sharp_i(A_1, \ldots, A_{m_i})$ with table ψ_i is translated using the table:

$$\sharp_i(A_1, \ldots, A_{m_i})^* = \psi_i(t_1, \ldots, t_m, S, A_1^*, \ldots, A_{m_i}^*)$$

Example 6.11 The translation of $N(Nq \to q)$ of the previous Example 6.7 with table:

$$\mathbf{N}(t, Q) = \exists t_1, t_2, (t_1 < t < t_2) \wedge \forall s(t_1 < s < t_2 \to Q(s))$$

is:

$$\exists t_1, t_2(t_1 < t < t_2) \wedge \forall s(t_1 < s < t_2) \to [Nq \to q] * (s)$$

where

$$(Nq \to q)^*(s) = (Nq)^*(s) \to q^*(s) = \exists x_1, x_2(x_1 < s < x_2$$
$$\wedge \forall y(x_1 < y < x_2 \to Q(y)) \to Q(s))$$

Thus the translation is:

$$(\exists t_1, t_2)[(t_1 < t < t_2)] \wedge \forall s[t_1 < s < t_2] \to (\exists x_1, x_2)((x_1 < s < x_2))$$
$$\wedge \forall y(x_1 < y < x_2 \to Q(y)) \to Q(s)$$

The semantical presentation of definition 6.0.6 and the translation of definition 6.0.10 are good for any flow of time. To define any specific temporal logic, we need to fix the flow of time. Thus we are led to the following:

Definition 6.12 (a) An **TL** semantical presentation of an m-dimensional temporal logic with constant domains and with connectives $\sharp_1, .., \sharp_k$,is determined by the following components:

(a1) m-dimensional tables ψ_i for each connective $\sharp_i, i = 1, \ldots, k$. ψ_i is a formula $\psi_i(t_1, \ldots, t_m, S, Q_1^*, \ldots, Q_{m_i}^*)$ where $\sharp_i$ is m_i-placed. S is unary and $Q_1^*, \ldots, Q_{m_i}^*$ are m-place predicates.

(a2) A family **K** of flows of time of the form $(T, <, S)$, with $S \subseteq T$. The flows of time in **K** must satisfy the restriction **R**. Consider the first order language with the predicate $<$ (binary) S, T (unary) and $Q_1^*, \ldots, Q_n^*$ (m-place). By abuse of notation, $<, T, S$ represent $<$ and T and S of the flow of time with $S \subseteq T$ and Q_i^* can range over the sets of the form $\|A\|^h = \{(t_1, \ldots, t_m) \mid (t_1, \ldots, t_m) \vDash A$, for some wff A of **TL**$\}$. **H** and **R** are in the above language. **R** restricts the model $(T, <, S)$ and **H** restricts the assignment h to the atomic wffs $A(x_1, \ldots, x_n)$ to sets of the form $\|A\|^h$ which satisfy **H**.

(b) A temporal model has the form $(T, <, S, h, D) \in \mathbf{K}$ where S and h satisfy the restrictions $\mathbf{H}$ and $\mathbf{R}$. D is the non-empty domain of the model and h is an assigment giving for each $t_1, \ldots t_m \in T$ and each n-place atomic predicate Q of the temporal language a subset $h(t_1, \ldots, t_m, Q) \subseteq D^n$. (D, h) must satisfy the restriction $\mathbf{H}$ and $\mathbf{R}$ for $(T, <, S, h)$.

(c) The truth set $\|B\|$ of each formula $B(x_1, \ldots, x_n)$ of the temporal language is defined by induction, for each substitution $x_i/y_i \in D$ (i.e. we are defining $\|B(y_1, \ldots, y_n)\|$ for each $y_i \in D$):

(1) $\|Q(y_i)\| = \{t|(y_1, \ldots, y_n) \in h(t, Q)$, where $t = (t_1, \ldots, t_m)\}$.

(2) $\|(B_1 \wedge B_2)\| = \|B_1\| \cap \|B_2\|$

(3) $\|(\sim B)\| = T^m - \|B\|$

(4) $\|(B_1 \vee B_2)\| = \|B_1\| \cup \|B_2\|$

(5) $\|(B_1 \to B_2)\| = (T^m - \|B_1)\| \cup \|B_2\|$

(6) $\|(\exists x B_1(x))\| = \{t|$ for some $d \in D, t \in \|B_1(d)\|, t = (t_1, \ldots t_m)\}$

(7) $\|\forall x B_1(x)\| = \{t|$ for all $d \in D, t \in \|B_1(d)\|, t = (t_1, \ldots, t_m)\}$.

(8) $\|\sharp_i(B_1, \ldots, B_{m_i})\| = \{t|\psi_i(t, B_1, \ldots, B_{m_i})$ holds in $(T, <, S, h, D)\}$

(9) We say a formula $B(x_1, \ldots, x_n))$ holds at $(T, <, S, h, D)$ iff its universal closure holds, i.e. for any $d_1, \ldots, d_n \in D, \|B(d_1, \ldots, d_n)\|^h \supseteq S^m$.

(10) We say $B(x_1, \ldots, x_n)$ is valid in the logic if $B(x_1, \ldots, x_n)$ holds at any $(T, <, S, h, D)$, for which h and D satisfy the restrictions $\mathbf{H}, \mathbf{R}$.

Example 6.13 (a) Take the connective N with the table of Example 6.0.7(a) and a constant $\mathbf{c}$. Take as the flows of time in $\mathbf{K}$ all linearly ordered $(T, <)$. Take a subset $C \subseteq T$ as a truth table subset for c. Take as restrictions $\mathbf{H}$,$\mathbf{R}$ the condition $\mathbf{H}(Q)$ for any atom Q, and $\mathbf{R}(t)$ below:

Let for atomic q which is not the constant $\mathbf{c}$ (i.e. predicate Q (associated with q) which is not the predicate $\mathbf{C}$ associated with $\mathbf{c}$):

(1) $\mathbf{H}(Q) = \exists t \forall x(\forall s > t)(Q(s, x) \leftrightarrow Q(t, x))$, for $Q \neq \mathbf{C}$(ie $q \neq \mathbf{c}$).

(2) $\mathbf{H}(\mathbf{C}) = \exists t \sim \mathbf{C}(t) \wedge \exists t \mathbf{C}(t) \wedge \forall s, s'(\mathbf{C}(s) \wedge s < s' \to \mathbf{C}(s'))$

(3) Let $\mathbf{R}(D)$ say that $D = \{t \mid\sim \mathbf{C}(t)\}$.
Thus $\mathbf{H}(Q)$ says that after some time t, Q keeps on giving the same values to any x. $\mathbf{H}(\mathbf{C})$ says that $\mathbf{C}$ is initially false and then becomes true. $\mathbf{R}$ says the domain is determined by $\mathbf{C}$ being false.

(b) Take as flows of time all finite chains for the connective N of 6.0.7a, which is the same connective taken in (a) above, (without $\mathbf{c}$). Take no restrictions $\mathbf{H}$ or $\mathbf{R}$.

One can show that the logic of the connective N_b (i.e. N of 6.0.7b above) is related (can be translated) into the logic of the connective N_a of 6.0.7a by the (more or less) following translation:

$$N_b q \equiv\sim \mathbf{c} \wedge N_a(q \vee \mathbf{c})$$

We can now say what it means, as a first approximation, for a temporal logic to be 1st order expressible in $\mathbf{PC}$. The temporal logic must have tables ψ_i for its connectives $\sharp_i$ and 1st order conditions defining the class $\mathbf{K}$ of flows of time and first order conditions defining the restrictions $\mathbf{R}$,$\mathbf{H}$ on the assignment. If a logic $\mathbf{L}_1$ is defined by other means, not by semantical means, then we can say that if the set of valid theorems of this logic is the same as the set of another logic $\mathbf{L}_2$ which is expressible in $\mathbf{PC}$ then $\mathbf{L}_1$ is also considered expressible in $\mathbf{PC}$. We are not ready yet to give the above as formal definition, partly because we are going to reject it, as it is too restrictive and misses out on some intuitions. The reader is referred to the case studies for examples of different translations and their discussion.

7 Linked predicate languages: classical logic as a target for translation

The previous sections discussed to various degrees of detail, three seemingly independent families of concepts:

- We discussed the proposed *LDS* discipline.
- We discussed at the beginning of Section 5 the possibility of two languages $\mathbf{L}_1$ and $\mathbf{L}_2$ being linked in some way and through that link one can implement some metalevel features of the other. This was shown to be a way of looking at *LDS*, namely that we are linking the labels of the algebra with the formulas.

  This linkage of $\mathcal{A}$ and $\mathbf{L}$ can be done for any two predicate systems within predicate logic.
- Section 6 showed how temporal connectives can be 'talked about' from within predicate logic. The temporal meta predicate '$A(x_1, \ldots, x_n)$ holds at time t' is translated into predicate logic as $A^*(t, x_1, \ldots, x_n)$, where A^* is a two sorted $n+1$ place predicate, with t a variable of the time sort and $x_1 \ldots x_n$ of the element sort. The relation $<$ on time also has to be introduced as well as suitable axioms to ensure that the interpretation of the temporal system within predicate logic functions properly.

  We also saw how an *LDS* system say $(\mathcal{A}, \mathbf{L})$ can be translated into two sorted classical logic, with two sorts, one for the labels and one for formulas. We thus have $t : A(x)$ is translated into $A^*(t, x)$. t is a term in $\mathcal{A}, A$ a wff in $\mathbf{L}$. The sort t is from the algebra and so has an algebraic language for manipulating it. A^* is a predicate of two sorts.

There is a more general way of looking at what we are doing, which is quite independent of the above particular examples. It has to do with linking two languages via the sharing of variables. The task of this section is to develop this theme.

We show how to prepare (separate, present) classical logic as a two sorted system so that it can serve as a target for translation. For this we need to 'link' predicate languages, which we denote by $\mathbf{G}$ and $\mathbf{L}$ to emphasise the complete generality.

Obviously a lot depends on the mechanism of 'linking'. We are going to clarify this notion for the case of two predicate logic theories. This will suffice for the handling of reduction of *LDS* or temporal logic into classical logic. $\mathbf{G}$ acts as object level language implementing metalevel restrictions on $\mathbf{L}$.

Consider the first order language $\mathbf{G}$ with a binary relation $<$ and variables $\{t, s, \ldots\}$. Think of it as a theory of the flow of time. Consider the predicate language $\mathbf{L}$, with variables $\{x_1, x_2, \ldots\}$ and predicates $A(x_1, \ldots, x_n)$, $B(y_1, \ldots, y_m)$. The formal operation of what we have done was to 'combine' these two languages by replacing $\mathbf{L}$ by $\mathbf{L}^*$, where $\mathbf{L}^*$ is a two sorted language, with variables $\{t, s, \ldots\}$ and $\{x_1, x_2, \ldots\}$ and taking as atomic predicates the two sorted predicates $A^*(t, x_1, \ldots, x_n)$, $B^*(s, y_1, \ldots, y_n)$, etc.

We allow for common quantification and mixed wffs. Thus we can write for example the mixed formula $\varphi(t)$

$$\varphi(t) = \forall s(t < s \to A^*(t, x_1, \ldots, x_n))$$

which you will recognise as the truth table for '$GA(x_1, \ldots, x_n)$ holds at t'.

Formally φ is just a formula in the mixed language. Let us give a quick definition to clarify our concepts before we continue our discussion.

Definition 7.1 The two sorted language $\mathbf{L}_k^*(\mathbf{G})$. Let $\mathbf{L}$ and $\mathbf{G}$ be the two languages. Both $\mathbf{L}$ and $\mathbf{G}$ have atomic predicates. Let $\mathbf{L}_k^*$ be obtained from $\mathbf{L}$ by replacing each atomic predicate $R(x_1, \ldots, x_n)$ of $\mathbf{L}$ by $R^*(t_1, \ldots, t_k, x_1, \ldots x_n)$ where t_i are variables of a new sort. Thus the resulting $\mathbf{L}_k^*$ is a two sorted language, with the new sort for t-variables. We allow quantification over t-variables. The intention is that the t-variables will be from the language $\mathbf{G}$.

1. The atomic wffs of $\mathbf{L}_k^*(\mathbf{G})$ are of the form $R^*(t_1, \ldots, t_k, x_1, \ldots, x_n)$, where $R(x_1, \ldots, x_n)$ is an atomic wff of $\mathbf{L}$ and $t_1, \ldots t_k$ are terms of $\mathbf{G}$, or of the form $Q(t_1, \ldots, t_m)$, where $Q(t_1, \ldots, t_m)$ is an atomic wff of $\mathbf{G}$.

2. If φ and ψ are wffs of the language with t free in φ and x free in φ then $\varphi \wedge \psi, \varphi \vee \psi, \varphi \to \psi, \sim \varphi, \forall t\varphi, \forall x\varphi, \exists t\varphi, \exists x\varphi$ are wffs of the language.

Example 7.2 1. Let $\mathbf{G}$ be the language of order $<$ and let $\mathbf{L}$ be classical predicate logic. $\mathbf{L}_1^*(\mathbf{G})$ gives a language with atoms $R^*(t, x_1, \ldots x_n)$, which we used in the previous example to mean 'R is true at t'.

2. The language $\mathbf{G}$ can be a metalanguage. Consider a propositional language, eg the propositional language with $\neg, \wedge, \vee$ and $\to$. For each wff A of the propositional language introduce a constant τ_A. Introduce the operations $\mathbf{f}_\sim(\tau_A) = \tau_{\sim A} \quad \mathbf{f}_\wedge(\tau_A, \tau_B) = \tau_{A \wedge B}$ and similarly $\mathbf{f}_\vee$ and $\mathbf{f}_\to$. Let $\mathbf{G}$ be the language with τ_A as terms and $\{\mathbf{f}_\sim, \ldots\}$ as functions. The language $\mathbf{L}_1^*(\mathbf{G})$ talks about the formulas of the propositional language as labels $R^*(\tau_A, x_1, \ldots x_n)$ represents $A : R(x_1, \ldots x_n)$, where A acts as a label.

The above linkage is still not the most general case. To get an idea of what we need, consider the following set-up. Consider a distributed system of stations $s_1, s_2, s_3, \ldots$ related in some manner. Each station has its own internal structure and language $\mathbf{L}$. Assume they all use the same language and logic. We also need a global configuration language to describe how these stations are related. This language denoted by $\mathbf{G}$ (for global) need not be the same logic as $\mathbf{L}$. For example $\mathbf{G}$ can be based on intuitionistic logic, while $\mathbf{L}$ is based on classical logic. $\mathbf{L}$ may be a Horn clause logic programming language while $\mathbf{G}$ could be a form of Petri net langauge. The stations s_i need to communicate, so they would have output and input ports which can be accessed by the other stations. For simplicty assume that there are some variables $x_1, \ldots, x_k$ which can be accessed by the stations. This means that the language $\mathbf{G}$ can talk about x_i.

If we want to link these two languages into one, we can form the syntactic combination $\mathbf{L}_k^*(\mathbf{G})$ of the previous definition. The additional feature to worry about is that the logic of $\mathbf{G}$ and $\mathbf{L}$ may not be the same. Consider the following cases: ($A^*(t, x)$ could mean 'at station t mailbox x is down', and $B^*(t, x)$ can mean 'at station t printer x is down'.)

1. $A^*(t, x) \to B^*(s, y)$
2. $A^*(t, x) \to B^*(t, y)$
3. $A^*(t, x) \to B^*(s, x)$
4. $A^*(t, x) \to B^*(t, x)$

In cases 2 and 4 we can say that this is an internal statement of station t and hence the implication follows the logic of $\mathbf{L}$. In case 3 we can use the logic of $\mathbf{G}$ as this is a relation between stations.

What do we do in case 1?

The simplest course of action is to understand it as a relationship between stations and hence use the logic of $\mathbf{G}$. We thus have a simple rule. Any formula with *different* station constants in it is to be manipulated logically according to $\mathbf{G}$ but if the constants are equal, it is to be manipulated according to $\mathbf{L}$. In fact once we agree to that, different stations can have different logics, $\mathbf{L}_t$. They must all be based on the same language.

We have one restriction on the system, that is the configuration logic must be sound with respect to the station logics. In symbols, $\Delta \vdash_{\mathbf{G}} A$ in the configuration logic implies $\Delta \vdash_{\mathbf{L}} A$ in the station logic. The reason for that is unification. We must be able to reason about two possibly different stations in $\mathbf{G}$ and if we identify them our reasoning (now in $\mathbf{L}$) must still be valid.

We now consider the Horn clause fragment of $\mathbf{L}_k^*(\mathbf{G})$. The modal and temporal logics of the case studies used the language $\mathbf{G}$ with $<$, the linking of classical predicate logic with the theory of order $<$. In fact, we were operating in the Horn clause fragment of $\mathbf{L}_1^*(<)$. We should therefore define the Horn clause fragment of $\mathbf{L}_k^*(\mathbf{G})$ in general.

Definition 7.3 The Horn clause fragment of $\mathbf{L}^*_k(\mathbf{G})$.

1. Any atom of the form $R^*(t_1, \ldots, t_k, x_1, \ldots, x_n)$ of $\mathbf{L}^*_k$ and $Q(t_1, \ldots, t_m)$ of $\mathbf{G}$ is in the Horn clause fragment and in the generated body fragment.

2. If A and B are in the Horn clause fragment or in the generated body fragment then so are $A \wedge B, \forall xA, \exists xA, \forall tA, \exists tA$.

3. If A is in the generated body fragment and B is in the Horn clause fragment then $A \to B$ is in the Horn clause fragment.

Example 7.4 1. If $A(t, x), B(s, y)$ are atomic, then $A' = \forall t \exists x A$ is in the atomic fragment and $C = A' \to \exists s \forall y B$ is in the Horn clause fragment. In classical logic, one can write C as:

$$\exists t \forall x \exists s \forall y (A \to B)$$

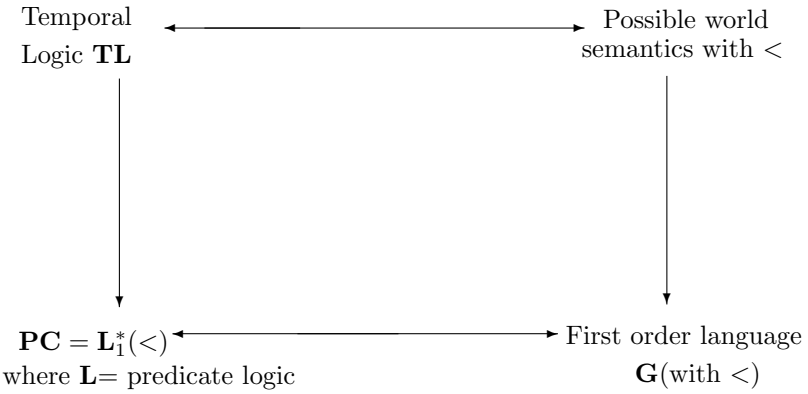
because we can pull the quantifiers to the front and hence the previous definition can be simplified. In intuitionistic logic this is not possible and so we need clause 3 of the previous definition and the notion of atomic fragment, in order not to be committed to classical logic.

2. Notice that for $k = 1$ and $\mathbf{G}$ the theory of order $<$, the Horn clause fragment of $\mathbf{L}^*_1(<)$ gives us the Horn fragment for temporal logic.

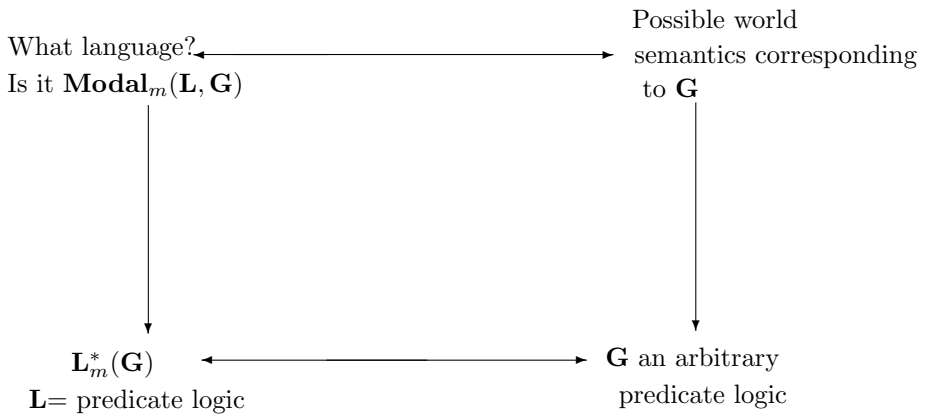
The above discussion about linking two languages $\mathbf{L}$ and $\mathbf{G}$ is a general discussion of how a language $\mathbf{G}$ can be linked to a language $\mathbf{L}$ and 'influence it' by forming $\mathbf{L}^*_k(\mathbf{G})$. We gave no particular meaning to this operation beyond the 'metalevel' one. We did see that this linkage did generalise the special construction of **PC** in the previous section. $\mathbf{G}$ corresponded to the time order relation $<$ and $\mathbf{L}$ corresponded to the predicate logic. The linkage formulated in definition 7.0.1 allows for linking arbitrary $\mathbf{G}$ and $\mathbf{L}$.

Can we give modal or temporal meaning to $\mathbf{L}^*_k(\mathbf{G})$ for arbitrary $\mathbf{G}$ and $\mathbf{L}$?

Furthermore, we have the following diagram, which we want to generalize:



We want to complete the following diagram:



What would be the counterpart of **TL** for arbitrary **G** and **L**?

Obviously we must generalise the notion of a temporal connective based on time ordering $<$ to some general modal connective based on some language **G**.

We are now ready to define a modal logic $\mathbf{Modal}(\mathbf{L},\mathbf{G})$ of **L** and **G**. To explain our options, let us reconsider the basic temporal connectives F and P. The truth table for $FA(x)$ is:
$FA(x)$ is true at t iff for some $s > t$, $A(x)$ is true at s.

If $A(x)$ is atomic, we translate it into $A^*(t,x)$, understanding $A^*(t,x)$ as '$A(x)$ is true at t' and thus we get:

$$FA(x) \text{ is true at } t \text{ iff } \exists s(t < s \wedge A^*(s,x))$$

or

$$(FA(x))^*_t = \exists s(t < s \wedge A^*(s,x))$$

In $\mathbf{L}^*_1(<), <$ is the atomic relation of the order. $A^*(t,x)$ is an atomic relation of $\mathbf{L}^*_1$. 'F' represents the existential quantifier on the variable s in $t < s \wedge A^*(s,x)$ and it is denoted by 'F'. The existential quantifier on t would be $\exists t(t < s \wedge A^*(t,x))$ and that would correspond to 'P'.

Let us generalise this situation to $\mathbf{L}^*_1(\mathbf{G})$. Here the atoms $A^*(t,x)$ are still the same, but the atoms of **G** could be any $Q(t_1,\ldots,t_m)$. The parallel construction would be:

$$\exists t_i(Q(t_1,\ldots,t_i,\ldots,t_m) \wedge A^*(t_i,x))$$

We could introduce for each atom $Q(t_1,\ldots,t_m)$ of **G** and each $1 \leq i \leq m$ the existential connective $\Diamond_{Q,i}A(x)$ to mean

$$(\Diamond_{Q,i}A)^*_t = \text{ (def) } \exists t_i[Q(\cdots t_i \cdots) \wedge A^*(t_i,x)]$$

Similarly we can define $\Box_{Q,i}A$

$$(\Box_{Q,i}A)^*_t = \text{ (def) } \forall t_i[Q(\cdots t_i \cdots) \rightarrow A^*(t_i,x)]$$

These correspond to 'G' and 'H' of temporal logic. To summarise we have:

$$\begin{array}{ll} \Box_{<,2} & = G \\ \Box_{<,1} & = H \\ \Diamond_{<,2} & = F \\ \Diamond_{<,1} & = P \end{array}$$

Let us consider what *Since*(A,B) does in this context.

$S(A,B)$ is true at t iff $\exists s < t[A$ is true at s and $\forall y(s < y \wedge y < t$ implies B is true at $y)]$.

Let ψ_S be the following formula in a language with $<$ and the two *new monadic* predicates Q_1 and Q_2.

$$\psi_S(t,Q_1,Q_2) = \text{ (def) } \exists s < t[Q_1(s) \wedge \forall y(s < y \wedge y < t \rightarrow Q_2(y))]$$

Then we have $S(A,B)$ is true at t iff $\psi_S(t,A^*,B^*)$, where A^*,B^* are considered monadic predicates in the **G** sort.

To generalise the above, let $\mathbf{G}_1^+$ be the extension of the language of **G** with new monadic predicates $\{Q_1,Q_2,\ldots\}$. Let $\psi(t,Q_1,\ldots,Q_n)$ be a formula of $\mathbf{G}_1^+$ with 1 free variable t and the n new monadic predicates. Let $A_1^*,\ldots,A_n^*$ be n formulas of $\mathbf{L}_1^*$, which can be considered monadic in the sort t. We can introduce the n place connective $\sharp(A_1,\ldots,A_n)$ with the truth table

$\sharp(A_1,\ldots,A_n)$ true at t iff $\psi(t,A_1^*,\ldots,A_n^*)$.

We will now give the general definition and an example. Further mathematical study of these concepts can be found in my book [Gabbay *et al.*, 1993].

Definition 7.5 Let **G**, **L** be two languages. We construct the modal language $\mathbf{Modal}_m(\mathbf{L},\mathbf{G})$ as follows:

1. Let $\mathbf{G}_m^+$ be the language **G** supplemented with an infinite sequence of *new* m place atomic predicates $\{R_1,R_2,\ldots\}$.

2. Let $\mathbf{L}_m^*(\mathbf{G})$ be the language obtained from **L** as in definition 7.0.1. We can regard each atomic formula $A^*(t_1,\ldots,t_m,x_1,\ldots,x_n)$ as m-place atomic in the variables $t_1,\ldots,t_m$.

3. Let $\mathcal{G}$ be any model of **G**. We regard $\mathcal{G}$ as a model of possible worlds. With each $t_1,\ldots,t_m\in\mathcal{G}$, associate a model $\mathcal{L}_{(t_1,\ldots,t_m)}$ of **L**. Assume all the $\mathcal{L}_{(t_1,\ldots,t_m)}$ models have the *same* domain D (constant domain semantics). We refer to $(\mathcal{G},\mathcal{L}_{(t_1,\ldots,t_m)})$ as a *modal model*.

4. For each wff $\psi(t_1,\ldots,t_m,R_1,\ldots,R_n)$ of $\mathbf{G}_m^+$ with t_i free and R_i new atomic predicates, associate a n place connective $\sharp_\psi$. The full modal language $\mathbf{Modal}_m(\mathbf{L},\mathbf{G})$ is $\mathbf{L}\cup\{\sharp_\psi \mid \psi\in\mathbf{G}_m^+\}$.

Definition 7.6 Given a modal model $(\mathcal{G},\mathcal{L}_{(t_1,\ldots,t_m)})$ and an assignment h of **L** into D, a model $\mathcal{G}^h$ of $\mathbf{G}_m^+$ is induced on $\mathcal{G}$ as follows.
Let $A^*(t_1,\ldots,t_m,x_1,\ldots,x_n)$ be an atom of $\mathbf{L}_k^*$. It can be viewed, for $x_1,\ldots,x_n$ fixed, as an atomic predicate of $\mathbf{G}_m^+$. Let the extension of this predicate in $\mathcal{G}$ be $\{(t_1,\ldots,t_m) \mid \mathcal{L}_{(t_1,\ldots,t_m)}\vDash A(h(x_1),\ldots,h(x_n))\}$.
We define the notion of a formula $A(x_1,\ldots,x_n)$ of $\mathbf{Modal}_m(\mathbf{L},\mathbf{G})$ holding at the indices $(t_1,\ldots,t_m)$, notation $\|A\|^h_{t_1,\ldots,t_m}=1$, as follows:

1. $\|A\|^h_{(t_1,\ldots,t_m)}=1$ iff (def) $\mathcal{L}_{t_1,\ldots,t_m}\vDash A(h(x_1),\ldots,h(x_n))$.
   $\|A\|^h_{t_1,\ldots,t_m}=0$ otherwise.

2. $\|A\wedge B\|^h=1$ iff $\|A\|^h=\|B\|^h=1$

3. $\|\sim A\|^h=1$ iff $\|A\|=0$.

4. $\|\exists xA(x)\|^h=1$ iff for some $d\in D$, and h_1, such that $h_1(x)=d$ and h_1 agrees with h on all other variables we have $\|A(x)\|^{h_1}=1$

5. $\|\sharp_\psi(A_1,\ldots,A_n)\|^h_{t_1,\ldots,t_n}=1$ iff $\mathcal{G}_m^+\vDash\psi(t_1,\ldots,t_m,Q_1,\ldots,Q_n)$
   where $\mathcal{G}_m^+$ is the extension of $\mathcal{G}$ to a model of the language $\mathbf{G}_m^+$ by the assignment
   $Q_i=\{(t_1,\ldots,t_m) \mid \|A\|^h_{t_1,\ldots,t_m}=1\}$.

6. Let $\mathcal{K}$ be a class of **G** models $(\mathcal{G},(s_1,\ldots,s_m))$ with a distinguised seqeunce of $s_i\in\mathcal{G}$. Let φ be a formula of $\mathbf{Modal}_m(\mathbf{L},\mathbf{G})$. We say $\mathcal{K}\vDash\varphi$ iff for all models $(\mathcal{G},\mathcal{L}_{(t_1,\ldots,t_m)})$ and all h, $\|A\|^h_{s_1,\ldots,s_m}=1$

Example 7.7 To show the generality of the previous definition, we give an example from another area of logic. Let $\mathcal{G}$ be a commutative semigroup with multiplication $*$ and a unit 1. That is, we have the axioms:

1. $\forall xyz((x*y)*z=x*(y*z))$

2. $\forall xy(x * y = y * x)$

3. $\forall x(1 * x = x)$

Let **L** be classical propositional logic. Consider the formulas $\psi(t, Q_1, Q_2)$ and $\varphi(t, Q_1, Q_2)$ where:

$$\psi(t, Q_1, Q_2) = \text{ def } \forall y[Q_1(y) \to Q_2(t * y)]$$

$$\varphi(t, Q_1, Q_2) = \text{ def } \exists xy[Q_1(x) \wedge Q_2(y) \wedge t = (x * y)]$$

Then the logic with the connectives $\sharp_\psi(A, B)$ and $\sharp_\varphi(A, B)$ is linear logic with linear implication and external conjunction.

$$\sharp_\psi(A, B) = A \multimap B$$

$$\sharp_\varphi(A, B) = A \otimes B$$

The semantics induced on these connectives by the general definition can be directly defined as follows.
The set of possible worlds is a semigroup $\mathcal{G}$. The propositional assignment h gives a truth value to every atom at a point ie $h(t, q) \in \{0, 1\}$ for $t \in \mathcal{G}$, q atomic. The truth conditions for $\multimap$ and $\otimes$ are as follows:

$$\|A \multimap B\|_t^h = 1 \text{ iff } \forall y(\|A\|_y^h = 1 \text{ implies } \|B\|_{t*y}^h = 1)$$

$\|A \otimes B\|_t^h = 1$ iff for some x, y we have $\|A\|_x^h = 1$ and $\|B\|_y^h = 1$ and $t = x * y$.
We have $\vDash A$ iff for all $\mathcal{G}$ and all h, $\|A\|_1^h = 1$.

We will not pursue further these topics. See my forthcoming books on Labelled Deductive Systems and Temporal Logic [Gabbay, 1994] and [Gabbay *et al.*, 1993].

8 The metalanguage HFP: computational classical logic

The previous two sections studied first the possiblity of classical logic serving as a metalanguage, and second the general mechanism of linking two languages **G** and **L**, so that the linkage can implement metalanguage features of the modal system based on **G**. In both cases our examples used classical logic for **L**. Since different languages **G** seem to serve as the object language which implements the meta features involved turning classical logic **L** into different new logics $\mathbf{L}_1^*(\mathbf{G})$, we ask ourselves, is there a convenient general purpose language **M** which is specially suited for expressing meta language features, and which will contain all the $\mathbf{L}_1^*(\mathbf{G})$?

The answer is yes. The language we have in mind is the language **HFP**.[22]

The relevance of the existence of **HFP** to the Debate is as follows. We claim that classical logic or its variants are universal languages at least from the point of view of automated deduction. So given for example an arbitrary logic **N** we can translate it into $\mathbf{L}_k^*(\mathbf{G})$, for some suitable **G** (usually obtained from the semantics or equivalent *LDS* proof theory of **N**), an appropriate dimension k and classical logic **L**. This method suffers from some disadvantages. The target system is many sorted and may be computationally inconvenient. We also get for different $\mathbf{N}_1, \mathbf{N}_2, \ldots$ different target many sorted languages $\mathbf{L}_{k(i)}^*(\mathbf{G}_i), i = 1, 2, \ldots$. We would like one very nice language which is convenient and can serve any logic **N**. We propose the language **HFP** and we will show at the end of this section how any logic **N** and any $\mathbf{L}_k^*(\mathbf{G})$ can be nicely translated into **HFP**.

We begin by describing the intuition behind the language **HFP**. Consider first propositional temporal logic, formalised in the usual way, via the addition of extra connectives to classical propositional logic. Assume the connectives are FA and PA, 'A will be true' and 'A was true' respectively.

In symbols:

[22] In fact, the Logic Programming Community, when working in extensions of logic programming, are actually working in **HFP**, more specifically in Horn clause **HFP** which, you will be surprised to learn, is essentially Micro-PROLOG. Unfortunately, no one uses Micro-PROLOG anymore.

$$\|FA\|_t = 1 \text{ iff } \exists s > t \quad \|A\|_s = 1$$

$$\|PA\|_t = 1 \text{iff } \exists s < t \quad \|A\|_s = 1.$$

Suppose we want to choose the first moment s_0 such that $s_0 > t$ and $\|A\|_{s_0} = 1$, assume such a moment exists. This moment is a function of t and A. Let us denote it by $s_0 = f_1^1(A, t)$. Here f_1^1 is a two place function, taking a formula and a point and yielding a point.

Consider now the connective F itself, this connectives takes a formula A yields a formula FA, thus its function is $R_0^1(A)$.

The functionality of f_1^1 and R_0^1 can be described set theoretically. If X is the set of all points in which A is true, we get

$$f_1^1(X, t) = \min (X \cap \{s \mid s > t\})$$

$$R_0^1(X) = \{s \mid \exists t \in X(s < t)\}$$

In general we can have general connectives and functions of the form f_n^m, R_n^m, taking m formulas and n points and yielding a point or a formula repsectively.

To give another example, consider the connective $S(A, B)$. Its meaning is that since a point in the past where A was true, B has been true all the time. We can now form a new connective $S_y^x(A, B)$ reading since a point t in the past which is different from x in which A is true, B has been continuously true at all points since which are greater than y. This connective has the form $S(A, B, x, y)$.

The language **HFP** is a general language in which tables and connectives can be described. It allows for predicates R_n^m and function symbols f_n^m to take both formulas and terms as arguments.

The use of **HFP** is not restricted to the metalevel of modal and temporal languages. It is also the metalanguage of actual PROLOG programs. Although officially logic programming deals with the Horn clause fragment of classical logic (possibly with negation by failure), namely with clauses of the form

$$\bigwedge A_i \to B$$

where A_i and B are atoms, in practice the PROLOG programmer uses **HFP** as the language. (Negation by failure can be incorporated using a metalevel failure predicate).

Expressions like *Hold*(A, B) *Demo*$(A(x), B(y))$ are written in PROLOG. The above are expressions of **HFP**. It may be thought that *Demo* and *Hold* are metapredicates. This is partially correct. They have special properites. The variables x and y in *Demo*$(A(x), B(y))$ can be quantified and unified. Thus *Demo* is not like the general metapredicates of a pure metalanguage **M** but rather special and we claim it is like in **HFP**. (We shall see later in this Section how to quantify on seemingly object level variables occurring in a metalevel predicate.)

We see in definiton 8.0.3 below that **HFP** is given modal semantics. So the question is whether this semantics is adequate for the way PROLOG uses the language. We will be in a better position to answer in the next section. The answer in general is no, PROLOG predicates are not in general modal connectives, but under certain restrictions they can be viewed as such though this is not the most convenient way of looking at them.

We are now ready to define the general metalanguage **HFP**, of *Hereditarily Finite Predicates*.

Definition 8.1 (The Language **HFP**)
The language contains the classical connectives $\sim, \wedge, \vee, \to$ and the variables $\{x, y, z, \ldots\}$ for terms and variables $\{X, Y, Z, \ldots\}$ for formulas, and the quantifiers $\forall$ and $\exists$. There is a list of atomic predicates R_n^m allowing us to form $R_n^m(\psi_1, \ldots, \psi_m, x_1, \ldots, x_n)$, where ψ_i are formula variables or formulas and x_j are term variables or terms . There are function symbols f_n^m allowing us to form $f_n^m(\psi_1, \ldots, \psi_m, x_1, \ldots x_n)$, where ψ_i are formula variables and x_j are term variables. We define now the notions of a *wff* and a *term* of the language **HFP**.

1. $\top$ is a formula with no free variables.

2. x is a term with x free. Y is a formula with Y free.

3. If $\psi_i(x^i_1, \ldots x^i_{k(i)}), i = 1, \ldots, m,$ are formulas with free variables $\{x^i_1, \ldots, x^i_{k(i)}\}$ and $F_j(\overline{x}_1, \ldots, \overline{x}_{r(j)}), j = 1, \ldots, n,$ are terms with $\{\overline{x}^j_1, \ldots, \overline{x}^j_{r(j)}\}$ as free variables and R^m_n is a predicate symbol with (m, n) places and f^m_n is a function symbol with (m, n) places then:

   (a) $R^m_n(\psi_1, \ldots, \psi_m, F_1, \ldots, F_n)$ is a formula with the above free variables, namely $(\overline{x}^i_j, x^t_s), i = 1, \ldots, m, j = 1, \ldots, k(i), t = 1, \ldots, n, s = 1, \ldots, r(t)\}$.

   (b) $f^m_n(\psi_1, \ldots, \psi_m, F_1, \ldots, F_n)$ is a term with the same free variables as in (a).

   The free variables mentioned can be either term or formula variables.

4. If ψ_1, ψ_2 are wffs, then so are $\sim \psi_1, \psi_1 \wedge \psi_2, \psi_1 \rightarrow \psi_2, \psi_1 \vee \psi_2$. $\sim \psi_1$ has the same free variables as ψ_1, and $\psi_1 \wedge \psi_2, \psi_2 \vee \psi_2$ and $\psi_1 \rightarrow \psi_2$ have a set of free variables which is the union of the sets of free variables of ψ_1 and ψ_2.

5. If $\psi(x, y_i)$ is a formula with free variables $\{x, y_i\}$ being either term or formula variables then $\forall x\psi, \exists x\psi$ are formulas with the free variables $\{y_i\}$.

Example 8.2 To give some examples consider the **Hold** predicate. This has the form **Hold** (ψ, t) = 'ψ is true at time t'.
We can now write **Hold**(**Hold**$(\psi, t), s)$

'At time s it was true that ψ was true at t';

compare this sentence with **Thought**(**Thought**$(\psi, 0), 5)$

'At time 5 people thought that at time 0 it was thought that ψ was true';

consider **Thought-F**(ψ, t) which reads:

'At time t, ψ was considered true in the future'.

Thus we can write a rule: If at any time you expect to get a budget for a Research Assistant, then hire one. (It may be that you never get the budget!).

$$\forall t[\textbf{Thought-F}(\text{budget}, t) \rightarrow \textbf{Hire}(t)].$$

We now define several types of models for this language.

Definition 8.3 (Modal Semantics)
Let D be a nonempty set. A function h is an assignment for **HFP** iff the following holds:

1. If x is an individual term variable then $h(x) \in D$. If x is a formula variable then $h(x) \subseteq D$.

2. If R^m_n is a predicate symbol then $h(R^m_n)$ is a function with domain $(2^D)^m \times D^n$ and range 2^D

3. If f^m_n is a function symbol then $h(f^m_n)$ is a function with domain $(2^D)^m \times D^n$ and range D.

4. Given a formula $\psi(x_1, \ldots, x_n)$ with free variables $x_1, \ldots x_n$ and a term $F(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$ we define the value $h^*(F) \in D$ and $h^*(\psi) \subseteq D$ as follows:

   (a) $h^*(x) = h(x)$

   (b) $h^*(\top) = D$

   (c) $h^*(R^m_n)(\psi_1, \ldots, \psi_m, F_1, \ldots, F_n) =$
   $h(R^m_n)(h^*(\psi_1), \ldots, h^*(\psi_m), h^*(F_1), \ldots, h^*(F_n))$

   (d) $h^*(f^m_n(\psi_1, \ldots, \psi_m), F_1, \ldots, F_n) =$
   $h(f^m_n)(h^*(\psi_1), \ldots, h^*(\psi_m), h^*(F_1), \ldots, h^*(F_n))$

(e) $h^*(\sim \varphi) = D - h^*(\varphi)$
$h^*(\varphi \wedge \psi) = h^*(\varphi) \cap h^*(\psi)$

(f) $h^*(\forall x \psi(x)) = \bigcap_{d \in D} h^*_{x/d}(\psi(x))$
where $h_{x/d}$ is the function which differs from h only by having $h_{x/d}(x) = d$.
ie
$h_{x/d}(y) = h(y)$ for $y \neq x$
$h_{x/d}(y) = d$ for $y = x$
Similarly
$h^*(\exists x \psi(x)) = \bigcup_{d \in D} h^*_{x/d}(\psi(x))$.

5. (a) A model is a domain and an assignment, (D, h).
   (b) A formula ψ holds in a model if $h^*(\psi) = D$
   (c) A formula ψ is valid if it holds in all models.

Example 8.4 1. Consider classical propositional temporal logic. We can associate with any atomic proposition q a 0-place predicate Q of **HFP**. The domain D is time and assume an order $<$ on D. Q is a $(0,0)$ predicate. Let $\mathbf{F}$ be the $(1,0)$ predicate $\mathbf{F}(Q)$. Let $h_<(\mathbf{F})(Y)$, for $Y \subseteq D$ be the set $Y^+ = \{s \mid \exists y \in Y\, s < y\}$. Then if $h_<(Q)$ is the set where q is true then $h^*_<(FQ)$ is the set of points where Fq is true.

2. Consider the example in (a) above and consider the connective (predicate) $\mathbf{F}(Q, x)$. $\mathbf{F}$ is a $(1,1)$ predicate. It corresponds in the propositional temporal logic to a labelled connective $F^x q$. We may have for example $F^x q$ true at t iff $t < x$ and q true at x. In **HFP** the assignment would be:
$$h_<(\mathbf{F})(Y, x) = \{t \mid t < x \text{ and } x \in Y\}$$

3. Notice that the **Hold** predicate of 8.0.2 should be of type $\mathbf{Hold}^1_0$, ie $\mathbf{Hold}(A)$ if the modal semantics associates a set with it. Thus $\mathbf{Hold}(A) \equiv A$, ie $h(\mathbf{Hold}) \equiv$ identity.

The above discussion gave to **HFP** modal semantics. This semantics is adequate for describing the modal and temporal applications of **HFP**, as it talks about truth conditions of connectives. However, it does not seem satisfactory in explaining the way **HFP** predicates are used in practical PROLOG programs. Whenever we write in PROLOG a clause of the form

$$\mathit{Hold}(\mathit{Demo}(A(x), B(y)) \wedge A(z)) \rightarrow B(z)$$

We are using **HFP** expressions involving predicates of predicates. Sometimes we even write clauses of the form

$$X \rightarrow P(X)$$

which play an important role in metaprogramming.

For example the clause

$$\mathbf{Hold}(\varphi) \text{ if } \varphi$$

may look more familiar to the reader.

The logical status of these phrases and their semantics have to be explained. The main characteristic feature of such use is that the variables in the clauses can unify and change value. Thus it is not immediately obvious whether one can say that in the expression $\varphi(A(x), x)$, $A(x)$ is a name of a formula. If $A(x)$ represents the wff $A(x)$ (ie what we really have is $\varphi(\text{`}A(x)\text{'}, x)$, then the '$x$' in $A(x)$ cannot change or unify, certainly not in uniformity with the other $x$. In practice of course when '$x$' is unified to be '$a$', we get $\varphi(A(a), a)$ and not $\varphi(\text{`}A(x)\text{'}, a)$.)

We thus need to explain the above for the case of **HFP**, because like in PROLOG, **HFP** allows for x to be free in $\varphi(A(x), x)$ in both places.

Notice that in the modal semantics the above are interpreted as connectives. So for example $\varphi(B, y)$ can be interpreted as the connective saying: *B will always be true but not later than time y*, and $A(x)$ can be interpreted as saying: *Time x is in the future (of now)*. Together we form

$\varphi(A(x), x)$ to mean: *Time x will always be in the future but not later than time x*, which is really a temporal tautology.

We now proceed to give the *metalevel semantics* for **HFP**, as opposed to the *modal semantics*. Our strategy is to begin with ordinary predicate logic, describe a metalanguage **M** for it and identify **HFP** as part of the metalevel description of predicate logic in **M**.

Definition 8.5 [Term translation] Let **L** be predicate logic with predicate symbols $\{P_i, i = 1, 2, 3, \ldots\}$ which are n_i place; variables $\{x_i, y_i, \ldots\}$ and constants $\{a_i, b_i, \ldots\}$. The language **L** may contain the classical connectives and in general a stock of m-place connectives of the general form $\sharp(A_1, \ldots, A_m)$. We refer to **L** as *the object language*. A language **M** is said to be *a metalanguage* for **L** if it is capable of naming the symbols of **L** and contains the following representing terms (this translation is referred to as *term translation*):

1. For each symbol of **L** there exists a possibly unique name which is a term of **M**. The naming function is denoted by quotation marks.
   - variables x_i are named 'x_i'
   - constants a_i are named 'a_i'
   - predicate symbols P_i are named 'P_i'
2. For each P_i, which is n_i place predicate, there exists a function $\mathbf{f}_i$ of the metalanguage **M**, satisfying the following:
   $\mathbf{f}_i$('P_i','t_1', ..., 't_{n_i}') = '$P_i(t_1, \ldots t_{n_i})$'
   In fact, we can take the above as a definition of the name of $P_i(t_1, \ldots, t_{n_i})$.
3. There exist functions $\mathbf{f}_\wedge, \mathbf{f}_\vee, \mathbf{f}_\to, \mathbf{f}_\sim, \mathbf{f}_\forall$ and $\mathbf{f}_\exists$ and $\mathbf{f}_\sharp$ for each m-place connective, satisfying equations as follows:
   - $\mathbf{f}_\wedge$ ('A', 'B') = '$A \wedge B$'
   - $\mathbf{f}_\vee$ ('A', 'B') = '$A \vee B$'
   - $\mathbf{f}_\to$ ('A', 'B') = '$A \to B$'
   - $\mathbf{f}_\sim$ ('A') = '$\sim A$'
   - $\mathbf{f}_\forall$ ('A', 'x') = '$\forall x A(x)$'
   - $\mathbf{f}_\exists$('A', 'x') = '$\exists x A(x)$'
   - $\mathbf{f}_\sharp$ ('A_1', ..., 'A_m') = '$\sharp(A_1, \ldots, A_m)$'
4. There are functions **Sub**, **s** and **n** satisfying the following, for terms t:
   $\mathbf{n}(t)$ = 't'
   $\mathbf{s}$('t') = t
   Sub ('$A(x)$', 'x', $\mathbf{n}(t)$) = '$A(t)$'
   where t is any term and 't' is its name.

Note that **Sub** is a substitution function. We can assume that $\mathbf{f}_\sharp, \mathbf{f}_i, \mathbf{f}_\wedge, \mathbf{f}_\vee, \mathbf{f}_\sim, \mathbf{f}_\exists$ and $\mathbf{f}_\forall$ are actual function symbols of **M** and thus actually define the terms which name the formulas. **Sub** cannot be assumed as a function symbol because then a formula might have several names. In general metalevel language with functions representing substitution, the names will not be unique. Several function symbols including the substitution function symbol can combine to yield names for a formula in several different ways. If we choose a unique name via our formula construction functions as we did, the substitution function may not be representable.

So we have to assume that **Sub** is somehow definable in **M**. An alternative is to take **Sub** as a function symbol and add axioms to **M** to make things right.

Lemma 8.6 Let $\varphi(x_1, \ldots, x_n)$ be a formula, then there exists a term
$\mathbf{f}_\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of **M** with free variables $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (ranging over names) such that
$\mathbf{f}_\varphi$ ('a_1', ..., 'a_n') = '$\varphi(a_1, \ldots, a_n)$'

Proof. By induction on the structure of φ. For atomic $\varphi = P_i(x_1, \ldots x_{n_i})$, we have

$$\mathbf{f}_\varphi = \text{ (def) } \mathbf{f}_i(`P_i', \mathbf{x}_1, \ldots, \mathbf{x}_{n_i}).$$

If there exist terms $\mathbf{f}_A$ and $\mathbf{f}_B$ for A and B then $\mathbf{f}_{A\sharp B} = \mathbf{f}_\sharp(\mathbf{f}_A, \mathbf{f}_B)$ for $\sharp$ standing for $\wedge, \vee, \rightarrow$ and similarly for $\sim$ and for a general m-place $\sharp$.

Let $\varphi = \forall x\psi(x)$. Let $\mathbf{f}_\psi$ be given. Then $\mathbf{f}_\varphi = \mathbf{f}_\forall(\mathbf{f}_\psi, `x')$. Similarly for $\varphi = \exists x\psi(x)$. $\lhd$

Definition 8.7 [Metalevel semantics] Let $\mathcal{M}$ be a model of **M**. Let **Hold** be a unary predicate of **M**. We can derive from $\mathcal{M}$ and **Hold** a model $\mathcal{A}$ of **L** by defining

$$\mathcal{A} \vDash P(a_1, \ldots, a_n) \text{ iff (def) } \mathcal{M} \vDash \mathbf{Hold}(`P(a_1, \ldots, a_n)')$$

We called **Hold** a Hold predicate because we expect that for any $\varphi(a_1, \ldots, a_n)$ we have;

$$\mathcal{A} \vDash \varphi(a_1, \ldots, a_n) \text{ iff } \mathcal{M} \vDash \mathbf{Hold}(`\varphi(a_1, \ldots, a_n)')$$

This will not be the case unless we require axiomatically in **M** that **Hold** satisfies some axioms for example:

$$\mathbf{Hold}(`A') \wedge \mathbf{Hold}(`B') \leftrightarrow \mathbf{Hold}(\mathbf{f}_\wedge(`A', `B'))$$

and similarly for the other connectives and quantifiers.
Note the quantifier axioms are:

$$\forall \mathbf{x}\mathbf{Hold}(\mathbf{f}_{\varphi(x)}(\mathbf{x})) \leftrightarrow \mathbf{Hold}(\mathbf{f}_{\forall x\varphi(x)})$$

$$\exists x\mathbf{Hold}(\mathbf{f}_{\varphi(x)}(\mathbf{x})) \leftrightarrow \mathbf{Hold}(\mathbf{f}_{\exists x\varphi(x)})$$

Remark 8.8 Note that if we have **Sub** then we can define the diagonal function by:
Diag ($`A(x)$, $`x'$) = def **Sub** ($`A(x)'$, $`x'$, $\mathbf{f}_\sim$ ($`A(x)'$))
Recall that using the **Diag** function one can prove Tarski's inconsistency theorem, that there exists no truth predicate for **M** in **M**. Thus we cannot have a **Hold** predicate satisfying the axiom
Hold ($`A'$) $\leftrightarrow A$
and allow **Diag** to apply to **Hold** as well, because these conditions yield Tarski's inconsistency theorem. We need not worry however about this aspect. First, our aim is to provide semantics for **HFP** and PROLOG practice in order to do our 'computational classical logic', rather than construct self reflective languages which have their own truth predicates. So we do not mind to have the **Hold** predicate not to apply to itself. See the Perlis papers [Perlis, 1985, Perlis, 1988] for a good coverage of the subject. Second, in the Horn clause fragment of **HFP** negation is not available and it is possible to have a truth predicate in the object level for the object language itself, without leading to contradiction. The worst we can get are loops, if we use negation as failure. We cannot diagonlise with $A(`\neg A')$, with $\neg$ being negation as failure because its meaning is procedural. Even when it is given a declarative semantics, say **sem**$(A)$, we will get the formula $A(\sim \mathbf{sem}(`A'))$ which is not a direct diagonalisation. More directly we can define a **fail** connective via a program Δ, then we can only have $\mathbf{fail}_\Delta$ as a connective and we get $A(`\mathbf{fail}_\Delta A')$ for the diagonal function. In this case we get stratification of **fail** and we do not have a universal object level diagonalisation. We shall not pursue this topic any further, and will continue with the business at hand.

We are now in a position to explain what a formula of **HFP** stands for. We refer to the interpretation presented in the sequel as *the term interpretation* for **HFP**. Compare with definition 8.0.5. The object language **L** of that definition is **HFP** and the metalanguage **M** of the definition is classical logic. Note that since **HFP** extends classical logic we can say we are translating it into itself.

Suppose we name each letter x, P_i, a by itself. This can technically mean that we identify **L** as part of **M**. Thus P_i, x_i and a_i are already terms of **M**. We let $`a_i' = a_i$, $`P_i' = P_i$ and $`x' = x$ in **M**, for these particular symbols. We can define predicates **Pred**(t), **Term**(t), **Var**(t), **Wff**(t) and **Free** of **M** with the axioms:

- **Term** $(a_i), i = 1, \ldots$
- **Var** $(x_i), i = 1, \ldots$
- **Pred** $(P_i), \ldots$
- **Wff** $(\mathbf{f}_i(P_i, a_1, \ldots, a_{n_i})$
- **Wff** $(\mathbf{f}_i(P_i, x_1, \ldots, x_{n_i}))$
- **Wff** $(\mathbf{f}_i(P_i, \alpha_1, \ldots, \alpha_{n_i})$, where α_i are metalinguistic variables representing a choice of either a variable x or a term a of **L**. So for each such choice we get an axiom of **M**.
- **Free** $(\mathbf{f}_i(P_i, \alpha_1, \ldots, x_j, \ldots, \alpha_{n_i}), x_j)$
- **Wff** $(A) \wedge \mathbf{Wff}(B) \rightarrow \mathbf{Wff}(\mathbf{f}_\sharp(A, B))$
- **Wff** $(A) \wedge \mathbf{Free}(A, x) \rightarrow \mathbf{Free}(\mathbf{f}_\sharp(A, \mathbf{y}), x)$
  where $\mathbf{y}$ is a metavariable and $\sharp$ is one of $\wedge, \vee, \rightarrow, \sim$.
- **Wff** $(A) \wedge \mathbf{Free}(A, x) \rightarrow \mathbf{Wff}(\mathbf{f}_\forall(A, x))$
- **Wff**$(A) \wedge \mathbf{Free}(A, x) \rightarrow \mathbf{Wff}(\mathbf{f}_\exists(A, x))$
- **Wff**$(A) \wedge \mathbf{Free}(A, \mathbf{y}) \wedge x \neq \mathbf{y} \rightarrow \mathbf{Free}(\mathbf{f}_\forall(A, x)), \mathbf{y})$
  where $\mathbf{y}$ is a metavariable, i.e. a variable of **M**.

Having named formulas by themselves we can essentially regard the functions $\mathbf{f}_i(P_i, x_1, \ldots x_{n_i}), \mathbf{f}_\wedge, \mathbf{f}_\rightarrow$ etc to be the connectives themselves, taken as functions. So $\mathbf{f}_\varphi(`x_1', \ldots `x_n')$ will be '$\varphi(x_1, \ldots, x_n)$' which is $\varphi(x_1, \ldots, x_n)$.

If we consider $\mathbf{f}_\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ with $\mathbf{x}_i$ metavariables ranging over names, then in the convention of expressions of **L** naming themselves we get that it is equal to

$$\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n)$$

Thus we can unify and change $\mathbf{x}_i$ since they are metavariables.

Consider $\varphi(\mathbf{x}, \mathbf{y})$. This is really a term of **M**. We can certainly substitute in it the term $A(\mathbf{y})$. We thus get $\varphi(A(\mathbf{y}), \mathbf{y})$ which is an expression of **HFP**. It really stands for the term $\mathbf{f}_\varphi(\mathbf{f}_A(\mathbf{y}), \mathbf{y})$ of the metalanguage.

Satisfaction in a model $\mathcal{A}$ of $\varphi(A(\mathbf{y}), \mathbf{y})$ means according to the metalevel semantics the satisfaction in a model $\mathcal{M}$ of:

$$\mathbf{Hold}(\mathbf{f}_\varphi(\mathbf{f}_A(\mathbf{y}), \mathbf{y}))$$

Since the metalanguage is classical logic, the above term will look more familiar if we write it as $\mathbf{f}_1(\mathbf{f}_2(y), y))$, using two unary Skolem functions, $\mathbf{f}_1 = \mathbf{f}_\varphi$ and $\mathbf{f}_2 = \mathbf{f}_A$. Clearly in classical logic we can have unary predicates such as $Q(x)$. We can consider the formulas $Q(\mathbf{f}_2(y))$ and $Q(\mathbf{f}_1(\mathbf{f}_2(y), y))$. These are two non-equivalent formulas of classical logic. In an arbitrary model, one formula may be true while the other may be false, unless we have the equality axiom $\mathbf{f}_2(y) = \mathbf{f}_1(\mathbf{f}_2(y), y)$.

We need one more definition to clarify our metalevel concepts. Let Q be a classical unary predicate and let $\mathbf{g}(x_1, \ldots, x_n)$ be an n-place function. Let $\mathcal{M}$ be a model of classical logic with domain D. Define the Q denotation of $\mathbf{g}$ in $\mathcal{M}$ to be $\{(d_1, \ldots, d_n) \mid d_i \in D$ and $\mathcal{M} \vDash Q(\mathbf{g}(d_1, \ldots, d_n)\}$.

For the case where $Q =$ **Hold**, and $\mathbf{f}_2, \mathbf{f}_2$ are as above we can examine more closely the connection between the satisfaction of $A(d)$ and $\varphi(A(d), d)$. The model $\mathcal{A}$ has as its domain D terms of the metalanguage **M** which are names of wffs of **L**. $A(d)$ holds in $\mathcal{A}$ iff $\mathbf{Hold}(A(d))$ holds in $\mathcal{M}$.

The set $\{d \in D \mid \mathcal{M} \vDash \mathbf{Hold}(A(d))\}$ is the denotation of 'A' (i.e. of $\mathbf{f}_A$) and is a subset of the domain D. The denotation of 'φ' (i.e. $\mathbf{f}_\varphi$) is similarly a subset of $D \times D$. For $d_0 \in D, \mathbf{f}_A(d_0) = A(d_0)$ '$A(d_0)$' is a term in D. $\mathcal{A} \vDash A(`A(d_0)')$ may or may not hold, ie $\mathcal{M} \vDash \mathbf{Hold}(A(A(d_0)))$ may or may not hold. There is *no* connection in general between the denotation of 'A' and the term '$A(d_0)$'. We do not necessarily have $\mathbf{f}_A(d_0) \in \{d \in D \mid \mathcal{M} \vDash \mathbf{Hold}(\mathbf{f}_A(d))\}$.

Thus in the model $\mathcal{M}$ and consequently in $\mathcal{A}$ the predicate 'A' has three 'manifestations' or 'footprints', one as $\{d \in D \mid \mathcal{M} \vDash \mathbf{Hold}(A(d))\}$ one as a function symbol $\mathbf{f}_A$ and one as a term 'A' in D (i.e. the name of $\mathbf{f}_A$). **Hold** ('A') also has a value in $\mathcal{M}$, so does $\mathbf{Hold}(\text{'}A(\text{ '}A\text{'})\text{'})$ etc. These values are all independent. An **HFP** axiom of the form $X \to B(X)$ for X formula variable can be taken to mean in **M** the axiom $\mathbf{Hold}\ (X) \to \mathbf{Hold}(B(X))$, which is a connection axiom of the **Hold** in **M**. For example the axiom

$\mathbf{Hold}(A(y)) \to \mathbf{Hold}(A(A(y)))$, means $\mathbf{Hold}(\mathbf{f}_A(y)) \to \mathbf{Holdf}_A(\mathbf{f}_A(y))$.

We can in **M** read 'A' as representing the set $\{y \mid \mathcal{A} \vdash A(y)\}$. Terms of the form $\mathbf{f}_\varphi(\text{'}A\text{'}, \mathbf{y})$ can be written in the metalanguage and we can understand them in the object language as $\varphi(\lambda x A(x), y)$.

Example 8.9 The modal and temporal semantics provides us with examples of the above situation. A formula may be true at points of the interval $t = [1, 2]$ of time but whether it is true at the interval itself is another matter. For example, it is not true that one crossed the Atlantic at any moment at the interval $[1, 2]$ but one may crossed the Atlantic at $[1, 2]$. Connections between values at intervals and values at points are usually given as persistence criteria. For example, the criterion *if φ holds at an interval it holds at all subintervals*, can be expressed using our language, through axioms like:

$$\varphi(\text{'}A\text{'}, y) \to \forall x \subseteq y\ \varphi(\text{'}A\text{'}, y)$$

or

$$\forall x(A(x) \to B(x)) \to [\varphi(\text{'}B\text{'}, y) \to \varphi(\text{'}A\text{'}, y)]$$

where 'A' is the name of the predicate 'cross the Atlantic' and 'B' is the name of 'sail a boat' and φ is some metapredicate, for example $\varphi =$**Hold**.

The above freedom we still have with terms allows us to extend the expressive power of **HFP**. First let us take any binary atomic predicate of **HFP**, let us call it $P(X, Y)$ where X and Y are formula variables.

Form the formulas

$$P^2(A_1, A_2) = \text{ (def) } P(A_1, A_2)$$

$$P^{n+1}(A_1, \ldots, A_{n+1}) = \text{ (def) } P(P^n(A_1, \ldots, A_n), A_{n+1})$$

In the language **M**, there are no axioms connecting $\mathbf{Hold}(P(X, Y))$ with $\mathbf{Hold}(P(P(X, Y), Z))$ and so in $\mathcal{M}$, $\mathbf{Hold}(P^n(X_1, \ldots, X_n))$ has no connection with $\mathbf{Hold}(P^m(X_1, \ldots, X_m))$ for $m \neq n$. However, this trick allows us to regard 'P' as a sequence predicate (ie without a fixed number of places) with $P(X_1, \ldots, X_n)$ holding in $\mathcal{A}$, iff $\mathbf{Hold}(P^n(X_1, \ldots, X_n))$ holds in $\mathcal{M}$.

We can thus write our predicate as a set of sequences $(P, X_1, \ldots, X_n)$, where n is arbitrary.

We can similarly treat function symbols f and write $(f, x_1, \ldots, x_n)$. If we do the above for any pure formula predicate of the language **HFP**, we can identify the following Lisp-like fragment of **HFP**.

Definition 8.10 The sublanguage $\mathbf{HFP}_{Lisp}$
The sublanguage contains formula variables $X_1, X_2, \ldots$, term variables and Lisp-predicate letters $P_1, P_2, P_3, \ldots$.
The general notion of a wff of $\mathbf{HFP_{Lisp}}$, with free variables is defined in the same way as in the case of definition 8.0.1 of **HFP**, except that we replace the formation rule for atoms 8.0.1(3) by the following formation rule ($3_{\mathbf{Lisp}}$): If n is arbitrary and $\varphi_1, \ldots, \varphi_n$ are formulas, or predicate letters or formula variables and P is a predicate letter then $(P, \varphi_1, \ldots, \varphi_n)$ is a formula. We allow the case $n = 0$ in which case (P) is a formula.

The reader should note the translation $*$ from $\mathbf{HFP_{Lisp}}$ into **HFP**. Each P of $\mathbf{HFP_{Lisp}}$ is read in **HFP** as a binary predicate.

$(P)^* = P(\mathbf{truth}, \mathbf{truth})$
$(P, \varphi)^* = P(\mathbf{truth}, \varphi^*)$
and

$$(P, \varphi_1, \ldots, \varphi_n)^* = P^n(\varphi_1^*, \ldots, \varphi_n^*).$$

HFP can be translated into a fragment of $\mathbf{HFP_{Lisp}}$, via the translation $\sharp$:

$$P(\varphi_1, \ldots, \varphi_n)^\sharp = (P, \varphi_1^\sharp, \ldots, \varphi_n^\sharp).$$

We are now interested in defining the Horn-clause fragment of **HFP** and of $\mathbf{HFP_{Lisp}}$.

Definition 8.11 Horn Clause fragment of HFP.
We follow closely the defining clauses of 8.0.1 and of 8.0.10. Read 'Wff' as 'Wff in the the Horn fragment':

1. Same as in 8.0.1
2. Same as in 8.0.1
3. or $3_{\mathbf{Lisp}}$. Same as in 8.0.1 and 8.0.10 with the restriction that the formulas involved (i.e. $\varphi_1, \ldots, \varphi_n, \psi_1, \ldots, \psi_m, F_1, \ldots, F_n$) are all taken from the Horn fragment.
4. 1. If ψ_1 and ψ_2 are in the Horn fragment, so is $\psi_1 \wedge \psi_2$.
   2. If ψ_i, φ are atomic ie the result of applying clause (3) or ($3_{\mathbf{Lisp}}$) respectively, then $\wedge\psi_i \to \varphi$ is in the Horn fragment.
5. Same as in 8.0.1 and 8.0.10

The next question to ask is:

Do we have an implementation of the Horn fragment?

The answer is surprisingly yes, we do have in fact an old implementation, as the next remark shows.

Remark 8.12 The reader can see Clark and McCabe's book on Micro-PROLOG. Chapter 9 describes exactly the Micro-PROLOG system language which is the Horn clause fragment of $\mathbf{HFP_{Lisp}}$. Thus there exists implementations of the Horn clause fragment of this language. Micro-PROLOG on the IBM PC gives access to this language. Later versions of LPA-PROLOG concentrated on supporting Edinburgh syntax and therefore the original Micro-PROLOG syntax has been suppressed in these later versions. Another implementation for the full **HFP** is essentially the rewrite language **PLL** of E Babb, [1990].

The discussion of the section so far showed that if we name in **M** elements of **HFP** by themselves, we get that all formulas of **HFP** are terms in **M**. The term functions $\mathbf{f}_i, \mathbf{f}_\wedge, \mathbf{f}_\vee, \mathbf{f}_\to, \mathbf{f}_\forall, \mathbf{f}_\sim, \mathbf{f}_\exists$, and **Sub** of 8.0.5 all become familiar operations:

1. The naming function λx'x' is the identity.
2. $\mathbf{f}_i(P_i, t_1, \ldots, t_{n_i}) = P_i(t_1, \ldots, t_n)$
   Thus $\mathbf{f}_i$ is the P_i application function.
3. $\mathbf{f}_\wedge$ is conjunction $\wedge$
   $\mathbf{f}_\vee$ is disjunction $\vee$
   $\mathbf{f}_\to$ is implication $\to$
   $\mathbf{f}_\sim$ is negation $\sim$
   $\mathbf{f}_\forall$ is universal quantification
   $\mathbf{f}_\exists$ is existential quantification.
4. **Sub** is the substitution function.
   s, **n** are the identity function.
   $\mathbf{f}_\varphi$ for a formula φ becomes an application function for φ (just like $\mathbf{f}_i$ is for P_i).

The above functions however are functions in **M** and can therefore take any term of **M**. Assume P and Q are two unary predicates of **L**. Their names in **M** are also 'P' and 'Q'. Given a term a of **L** its name is also 'a'. The application functions $\mathbf{f}_P$ and $\mathbf{f}_Q$ of **M** allow us to form:

$$\mathbf{f}_P(\text{`}P\text{'}, \text{`}a\text{'})$$

and we are assured that:

$$\mathbf{f}_P(\text{`}P\text{'}, \text{`}a\text{'}) = \text{`}P(a)\text{'} = P(a) = \mathbf{f}_P(P, a).$$

We can, however, formally obtain the following terms of **M**.

1. $\mathbf{f}_P(\text{`}Q\text{'}, \text{`}a\text{'})$
2. $\mathbf{f}_P(\text{`}P\text{'}, \text{`}Q\text{'})$

We have no axioms in **M** to tell us what the above should mean. The **Hold** predicate of **M** would be true or false of the above terms without any restrictions. We do have axioms on **Hold** to assure us that e.g.

$$\mathbf{Hold}(\text{`}P(a)\text{'} \wedge \text{`}Q(a)\text{'}) \leftrightarrow \mathbf{Hold}(\text{`}P(a)\text{'}) \wedge \mathbf{Hold}(\text{`}Q(a)\text{'})$$

but nothing more, except for some general properties of **Hold**.

To extend our semantic meaning to terms of the form (1) and (2) above, we can first say that $\mathbf{f}_P, \mathbf{f}_Q$ should be the same, namely Application Functions. We can thus replace the n_i place function $\mathbf{f}_i$ by $\mathbf{App}_i$ and get

$$\mathbf{App}_{n_i}(\text{`}P_i\text{'}, t_1, \ldots, t_{n_i}) = \text{`}P_i(t_1, \ldots, t_{n_i})\text{'}$$

Thus all we need are the functions $\mathbf{App}_1, \mathbf{App}_2, \ldots$. So both $\mathbf{f}_Q$ and $\mathbf{f}_P$ are $\mathbf{App}_1$. This view allows us to 'compare' with general λ-abstraction languages.

Furthermore, one can code $\mathbf{App}_3(x, y, z)$ as $\mathbf{App}_2(\mathbf{App}_2(x, y), z)$ and thus all one needs is a binary **App** function. Further, there is no reason why we cannot take application to be concatenation. Thus $\mathbf{f}_i$ can be read as concatenation function. This also allows us to regard all predicates as unary predicates. $R(x, y)$ is understood either as $R(x)(y)$ or as $R((x, y))$. Probably the former is more convenient.

The above syntactical moves do not necessarily involve new semantical commitments. They can be handled as just a matter of notation. We are still left with the problem of the semantical meaning of $P(Q)$ or $P(P)$. If we understand Q to name the set $\{x \mid Q(x)\}$, then we can understand $P(Q)$ as expressing a property of this set. We can now express persistence conditions like

$$P(Q) \rightarrow \forall x(Q(x) \rightarrow P(x))$$

In general, $P(Q)$ is independent of $P(t_i)$, for t_i such that $Q(t_i)$ holds. We can see however, that we are touching on the problem of λ-calculus models.

We can still go on and ask what is the meaning of $\mathbf{App}(\text{`}a\text{'}, \text{`}Q\text{'})$. This question has now more than one option, since we have already agreed to read 'Q' as a name for $\{x \mid Q(x)\}$. We can thus read 'a' as a higher predicate on sets Q. It is true of Q iff it is an element of Q:

$$\mathbf{Hold}(a(Q)) \text{ iff } \mathbf{Hold}(Q(a)).$$

'a' can thus be identified with the set of all its properties. This is similar to the Montague semantics.

It is now up to us whether to allow in **HFP** (say in an extension which we can call $\mathbf{HFP}^+$) formulas of the form $P(Q)$ or not. If we do, we can still use the functions of the metalanguage **M** but we would need a semantical interpretation, possibly some sort of λ-calculus model. We will not pursue this line of research in this Chapter. PROLOG practice, which uses **HFP** formulas, does not frequently write expressions like:

Demo (*Demo*, *Demo*).

Let Δ be an **HFP** theory. Formally extend Δ to a saturated theory Δ'. Thus whenever $\Delta' \vdash \exists x A(x)$ then for some terms $t, \Delta' \vdash A(t)$ and whenever $\Delta' \vdash A \vee B$ then either $\Delta' \vdash A$ or $\Delta' \vdash B$. We can now define $\mathbf{Hold}_{\Delta'}(A)$ iff $A \in \Delta'$ and get a model for the **Hold** predicate of **M** as applied to the wffs of **HFP** regarded as **M** terms.

The term interpretation of **HFP** can be understood in another way, as notation for generating predicates. To illustrate our view, consider conjunction. It can generate predicates by letting $[P \wedge Q](x)$ mean $P(x) \wedge Q(x)$. This is done in the same way that $[\sim A](x)$ is $\sim A(x)$ and $[\Box A](x) = \Box A(x)$.

If we push this point of view further we can regard any formula $\varphi(x, y)$ of **HFP** as a functional $\lambda xy\varphi$ which takes formulas A, B to form a new formula $\varphi(A, B)$. This view, although *mathematically* compatible with the term semantics, is not *conceptually* compatible with the intuitive way we read PROLOG. For example, we read $Demo(A, B)$ as B can succeed from A and regard *Demo* (*Demo*, x) as meaningless.

Example 8.13 We can now show how an arbitrary *LDS* can be translated into the Horn clause fragment of **HFP**. Assume the languages **G** and **L** of *LDS* are distinct. Consider a version of **HFP** based on the union of the languages of **G** and **L** together with the additional predicate $\mathbf{Label}(x, y)$. we translate $\alpha : A$ as $\mathbf{Label}(\alpha, A)$ and all axioms and rules are translated as is. For example

$$\frac{\alpha : A,\ \beta : A \Rightarrow B}{\beta * \alpha : B}$$

is translated as
$\mathbf{Label}(\alpha : A) \wedge$ **Label** $(\beta, A \Rightarrow B) \rightarrow$ **Label** $(\beta * \alpha, B)$.

9 Semi-algebraic semantics for propositional logics

Previous sections discussed translations of logics into classical logic, translating of modal and temporal logics through their possible world semantics. Many logics do not have clear possible world semantics. Since we want to be able to translate almost an arbitrary logic into classical logic, we must develop semantic or proof theoretic methods for such logics which will help us with the translation. It seems that many logics are best presented through their algebraic semantics. A typical example is many valued logics, where the algebraic presentation is the most natural. Thus if we want to claim that a general logic can be translated into classical logic in a nice way, we must investigate a suitable general algebraic semantics. We do not want to use classical logic as a sledgehammer metalevel language. We are restricting ourselves to propositional algebraic semantics for two reasons. First because almost all known and useful non-classical logics use the usual quantifiers $\forall$ and $\exists$ and differ in their propositional part. Second we find it too difficult and complex to handle general non-standard quantifiers.

We begin by describing what we mean by algebraic semantics for a logic. We want a very general definition that would equally apply to many systems, whether they are monotonic or non-monotonic. There are many systems which have none of the classical connectives and which satisfy only some very restricted proof theory. There is no agreed standard definition of what is algebraic logic or semantics. Our primary concern in this Chapter is translation into classical logic, so any good definition will do. As it turns out our definitions are of independent interest. The new notion of algebraic logics for general consequence relations will be studied in [Gabbay, 1993b].

Logics can be identified via a general consequence relation $\Delta \mid\!\sim A$. This consequence relation will in general be non-monotonic. So we may not know, or have much to say about their consequence relation. We need minimal conditions on $\mid\!\sim$ to enable us to define a reasonable algebraic semantics for the logics which will allow us to define good translations into other logics.

Experience leads us to the following definition (which should be compared with more traditional definitions).

Definition 9.1 Let **L** be a language with atomic propositions and connectives.

1. A unitary consequence relation is a relation $\mid\!\sim$ between finite (including empty $\varnothing$) sets of wffs Δ and single wffs A, which satisfies the following:

   - *Identity*

   $$A \mid\!\sim A$$

   - *Unitary Cut*

   $$\frac{\Delta \mid\!\sim A; A \mid\!\sim B}{\Delta \mid\!\sim B}$$

   Note that we do not require reflexivity ($\Delta, A \mid\!\sim A$) nor do we require any full version of cut such as:

   $$\frac{\Delta \mid\!\sim A; \Gamma, A \mid\!\sim B}{\Gamma, \Delta \mid\!\sim B}$$

   Also note that we need not have substitutivity of equivalents, namely $A \mid\!\sim B$ and $B \mid\!\sim A$ need not imply $\varphi(A) \mid\!\sim \varphi(B)$, for arbitrary wff $\varphi(q)$. Many modal systems fail to have this property, as well as intuitionistic systems with Nelson's strong negation.

   We call the consequence relation unitary because we allow for unitary cut, the 'unit' being the single A in the rule $A \mid\!\sim B, \Delta \mid\!\sim A$ implies $\Delta \mid\!\sim B$. Also Δ being a set of single, unannotated formulas which is unstructured, as opposed to Δ being a list of wffs or a network of wffs etc.

2. A Hilbert consequence relation is a relation defined only for $\varnothing \mid\!\sim A$. A singleton consequence relation is a relation defined only for $\Delta \mid\!\sim A$, where $\Delta = \{B\}$ or $\Delta = \varnothing$.

3. Logical systems are usually formulated either as a Hilbert consequence, in which case we have only the notion of $\varnothing \mid\!\sim A$ but not $A \mid\!\sim B$, or as a singleton or unitary consequence. When a Hilbert $\mid\!\sim_1$ is given, there exists the smallest and biggest singleton $\mid\!\sim_2$ which agrees with it, (i.e. $\varnothing \mid\!\sim_1 A$ iff $\varnothing \mid\!\sim_2 A$, for all A). This will be proved later.

   Many nonmonotonic logics do not satisfy transitivity and unitary cut. We need to weaken these notions. Here is an example of a more general consequence relation in a language with $\rightarrow$ which satisfies the following rule in addition to identity but is not required to satisfy unitary cut or transitivity

   $$\frac{\begin{array}{l} A \mid\!\sim B \rightarrow C \\ X \mid\!\sim B \end{array}}{A \mid\!\sim X \rightarrow C}$$

Definition 9.2 1. A logic algebra is any algebra of the form $\mathbf{a} = (A, f_i, \leq, T)$, where A is a non-empty set (corresponding to the formulas of the logic) and $f_i, i = 1, \ldots, k$, are functions on A, with arity n_i (corresponding to the connectives of the logic) and $\leq$ is a reflexive and transitive relation on A (corresponding to the converse of the consequence relation of the logic) and $T \subseteq A, T \neq \varnothing$ is a subset of distinguished elements (corresponding to the theorems of the logic) satisfying:

$$(*) \quad \forall x, y \in A \; [y \in T \text{ and } x \leq y \text{ implies } x \in T]$$

2. A subset $F \subseteq A$ is a *filter* if $F \neq \varnothing$ and for all x, y ($x \in F$ and $x \leq y$ implies $y \in F$).

3. A function $f(x_1, \ldots, x_n)$ in a logic algebra is *extensional* iff

$$x_i \leq y_i \wedge y_i \leq x_i, i = 1, \ldots, n \text{ imply } f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n).$$

4. A function $f(x_1, \ldots, x_n)$ in a logic algebra is *directional* if it is either monotonic up or down in each of its variables.

5. An algebra is *directional* if all of its functions are directional.

Definition 9.3 1. A connective $\sharp(A_1, \ldots, A_n)$ is *extensional* iff the following holds

$$\frac{x_i \mathrel{|\!\sim} y_i, y_i \mathrel{|\!\sim} x_i, i = 1 \ldots n}{\sharp(x_1, \ldots, x_n) \mathrel{|\!\sim} \sharp(y_1, \ldots, y_n)}$$

2. A connective $\sharp(A_1, \ldots, A_n)$ is directional in A_1 iff either

   (a) *Upward monotonicity*

   $$\frac{p \mathrel{|\!\sim} q}{\sharp(p, A_2, \ldots) \mathrel{|\!\sim} \sharp(q, A_2, \ldots)}$$

   (b) *Downward monotonicity*

   $$\frac{p \mathrel{|\!\sim} q}{\sharp(q, A_2 \ldots) \mathrel{|\!\sim} \sharp(p, A_2 \ldots)}$$

3. A logic is extensional (directional) iff all of its connectives are extensional (directional, resp.) in all their variables.

Example 9.4 1. Intuitionistic logic is directional. $A \to B$ is directional up in B and down in A. So are all known substructural implications such as relevant, linear and Lambek implications.

2. Extensionality allows us to give neighbourhood semantics for the logic or to give it a Lindebaum-Tarski algebraic semantics. In the case of a language with one modal operator $\Box$, and the classical propositional connectives $\neg, \wedge, \vee$ and $\to$, extensionality of $\Box$ means we can give $\Box$ the usual neighbourhood semantics. Both $\Box$ and $\Diamond$ are monotonic upwards but satisfy different algebraic conditions.

3. Let $\sharp(x)$ be a connective and suppose it is both directional up and down in x. Then essentially $\sharp(x)$ does not depend on x.

   To see this, assume we have either truth $\top$ or falsity $\bot$ in the language. Then for any x we have

   $$\bot \mathrel{|\!\sim} x \mathrel{|\!\sim} \top$$

   Hence we get $\sharp(\bot) \mathrel{|\!\sim} \sharp(x) \mathrel{|\!\sim} \sharp(\top)$ and $\sharp(\top) \mathrel{|\!\sim} \sharp(x) \mathrel{|\!\sim} \sharp(\bot)$.

   This means $\sharp(x)$ is independent of x.

   In case neither $\top$ nor $\bot$ are in the language, we can prove 'equivalence', namely $\sharp(y) \mathrel{|\!\sim} \sharp(x)$ and $\sharp(x) \mathrel{|\!\sim} \sharp(y)$, only for x, y that are R-connected, where R is the transtive closure of $\leq$ and its converse. This is enough to allow us for definition 9.0.8, (1) to go through.

We now describe algebraic semantics for a logic $\mathbf{L}$, with consequence relation $\mathrel{|\!\sim}$. The semantics will be considerably simplified for the case of directional logics.

Definition 9.5 1. Let $\mathbf{L}$ be a logic with consequence $\mathrel{|\!\sim}$ and connectives $\sharp_1, \ldots, \sharp_k$ with arities $r_1, \ldots, r_k$ respectively. Let $\mathbf{a} = (A, f_1, \ldots, f_k, \leq, T)$ be an algebra with functions f_i with arities r_i resp.

   (a) An algebraic assignment for $\mathbf{L}$ into $\mathbf{a}$ is a function $h^{\mathbf{a}}$ assigning for each atomic q an element $h^{\mathbf{a}}(q) \in A$.

   (b) $h^{\mathbf{a}}$ can be extended to arbitrary wffs of $\mathbf{L}$ by

   $$h^{\mathbf{a}}(\sharp_i(A_1, \ldots, A_{r_i})) = f_i(h^{\mathbf{a}}(A_1), \ldots, h^{\mathbf{a}}(A_{r_i})).$$

(c) We write $(\mathbf{a}, h^{\mathbf{a}}) \models A$ iff $h^{\mathbf{a}}(A) \in T$. We write $A \models_{\mathbf{a},h\mathbf{a}} B$ iff $h^{\mathbf{a}}(B) \leq h^{\mathbf{a}}(A)$. In future we omit the '$\mathbf{a}$'.

2. An algebra is connected if any two elements are connected by the transitive closure of $R = (\leq \cup \geq)$ (i.e. of $\leq$ and its converse). This condition relates to the connectivity mentioned in Example 9.0.4 (3).

Theorem 9.6 [Completeness for general logics] Let $\mid\!\sim$ be a consequence relation then $A \mid\!\sim B$ iff for all directional algebras $\mathbf{a}$ and all assignments $h^{\mathbf{a}}$, we have $A \models_{h\mathbf{a}} B$.

Proof.

1. Soundness is immediate from the reflexivity and transitivity of $\leq$.

2. Completeness is obtained by taking the *canonical algebra* with $A =$ set of all wffs and defining $x \leq y$ iff $y \mid\!\sim x$. We let $T = \{B \mid \varnothing \mid\!\sim B\}$. Let $f_\sharp(A_1, \ldots, A_n) = \sharp(A_1, \ldots A_n)$ and let h be the assignment giving $h(q) = q$.

One can prove by induction on A that $h(A) = A$. $\lhd$

Theorem 9.7 Let $\mid\!\sim$ be an extensional consequence relation. Then it is complete for the class of all extensional algebras.

Proof. Immediate from the definitions and the canonical model of the previous theorem. $\lhd$

Definition 9.8 1. Let $\mathbf{L}$ be a directional logic with consequence $\mid\!\sim$ and connectives $\sharp_1, \ldots, \sharp_k$ with arities $r_1, \ldots, r_k$. Let $\mathbf{a} = (A, f_i, \leq, T)$ be a connected logic algebra with function symbols $f_i, i = 1, 2, \ldots$ with arities $r_1, r_2, \ldots$ resp (same as the connectives). Let $h^{\mathbf{a}}$ be a function (assignment) assigning for each atom q of $\mathbf{L}$ a filter $F_q \subseteq A$. Assume f_i are directional in the same way as $\sharp_i$. We can extend $h^{\mathbf{a}}$ to all wffs of $\mathbf{L}$ inductively as follows. Let $\sharp_j(A_1 \ldots, A_{n_j}, B_1, \ldots, B_{m_j})$ be a connective monotonic down in A_i and monotonic up in B_i. We let $h^{\mathbf{a}}(\sharp_j(A_1, \ldots, A_{n_j}, B_1, \ldots, B_{m_j})) = \{y \mid$ for all $x_1, \ldots, x_{n_j}$ such that $x_i \in h^{\mathbf{a}}(A_i)$ there exist y_k such that $y_k \in h^{\mathbf{a}}(B_k), i = 1, \ldots, n_j$ and $k = 1, \ldots, m_j$, we have $f_j(x_1, \ldots x_{n_j}, y_1, \ldots, y_{m_j}) \leq y\}$.

If a connective is monotonic both up and down in a slot we regard it as monotonic down. (We could regard it as montonic up—it does not matter.)

We show that the definition given here leads to the same assignment filter set no matter what we choose.

Let $\sharp(A, B, C)$ be a connective where the B slot is directional both ways, up and down.

We have two options

(a) Taking B as directional up:
$S_1 = h(\sharp(A, B, C) = \{y \mid$ for all x_1, x_2 such that $x_1 \in h(A), x_2 \in h(B)$, there exists a $x_3 \in h(C)$ such that $f_\sharp(x_1, x_2, x_3) \leq y\}$.

(b) Taking B as directional down:
$S_2 = h(\sharp(A, B, C)) = \{y \mid$ for all x_1 such that $x_1 \in h(A)$, there exists x_2, x_3 such that $x_2 \in h(B), x_3 \in h(C)$ such that $f_\sharp(x_1, x_2, x_3) \leq y\}$

We want to show that $S_1 = S_2$ under the assumption that the algebra is connected (as in Definition 9.0.5).

First note that clearly if $y \in S_1$ then $y \in S_2$. Assume $y \in S_2$. Show $y \in S_1$. Since $y \in S_2$ then for every $x_1 \in h(A)$, there are $x_2 = \alpha(x_1) \in h(B)$ and $x_3 = \beta(x_1) \in h(C)$ such that $f_\sharp(x_1, \alpha(x_1), \beta(x_1) \leq y$. We want to show that for every $x_1, x_2, x_1 \in h(A), x_1 \in h(B)$ there exists a $\gamma(x_1, x_2) \in h(C)$ such that $f_\sharp(x_1, x_2, \gamma(x_1, x_2)) \leq y$. We let $\gamma(x_1, x_2) = \beta(x_1)$. we must show that for all $x_1 \in h(A), x_2 \in h(B), f_\sharp(x_1, x_2, \beta(x_1)) \leq y$.

Since the second slot (x_2 slot in $f_\sharp$) is directional both up and down and since we assumed the algebra is connected, we get in view of the discussion in Example 9.0.4 (3) that

$$f_\sharp(x_1, x_2, \beta(x_1)) \leq f_\sharp(x_1, \alpha(x_1), \beta(x_1)) \leq y.$$

Thus we have shown that $S_1 = S_2$.

2. An algebraic model for $\mid\!\sim$ is a pair $(\mathbf{a}, h^{\mathbf{a}})$, as in 1. above. An algebraic semantics is a class of algebraic models.

3. Let $\mathbf{m} = (\mathbf{a}, h^{\mathbf{a}})$ be an algebraic model and let A, B, Δ be wffs, we write:

$$\begin{array}{rcl} A \mid\!\sim_{\mathbf{m}} B & \text{iff} & h^{\mathbf{a}}(A) \subseteq h^{\mathbf{a}}(B) \\ \mid\!\sim B & \text{iff} & h^{\mathbf{a}}(B) \subseteq T \\ \Delta \mid\!\sim_{\mathbf{m}} B & \text{iff} & \bigcap_{A \in \Delta} h^{\mathbf{a}}(A) \subseteq h^{\mathbf{a}}(B) \end{array}$$

4. Let $\mathcal{K}$ be a class of algebraic models. We say

   - $A \mid\!\sim_{\mathcal{K}} B$ iff $A \mid\!\sim_{\mathbf{m}} B$ for all $m \in \mathcal{K}$.
   - $\mid\!\sim_{\mathcal{K}} B$ iff $\mid\!\sim_{\mathbf{m}} B$ for all $\mathbf{m} \in \mathcal{K}$
   - $\Delta \mid\!\sim_{\mathcal{K}} B$ iff $\Delta \mid\!\sim_{\mathbf{m}} B$ for all $\mathbf{m} \in \mathcal{K}$.

5. Let $\mathbf{L}$ be a language and $\mid\!\sim$ a directional consequence relation. We define the canonical algebra $\mathbf{a}_{\mid\sim}$ as follows:

   - $A_{\mid\sim}$ = the set of all wffs of $\mathbf{L}$.
   - $f_\sharp$ for each connective $\sharp$ of $\mathbf{L}$ is defined by

   $$f_\sharp(A_1, \ldots, A_n) = \sharp(A_1, \ldots, A_n)$$

   - $A \leq B$ is defined as $B \mid\!\sim A$.
     T is $\{B \mid \varnothing \mid\!\sim B\}$.
   - Let $h^{\mathbf{a}}(q) = \{A \mid A \mid\!\sim q\}$. $h^{\mathbf{a}}(q)$ is a filter because if $A \in h^{\mathbf{a}}(q)$ (i.e. $A \mid\!\sim q$) and $A \leq B$ (i.e. $B \mid\!\sim A$) then certainly $B \in h^{\mathbf{a}}(q)$. Also $h^{\mathbf{a}}(q)$ is non empty (since $q \mid\!\sim q$).

Definition 9.9 [Term translation for a consequence relation] Let $\mathbf{L}$ and $\mid\!\sim$ be as in the previous Definition. Let A be a wff of $\mathbf{L}$ with atoms $q_1, \ldots, q_n$. Let $z_1, \ldots, z_n$ be variables of classical logic associated with $q_1, \ldots, q_n$. In fact, let q^* be the variable associated with q. With each connective $\sharp$ of $\mathbf{L}$ associate a function symbol $f_\sharp$ of the same arity. We associate with any formula A a term of classical logic A^* by structural induction as follows:

- $(q_i)^* = z_i$
- $(\sharp(A_1, \ldots, A_n))^* = f_\sharp(A_1^*, \ldots, A_n^*)$

Consider the classical language with the function symbols $\{f_\sharp\}$, the binary relation symbol $\leq$ and the unary predicate symbol T. Then the translation $* : A \mapsto A^*$, translates each formula A into a term in this language. The consequence assertion $A \mid\!\sim B$ is translated into $B^* \leq A^*$ and the assertion $\varnothing \mid\!\sim A$ is transated into $A^* \in T$.
The consequence relation rules of the form

$$\frac{A_i \mid\!\sim B_i, i = 1, \ldots, k}{\sharp_1(C_1, \ldots, C_{n_1}) \mid\!\sim \sharp_2(D_1, \ldots, D_{n_2})}$$

are transated into classical logic Horn clauses of the form

$$\bigwedge_{i=1}^{k} (B_i^* \leq A_i^*) \to f_{\sharp_2}(D_j^*) \leq f_{\sharp_1}(C_j^*)$$

where '$\rightarrow$' is classical implication.
For example, axioms schema such as $A \Rightarrow (B \Rightarrow A)$ become expressions like

$$f_{\Rightarrow}(x, f_{\Rightarrow}(y, x)) \in T$$

Modus ponens becomes

$$x \in T \wedge f_{\Rightarrow}(x, y) \in T \rightarrow y \in T.$$

Lemma 9.10 [Completeness] In the canonical model,

$$h^{\mathbf{a}}(B) = \{A \mid A \mathrel{|\!\sim} B\}.$$

Proof. By induction.

1. The lemma holds for atomic B.

2. Assume the lemma holds for $A_1, \ldots, A_n, B_1, \ldots, B_m$ we show the lemma holds for

   $$B = \sharp(A_1, \ldots, A_n, B_1, \ldots, B_m)$$

   We assume $\sharp$ is monotonic down in A_i and up in B_k.

   $$\begin{array}{c} h^{\mathbf{a}}(B) = \{y \mid \text{ for all } X_1, \ldots, X_n, \text{ such that } X_i \mathrel{|\!\sim} A_i \\ \text{there are } Y_1 \ldots Y_m, \text{ such that } Y_k \mathrel{|\!\sim} B_k \\ \text{we have } y \mathrel{|\!\sim} \sharp(X_1, \ldots, X_n, Y_1, \ldots, Y_m)\}. \end{array}$$

   We want to show

   $$h^{\mathbf{a}}(B) = \{y \mid y \mathrel{|\!\sim} \sharp(A_1, \ldots, B_m)\}$$

   Assume that $y \in h^{\mathbf{a}}(B)$ and show that $y \mathrel{|\!\sim} B$. From the assumptions we see that for $X_i = A_i$, there are $Y_i \mathrel{|\!\sim} B_i$ such that

   $$y \mathrel{|\!\sim} \sharp(A_i, Y_i)$$

   Since $\sharp$ is monotonic up in Y_i we get $y \mathrel{|\!\sim} B$.

   Assume $y \mathrel{|\!\sim} B$, we show $y \in h^{\mathbf{a}}(B)$. Let X_i be such $X_i \mathrel{|\!\sim} A_i$, we need to find Y_k such that $y \mathrel{|\!\sim} \sharp(X_i, Y_k)$. Take $Y_k = B_k$. We need to show $y \mathrel{|\!\sim} \sharp(X_i, B_k)$. Since $\mathrel{|\!\sim}$ is monotonic down in A_i, we get that $X_i \mathrel{|\!\sim} A_i$ yield $\sharp(A_i, B_k) \mathrel{|\!\sim} \sharp(X_i, B_k)$ and hence $y \mathrel{|\!\sim} \sharp(X_i, B_k)$, by transitivity

This completes the proof of the lemma. Note that we proved that each filter is generated by a minimal element. $\triangleleft$

Example 9.11 Let $X \twoheadrightarrow Y$ be a binary connective. Let $\mathrel{|\!\sim}$ be the smallest consequence relation which makes $\twoheadrightarrow$ directional down in X and up in Y,

$$\frac{X \mathrel{|\!\sim} X'}{X' \twoheadrightarrow Y \mathrel{|\!\sim} X \twoheadrightarrow Y} \qquad \frac{Y \mathrel{|\!\sim} Y'}{X \twoheadrightarrow Y \mathrel{|\!\sim} X \twoheadrightarrow Y'}$$

and the two standard conditions

- $$A \mathrel{|\!\sim} A$$

- $$\frac{A \mathrel{|\!\sim} B; B \mathrel{|\!\sim} C}{A \mathrel{|\!\sim} C}$$

The semantics for this system involve algebras of the form $\mathbf{a} = (A, f_{\twoheadrightarrow}(x,y), \leq, T, h)$ with the truth condition on $\twoheadrightarrow$ being:
$t \vDash A \twoheadrightarrow B$ iff for all $x \vDash A$ there exists a $y \vDash B$ such that $f_{\twoheadrightarrow}(x,y) \leq t$.
We have $A \mid\!\sim B$ iff for all $\mathbf{a}$ and all $t \in A^{\mathbf{a}}, t \vDash A$ implies $t \vDash B$.
This semantics should be compared with the more traditional semigroup semantics for substructural implications. A model has the form $(A, *, \leq, h)$ where $(\mathcal{A}, *)$ is a semigroup (associative?) and $\leq$ an ordering. The clause for $\twoheadrightarrow$ is

$$t \vDash A \twoheadrightarrow B \text{ iff } \forall x \vDash A, t * x \vDash B.$$

The two semantics will coincide if we take $\leq$ as identity and let

$$f_{\twoheadrightarrow}(x,y) = (\text{The } z \text{ such that})(z * x = y).$$

The previous definition of canonical model and the completeness theorem allow us to give a semi algebraic possible world semantics for directional logics. The following definition establishes the connection.

Remark 9.12 [A more general completeness theorem] The completeness theorem for Kripke like semantics can be generalised. We need not assume the consequence relation be transitive. The following conditions are sufficient:

- $A \mid\!\sim A$ identity
- Directionality of all connectives (as in 9.0.3)
- Downward strengthening: Let $\sharp(q, \ldots)$ be any connective which is downward monotonic in q, then for any A and B

$$\frac{A \mid\!\sim \sharp(q, \ldots); B \mid\!\sim q}{A \mid\!\sim \sharp(B, \ldots)}$$

Let $\mathbf{a}$ be any algebra satisfying reflexivity, directionality and downward strengthening. Let $t \in A^{\mathbf{a}}$, define a semi-filter I_t to be the *largest* set Y satisfying $(*)$ below

$$(*) \qquad \forall s \in A[s \leq t \text{ iff } \forall x \in Y(s \leq x)]$$

clearly $I_t^0 = \{t\}$ is such a set and the family of such sets is closed under union.
Let h be an assignment. Require h to satisfy:

$$(**) \qquad h(t,q) = 1 \text{ iff } (\forall s \in I_t) h(s,q) = 1.$$

Then in the canonical model, we have for such h and any A

$$t \vDash A \text{ iff } t \mid\!\sim A.$$

The proof is similar to that of Lemma 9.0.10. Downward stengthening plays the role of transitivity in the proof. In fact, if $\mid\!\sim$ is transitive then downward strengthening follows and each semifilter I_t is equal to $\{s \mid t \leq s\}$.

Definition 9.13 [Semi algebraic possible world semantics] Let $\mathbf{L}$ be a language with connectives $\sharp_1, \ldots, \sharp_k$ with arities $r_1, \ldots, r_k$ respectively. A semi algebraic model for $\mathbf{L}$ has the form $(A, R_1, \ldots, R_k, \leq, T, h)$ where A is a set of possible worlds, $T \subseteq A, T \neq \varnothing$ and each R is a relation on A of arity $1 + r_i$. h is an assignment from the atoms of the language into subsets of A. The following must hold.

1. $x \leq y$ and $y \in T$ imply $x \in T$.
2. $t \in h(q)$ and $t \leq s$ imply $s \in h(q)$.

3. Each R_i is directional in all its variables, in the same way as $\sharp_i$, respectively, and R_i is directional up in its first variable.

Note that for a connective $\sharp(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ there corresponds a relation $R_\sharp(t, x_1, \ldots, x_n, y_1, \ldots, y_m)$. The meaning of $R_\sharp$ in terms of the algebraic function $f_\sharp(x_1, \ldots, x_n, y_1, \ldots, y_m)$ and $\leq$ is

$$R_\sharp(t, x_1, \ldots, x_n, y_1, \ldots, y_m) \text{ iff } f_\sharp(x_1, \ldots, x_n, y_1, \ldots, y_m) \leq t.$$

Note that $R_\sharp$ satisfies that for each $(x_1, \ldots, x_n, y_1, \ldots, y_m)$ there exists a minimal t such that $R_\sharp(t, x_1, \ldots, x_n, y_1, \ldots, y_m)$ holds. Conversely, the function $f_\sharp$ can be retrieved from such an $R_\sharp$ in the presence of $\leq$.
Thus if $f_\sharp$ is monotonic up in y_j and monotonic down in x_i, then $R_\sharp$ will be monotonic up in x_i and t and monotonic down for y_j (inverse direction for x_i and y_j).
Define the following first truth table for the connectives $\sharp(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ as follows: ($\sharp$ is directional down in X_i and up in Y_i):
$t \vDash \sharp(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ iff for all t_i such that $t_i \vDash X_i$ there exist s_j such that $s_j \vDash Y_j$ such that $R_i(t, x_1, \ldots, x_n, y_1, \ldots, y_m)$ holds.
We say $A \mid\!\sim B$ holds in the model if whenever $t \vDash A$ then $t \vDash B$.
We say $\vDash A$ in the model if whenever $t \vDash A$ then $t \in T$.
Note that we can also assume that $R_\sharp$ satisfies that for every $t, x_1, \ldots, x_n$ there exist unique $y_1, \ldots y_m$ such that $R_\sharp(t, x_1, \ldots, x_n, y_1, \ldots, y_m)$. The reason being that the filters $\{y \mid y \vDash Y\}$ have a minimal element and $R_\sharp$ is monotonic down in y_i. So $R_\sharp$ holds for some y iff it holds for the minimal y. This last statement entails that we can get completeness for $R_\sharp$ and the following truth table:
$t \vDash \sharp(X_1, \ldots, X_n, Y_1, \ldots Y_m)$ iff for all t_i such that $t_i \vDash X_i$ and all s_j such that $R_\sharp(t_1, \ldots t_n, s_1, \ldots, s_m)$ we have for all $j, s_j \vDash Y_j$.

Theorem 9.14 Let $\mid\!\sim$ be the smallest directional consequence relation, then $\mid\!\sim$ is complete for the semi algebraic semantics.

Proof. Use the canonical model ◁

So far we have been mainly concerned with algebraic theorems preparing an almost arbitrary logic for translation into classical logic, not with the theory of partially ordered algebraic logics (cf. [Gabbay, 1993b, Gabbay, 1992b, Blok and Pigozzi, 1989]). We need the algebras as tools in order to translate any general logic, about which we do not know much. We found it convenient to apply our algebraic methods to singleton consequence relations $\mid\!\sim$, i.e. the kind defined for $A \mid\!\sim B$ and $\varnothing \mid\!\sim B$, which give rise to the ordering relation $\leq$ in the logic algebra. Many logics, however, are presented as Hilbert systems, and so we cannot apply our methods as we have only the relation $\varnothing \mid\!\sim_1 A$, for A wff. Suppose we are given a Hilbert presentation of a logic $\mathbf{L}_1$. This means that we are defining only the notion of $\varnothing \mid\!\sim_1 A$ (or just $\mid\!\sim_1 A$). We need to investigate a means of deriving from $\mid\!\sim_1$ a consequence relation $\mid\!\sim_2$ such that $A \mid\!\sim_2 B$ is always defined and $\mid\!\sim_2$ is a conservative extension, i.e. $\varnothing \mid\!\sim_2 A$ iff $\varnothing \mid\!\sim_1 A$ holds. In algebraic terms, given (A, f_i, T) can we define a suitable $\leq$ on A? In implicational logics with $\Rightarrow$ satisfying the deduction theorem this can be easily done: $A \mid\!\sim_2 B$ iff $\varnothing \mid\!\sim_1 A \Rightarrow B$.

Lemma 9.15 Let $\mid\!\sim_1$ be a Hilbert conseuence relation (i.e. defined for $\varnothing \mid\!\sim_1 A, A$ wff). Then there exists the minimal $\mid\!\sim_-$ and maximal $\mid\!\sim_+$ singleton consequence relations which agree with it.

Proof.

1. Let $\Delta \mid\!\sim_- B$ be defined by

$$\begin{array}{rl} \Delta \mid\!\sim_- B \text{ iff} & (1)\Delta = \varnothing \text{ and } \varnothing \mid\!\sim B \\ \text{or} & (2)\Delta = \{B\}. \end{array}$$

We show $\mid\!\sim_-$ is a consequence relation:

- clearly $A\mid\!\sim_{\_} A$ holds
- assume $\Delta\mid\!\sim_{\_} A$ and $A\mid\!\sim_{\_} B$. By definition $A = B$ and so $\Delta\mid\!\sim_{\_} B$.

We show $\mid\!\sim_{\_}$ is minimal. Suppose $\mid\!\sim_1$ agrees with $\mid\!\sim$. We show $\Delta\mid\!\sim_{\_} A$ implies $\Delta\mid\!\sim_1 A$. If $\Delta = \{A\}$, then clearly $A\mid\!\sim_1 A$. If $\Delta = \varnothing$, again by agreement $\varnothing\mid\!\sim_1 A$.

2. From (1) it is clear that the set of all singleton consequence relations agreeing with $\mid\!\sim$ is nonempty. Define $\mid\!\sim_+$ as follows.

   $A\mid\!\sim_+ B$ iff there exist $\mid\!\sim_0, \ldots \mid\!\sim_k$ agreeing with $\mid\!\sim$ and $A_1, \ldots, A_k$ such that

   $$A\mid\!\sim_0 A_1, A_1\mid\!\sim_1 A_2, \ldots, A_k\mid\!\sim_k B.$$

   We show that $\mid\!\sim_+$ is a consequence relation:

   - clearly $A\mid\!\sim_+ A$ because $A\mid\!\sim_{\_} A$ holds
   - assume $\Delta\mid\!\sim_+ B$ and $B\mid\!\sim_+ C$ we show $\Delta\mid\!\sim_+ C$. Then for some $\mid\!\sim_0, \ldots, \mid\!\sim_k$ and $\mid\!\sim_{k+1}, \ldots, \mid\!\sim_{m+1}$ and $A_1, \ldots, A_k, A_{k+1}, \ldots, A_m$ we have

     $$\Delta\mid\!\sim_0 A_1, \ldots, A_k\mid\!\sim_k B, B\mid\!\sim_{k+1} A_{k+1}, \ldots, A_m\mid\!\sim_{m+1} C.$$

     But then by definition $\Delta\mid\!\sim_+ C$.

   We now show $\mid\!\sim_+$ agrees with $\mid\!\sim$

   - Clearly $\varnothing\mid\!\sim B$ implies $\varnothing\mid\!\sim_+ B$.
   - assume $\varnothing\mid\!\sim_+ B$ and show $\varnothing\mid\!\sim B$

   Since $\varnothing\mid\!\sim_+ B$, there exist $A_1, \ldots, A_k$ and $\mid\!\sim_0, \ldots, \mid\!\sim_k$ such that $\varnothing\mid\!\sim_0 A_1, \ldots, A_k\mid\!\sim_k B$ hold. We prove $\varnothing\mid\!\sim B$ by induction on k.

Case $k = 0$
In this case $\varnothing\mid\!\sim_0 B$ and since $\mid\!\sim_0$ agrees with $\mid\!\sim$ we get $\varnothing\mid\!\sim B$.
Case $k > 0$
Consider the initial sequence

$$\varnothing\mid\!\sim_0 A_1 \text{ and } A_1\mid\!\sim_1 A_2.$$

Since $\mid\!\sim_0$ agrees with $\mid\!\sim$ which agrees with $\mid\!\sim_1$, we get $\varnothing\mid\!\sim_1 A_1$ and hence by transitivity, we get $\varnothing\mid\!\sim_1 A_2$

We now have a shorter sequence

$$\varnothing\mid\!\sim_1 A_2, A_1\mid\!\sim_2 A_3, \ldots, A_k\mid\!\sim_k B$$

Hence, by the induction hypothesis $\varnothing\mid\!\sim B$.
This completes the induction. Clearly $\mid\!\sim_+$ is maximal. Hence, the Lemma is proved. $\lhd$

The above considerations do not give us an algorithm for finding for a given Hilbert system $\mid\!\sim_1$ a singleton consequence relation $\mid\!\sim_2$ which agrees with it. Of course, we can always take the minimal one $\mid\!\sim_1^-$, but the minimal one is not particularly infomative. The stronger the consequence relation the better. One can try to find one using a theorem prover as the next example shows.

Example 9.16 Let $\mathbf{L}$ be a propositional language with some connectives and let $\mathbf{H}$ be a Hilbert system for $\mathbf{L}$ with schematic rules of the form $\rho_i, (i = 1, \ldots, m)$:

$$\rho_i : \quad \frac{\mid\!\sim\varphi_1^i; \ldots; \mid\!\sim\varphi_{k_i}^i}{\vdash \varphi_i}$$

For example, Łukasiewicz formulation of the implicational fragment of classical logic (using the binary connective $\Rightarrow$) is formulated by the schemas

$$\rho_i : \quad \frac{\mid\!\sim P; \mid\!\sim P \Rightarrow Q}{\mid\!\sim Q}$$

$$\rho_2 : \quad \frac{\varnothing}{\mid\!\sim((P \Rightarrow Q) \Rightarrow R) \Rightarrow ((R \Rightarrow P) \Rightarrow (S \Rightarrow P))}$$

We can systematically look (by complexity) for a formula $\Psi(X, Y)$ (binary) and try and prove using the term translation of Definition 9.0.9 and a classical theorem prover, that the following holds:

- $\mid\!\sim_1 \Psi(X, X)$
- $\mid\!\sim_1 \Psi(X, Y)$ and $\mid\!\sim_1 \Psi(Y, Z)$ imply $\mid\!\sim_1 \Psi(X, Z)$
- $\mid\!\sim_1 X$ and $\mid\!\sim_1 \Psi(X, Y)$ imply $\mid\!\sim_1 Y$.

For a Ψ satisfying the above we can let

$$A \mid\!\sim_2 B \text{ iff (definition) } \mid\!\sim_1 \Psi(A, B)$$

$$\varnothing \mid\!\sim_2 B \text{ iff } \mid\!\sim_1 B.$$

In fact, since singleton consequence relations are closed under intersection, if we have several such Ψ_i we can let $A \mid\!\sim_2 B$ be $\bigwedge_i \mid\!\sim_1 \Psi(A, B)$.
We now ask how do we find such a Ψ automatically? We can let Ψ range over all wffs of the language in order of increasing complexity and then for this Ψ, using the term translation of 9.0.9 and a classical theorem prover see if each Ψ satisfies the required condition.
A good heuristic is to look at the Hilbert rules of the form

$$\frac{\mid\!\sim_1 \varphi_1; \ldots; \mid\!\sim_1 \varphi_k}{\mid\!\sim_1 \varphi}$$

and try and substitute into φ_i and φ some formula that will turn the above rule into the form

$$\frac{\mid\!\sim_1 A; \mid\!\sim_1 \Psi_1(A, B); \ldots \mid\!\sim_1 \Psi_m(A, B)}{\mid\!\sim_1 B}$$

We can now check whether the metapredicate

$$\bigwedge_{i=1}^{m} [\mid\!\sim_1 \Psi_1(X, Y]$$

is transitive, in which case, we define

$$A \mid\!\sim_2 B \text{ iff } \bigwedge_{i=1}^{m} \mid\!\sim_1 \Psi_i(A, B)$$

$$\varnothing \mid\!\sim_2 B \text{ iff } \mid\!\sim_1 B.$$

Applying the above procedure to Łukasiewicz logic, we immediately find that $\Psi(A) = (\text{def}) A \Rightarrow B$ is a candidate. We need to check whether

- $\mid\!\sim_1 A \Rightarrow B$ and $\mid\!\sim_1 B \Rightarrow C$ imply $\mid\!\sim_1 A \Rightarrow C$
- $\mid\!\sim_1 A \Rightarrow A$.

These can be checked using the term translation.

Example 9.17 The previous example dealt with the problem of finding a singleton consequence relation $\Delta|\!\sim_2 B$ ($\Delta = \{B\}$ or $\Delta = \varnothing\}$ agreeing with a Hilbert consequence $\varnothing|\!\sim_1 B$. We can continue this line of thought and try to extend a singelton consquence $\Delta|\!\sim_2 B$ to a unitary consequence $\Delta|\!\sim_3 B$, Δ, arbitrary which agrees with it.
Assume now that a proper $|\!\sim$ is given, where $A|\!\sim B$ andn $\varnothing|\!\sim B$ are defined for arbitrary A and B.
The second concept we define is that of a theory. We defined a theory Δ as a set of wffs and gave the notion $\Delta \mid\!\sim A$ the semantic meaning:
$\Delta \mid\!\sim A$ iff for all $\mathbf{a}$ and for all $t \in A^{\mathbf{a}}$, if $t \vDash B$, for all $B \in \Delta$ then $t \vDash A$.
There is a more general notion of a theory. Let $\Psi(X, Y)$ be formulas defined using the connectives of the logic. Assume Ψ is monotonic down in X and up in Y. Using Ψ we can define the following notion of a theory Δ.

- A theory Δ is a sequence of formulas $(A_1, \ldots, A_n)$
- $(A_1, \ldots, A_n) \mid\!\sim A =$ def $\varnothing \mid\!\sim \Psi(A_1, \Psi(A_2, \ldots, \Psi(A_n, A)\ldots))$

This definition gives the deduction theorem for $|\!\sim$.

$$A_1, \ldots, A_n \mid\!\sim \Psi(A, B) \text{ iff } A_1, \ldots, A_n, A \mid\!\sim B$$

Consider for example the connective $\twoheadrightarrow$ of example 9.0.11. We have

$$A_1, \ldots, A_n \mid\!\sim B \text{ iff } \varnothing \mid\!\sim A_1 \twoheadrightarrow (\ldots, \twoheadrightarrow (A_n \twoheadrightarrow B)\ldots)$$

The notion of a theory can be generalised. Let $\tau = \{\sharp_1, \ldots, \sharp_k\}$ be a set of monotonic upwards connectives of the language. Close τ under substitution. Let $\varphi(x_1, \ldots, x_n)$ be a wff generated from τ with n-places. Then the formula expression $[\varphi(A_1, \ldots, A_n)]$ is a φ-theory of $(A_1, \ldots, A_n)$ with $\varphi[A_1, \ldots, A_n] \mid\!\sim B$ iff $\varnothing \mid\!\sim \Psi(\varphi(A_1, \ldots, A_n), B)$.
Other definitions of algebraic logics and theories existing in the literature do not use the partial order but only the set of designated elements. One example is ($\mathbf{a}$ an algebraic model): $\Delta \vDash_{\mathbf{a}} A$ iff for every assignment h, if $h(B) \in T$ for all $B \in \Delta$ then $h(A) \in T$.

We can now translate, through the semantics, any directional logic into classical logic. This is the task of the next Section.

10 An automated universal translator into classical logic

The aim of this section is to show an almost algorithmic way of translating an arbitrary logic into classical logic. The method is based on results of previous sections and on an algorithm called SCAN for eliminating second order existential quantifers in classical logic. The SCAN algorithm and examples will be given later in this section.

10.1 The translation steps

Let us now outline the steps in our automatic translation.
Step 1: Presentation
We assume a logic is given as a consequence relation, satisfying rules of the form:

$$\frac{A_i \mid\!\sim B_i,\ i = 1, \ldots, n}{C \mid\!\sim D}$$

We assume $|\!\sim$ is finitely schematically generated. This means that the set of rules can be recursively presented (generated) on a computer. We assume we do not have fancy quantifiers but possibly $\forall$ and $\exists$. So the main features of the logic come from its connectives.

Such a presentation is still very general.

Any Hilbert presentation has the form

1.

$$\frac{\varnothing \mid\!\sim A_1, \ldots, \varnothing \mid\!\sim A_n}{\varnothing \mid\!\sim B}$$

2.

$$\frac{\varnothing}{\varnothing \mid\!\sim \text{ Axioms}}$$

Any Gentzen system can be regarded as a recursive inductive definition of $\mid\!\sim$. Our subsequent algebraic analysis and translations depend on the availability of the consequence relation $\mid\!\sim$ for the logic involved. If the logic, say $\mathbf{L}_1$, is presented as a Hilbert system, then we have available only the fragment of $\mid\!\sim$ for the form $\varnothing \mid\!\sim_1 A$. In algebraic terms this means that instead of $(A, f_i, \leq, T_1)$ we have only (A, f_i, T_1), (since $A \in T$ means $\varnothing \mid\!\sim_1 A$). We now have two options. The first is to present methodologies for defining some useful $\leq$ which agrees with T_1 (i.e. extend the provability of the Hilbert system into a proper consequence relation, usually done by some form of a deduction theorem. The second is to apply our algebraic analysis and translations directly to the Hilbert consequence (i.e. consequences of the form $\varnothing \mid\!\sim A$).

We shall opt for the first approach because it also shows how to find consequence relations (of the form $A \mid\!\sim B$) corresponding to Hilbert consequences. Examples 9.0.16 ad 9.0.17 show how one might find such a consequence.

From now on we assume we are given a proper consequence $A \mid\!\sim B$.

Step 2: Algebraic analysis

We need to find out whether $\mid\!\sim$ is directional or not in all of its connectives. This is essential to allow us to define a possible world semantics for $\mid\!\sim$, using definition 9.0.13. In fact, we do not necessarily need to show that the official connectives of $\mid\!\sim$ are directional. It is sufficient to show that there exists another set of connectives, definable in $\mid\!\sim$ and capable of redefining the original connectives of $\mid\!\sim$, and this other set is directional. This will give semantics for the connectives of $\mid\!\sim$. We need to solve the following problem.

Problem 1

(a) Find an algorithm which for a given $\mid\!\sim$ with connectives $\{\sharp_i\}$, can determine the existence of a mutually definable set of directional connectives $\{\sharp'_j\}$ and exhibit it.

(b) If $\mid\!\sim$ is a Hilbert consequence, i.e. is known (or is given) only for the form $\varnothing \mid\!\sim B$, then we need an algorithm which can produce a conservative extension of $\mid\!\sim$ also to the form $A \mid\!\sim B$.

(c) Find an algorithm which can check whether $\{\sharp_i\}$ themselves are directional.

Problem 1(a) is open. Problem 1(b) can be overcome by methods indicated in Example 9.0.16. Problem 1(c) can be checked by a resolution machine using the term translation (of definition 9.0.9) on the propositional part. We have assumed that there are no fancy quantifiers in our logic.

Step 3 Translation

1. Assuming $\mid\!\sim$ is directional, we can give it semantics as in 9.0.13 with relations $R_\sharp(t, x_1, \ldots, x_n, y_1, \ldots, y_m)$ associated with each connective $\sharp$ and the truth condition $t \vDash \sharp(A_1, \ldots, A_n, B_1, \ldots, B_m)$ iff $\forall t_1, \ldots, t_n, s_1, \ldots, s_m (\bigwedge_{i=1}^n t_i \vDash A_i$ and $R_\sharp(t, t_1, \ldots, t_n, s_1, \ldots s_m)$ imply $\bigwedge_{j=1}^m s_j \vDash B_j)$.

2. Using the translations of definition 10.1.1 and Remark 10.1.2 below, we construct the following reduction into classical logic:

   $A \mid\!\sim B$ holds iff $\vdash_{\text{classical}} \Psi_{A,B}$, where $\Psi_{A,B}$ is a second order formula defined in the definitions and remark below.

SCAN Algorithm

We need an algorithm which will eliminate (if possible) the second order quantifiers in $\Psi_{A,B}$ and yield an equivalent first-order formula τ_Ψ. Such an algorithm may not always terminate. We thus need to solve the following problem.

Problem 2
Let $\psi(Q_1, \ldots, Q_n)$ be a first-order formula with predicates $Q_1, \ldots, Q_n$, among other predicates. Consider the second-order formula

$$\Psi = (\forall Q_1, \ldots, Q_n)\psi(Q_1, \ldots, Q_n).$$

Ψ is a second-order formula which is no longer dependent on $Q_1, \ldots, Q_n$. find (if possible) a first order formula B (not containing Q_i but maybe containing the rest of the predicates from ψ) such

$$\vdash B \leftrightarrow \Psi$$

in classical logic.

If we solve problem 1 and problem 2 we will have a completely automatic way of translating an arbitrary logic $\mid\sim$ (given to us recursively via its consequence relation) into first-order classical logic. Of course, the algorithm may not always terminate.

We now proceed to present the translation as promised in Step 3.

Definition 10.1 [Translation into second-order classical logic]

1. Let $\mid\sim$ be a directional logic, with connectives $\Rightarrow_1, \ldots, \Rightarrow_k$ presented through rules as in 9.0.3. We display the connective $\Rightarrow_i$ as

   $$(X_1, \ldots, X_{n_i}) \Rightarrow_i (Y_1, \ldots, Y_{m_i})$$

   where $\Rightarrow_i$ is directional down in the Xs and directional up in the Ys.

2. Consider a linked predicate language $(\mathbf{G}, \mathbf{L})$, where $\mathbf{G}$ contains function symbols $f_i(x_1, \ldots, x_{n_i}, y_1, \ldots, y_{m_i}), i = 1, \ldots k$ and the binary relation $\leq$ and $\mathbf{L}$ is ordinary predicate logic. Consider $\mathbf{L}_1^*(\mathbf{G})$. This means that we turn $\mathbf{L}$ into a two sorted language $\mathbf{L}^*$ associating with each atomic $Q(x_1, \ldots, x_n)$ a predicate $Q^*(t, x_1, \ldots, x_n)$, where t ranges over terms of $\mathbf{G}$.

We now translate every wff A of $\mid\sim$ into a formula $[A]^*(t)$ with free variable t of sort G as follows:

1. $[Q(x_1, \ldots, x_n)]^*(t) = \text{def} Q^*(t, x_1, \ldots, x_n), Q$ atomic.
2. $[(A_1, \ldots, A_{n_i}) \Rightarrow_i (B_1, \ldots, B_{m_i})]^*(t) = \text{def}$
   $\forall t_1, \ldots, t_{n_i}[\bigwedge_{j=1}^{n_i}[A_j]^*(t_j) \rightarrow \exists s_1, \ldots, s_{m_i}(f_j(t_1, \ldots, t_{n_i}, s_1, \ldots, s_{m_i}) \leq t \wedge \bigwedge_{j=1}^{m_i}[B_j]^*(s_j))]$
3. The theory τ of second-order classical logic which ensures the translation is correct is a conjunction of the following
   (a) $\forall t, s, u[t \leq s \wedge s \leq u \rightarrow t \leq u) \wedge \forall t(t \leq t)]$.
   (b) $\forall Q^* \forall t, s[Q^*(t) \wedge t \leq s \rightarrow Q^*(s)]$.
   (c) For each i and variables u_1, u_2

   $$\forall u_1, u_2[u_1 \leq u_2 \rightarrow f_i(\ldots u_1 \ldots) \leq f_i(\ldots u_2 \ldots)]$$

   If $f_i(\ldots u \ldots)$ is monotonic up in the variable u and otherwise replace $u_1 \leq u_2$ by $u_2 \leq u_1$.
   (d) For each rule of the form

   $$\frac{A_1 \mid\sim B_1, \ldots, A_r \mid\sim B_r}{C \mid\sim D}$$

   displaying the atomic formulas $Q_1, \ldots, Q_n, \tau$ contains the second-order formula:

   $$(\forall Q_1^* \ldots Q_n^*)[(\bigwedge_{j=1}^{r} \forall t \in T([B_j]^*(t) \rightarrow [A_j]^*(t))) \rightarrow \forall t \in T([D]^*(t) \rightarrow [C]^*(t))]$$

4. We have in second-order classical logic that $A \mid\sim B$ iff the following second-order formula $\Psi_{A,B}$ is a theorem of second-order classical logic.

$$\forall T(\forall t, s[s \in T \wedge t \leq s \rightarrow t \in T] \wedge \tau \rightarrow \forall t \in T([B]^*(t) \rightarrow [A]^*(t)))$$

5. Some researchers translate rules of the form (d) above using a *Hold* predicate, namely (d) is translated into

$$Hold(A_1, B_1) \wedge \ldots \wedge Hold(A_r, B_r) \rightarrow Hold(C, D)$$

where $\rightarrow$ is classical implication and the formulas appearing in *Hold* are representing themselves as terms. This translation is really an **HFP** translation. Compared with the translation in 4 above, '*Hold*' is really '$<$' and τ will be represneted as meta-axioms on the predicate *Hold*. $T(x)$ can be represented as $Hold(\top, x)$.

We shall in Section 10.2 give an algorithm which in some cases reduces the second-order τ into a first order one.

Remark 10.2 If we use the relational $R_\sharp$ semantics, with the second truth table of 9.0.13 then **G** contains R_i instead of f_i and the translation of $\Rightarrow_i$ becomes:

$$\begin{gathered}[(A_1, \ldots, A_{n_i}) \Rightarrow_i (B_1, \ldots, B_{m_i})]^*(t) = \text{def} \\ \forall t_1, \ldots, t_{n_i}, s_1, \ldots, s_{m_i}[\bigwedge_{j=1}^{n_i}[A_j]^*(t_j) \wedge R_{\Rightarrow_i}(t, t_1, \ldots, t_{n_i}, s_1, \ldots, s_{m_i}) \\ \rightarrow \bigwedge_{j=1}^{m_i}[B_j]^*(s_j)]\end{gathered}$$

Condition (c) in the definition of τ stipulates the monotonicity of $R_{\Rightarrow_i}$.

Example 10.3 Consider the smallest directional logic for binary $X \twoheadrightarrow Y$ which is directional down in X and up in Y, with the axiom $A \twoheadrightarrow A$ only.
This axiom has the form of the rule

$$\frac{\varnothing}{\varnothing \mid\sim A \twoheadrightarrow A}.$$

Its translation to second-order logic becomes (using 9.0.16)

1. $\forall A^* \forall t \in T \forall x \forall y[A^*(x) \wedge R_{\twoheadrightarrow}(t, x, y) \rightarrow A^*(y)]$
   The other properties available are
2. $\forall x, y[y \in T \wedge x \leq y \rightarrow x \in T]$
3. $\forall A^* \forall x, y[A^*(x) \wedge x \leq y \rightarrow A^*(y)]$
4. $R_\rightarrow(t, x, y)$ is monotonic up in t and x and down in y.

First note that (3) is not really second-order. If it holds for atoms it holds for the translation of any wff. It is therefore equivalent to a first-order theory of its instances for atoms.
We now ask can we replace the second-order (1) by a first-order condition?
The answer is that we can impose further first-order conditions on $R_{\twoheadrightarrow}$ which will be equivalnet to the second-order (1). This is

(1*) $\forall t, x, y[R_{\twoheadrightarrow}(t, x, y) \rightarrow x \leq y]$.

The next subsection gives an algorithm which can find such first-order equivalents in many cases.

10.2 The SCAN algorithm

We saw in the previous subsection that any directional logic $\mid\!\sim$ can be translated into second-order logic through its semi-algebraic semantics. To reduce the translation to first order logic, we need to solve Problem 2 mentioned above. This subsection shows how to achieve this end.

An algorithm is presented which eliminates second-order quantifiers over predicate variables in formulae of type $\exists P_1, \ldots, P_n \psi$ where ψ is an arbitrary formula of first–order predicate logic. The resulting formula is equivalent to the original formula – if the algorithm terminates. The algorithm can for example be applied to do interpolation, to eliminate the second-order quantifiers in circumscription, to compute the correlations between structures and power structures, to compute semantic properties corresponding to axioms in non-classical logics and to compute model theoretic semantics for new logics.

Several methods have been developed for computing from a given second–order formula an equivalent first–order formula. These methods basically fall into two classes. The first class of algorithms computes or guesses suitable instantiations for the second–order predicate variables that are guaranteed to preserve equivalence [Ackermann, 1935a, van Benthem, 1984, Szałas, 1992, Simmons, 1993]. The idea of the second class of algorithms is to compute sufficiently many consequences from the formulae containing the second–order variables and then keeping from the resulting set of formulae only those without the second–order variables [Ackermann, 1935a, Bachmair *et al.*, 1992]. The algorithm we are going to present falls into this second class.

The structure of the formulae our algorithm can handle is $\exists P_1, \ldots, P_n \psi$ where the P_i are predicate variables for n–place predicates and ψ is an arbitrary first–order formula. Our algorithm essentially normalizes ψ into clause form and generates all (constraint–) resolvents of the clauses with the predicates P_i. It is shown that the subset of the generated clauses not containing predicates P_i (which may be infinite) is equivalent to the original formula. Since $\forall P_1, \ldots, P_n \psi \leftrightarrow \neg\exists P_1, \ldots, P_n \neg\psi$ ($\leftrightarrow$ is the equivalence sign), the algorithm can of course also handle universal quantifiers by reducing this case to the case with existential quantifiers. This algorithm is simple and it can be realized easily with existing theorem provers, for example OTTER.

Let us illustrate the SCAN algorithm with some simple examples.
It is easy to see that

$$\exists P \begin{pmatrix} P \vee Q \\ \neg P \vee R \end{pmatrix} \begin{array}{l} \text{is logically} \\ \text{equivalent to} \end{array} Q \vee R$$

where $Q \vee R$ is just the resolvent between the two clauses on the left hand side. The left-to-right-direction of the above equivalance follows from the fact that $Q \vee R$ as a resolvent is of course implied by the original formula. To see the other direction, suppose Q is true in an interpretation. In this case the assigment for P, which is existentially quantified, can be chosen to be true also, making the existentially quantified formula true as a whole. If instead, R is true then P must be chosen to be false and again the left hand side formula is true. Thus, the existentially quantified P can be eliminated by just taking the single resolvent with P.

A slightly more complex example illustrates that in fact *all* (not redundant) resolvents with the second–order predicate have to be generated.

$$\exists P \begin{pmatrix} P \vee Q \\ \neg P \vee R \\ \neg P \vee S \end{pmatrix} \begin{array}{l} \text{is logically} \\ \text{equivalent to} \end{array} \begin{pmatrix} Q \vee R \\ Q \vee S \end{pmatrix}$$

If Q is false in a model, it is necessary that R and S are both true in order to choose P such that all three clauses on the left hand side become true. Falsity of Q enforces truth of both R and S on the right hand side only if both resolvents are present.

In the presence of second–order predicates with arguments, the resolution rule has to be changed slightly, as the third example demonstrates.

$$\exists P \begin{pmatrix} P(a) \vee Q \\ \neg P(b) \vee R \end{pmatrix} \begin{array}{l} \text{is logically} \\ \text{equivalent to} \end{array} a = b \Rightarrow (Q \vee R)$$

Only in models of the right hand side where a and b are mapped to the same objects, it is necessary that one of Q or R must be true in order to satisfy the left hand side. If a and b are

interpreted differently, we may well choose both $P(a)$ and $\neg P(b)$ to be true. That means instead of unification, just a constraint for the arguments of the resolution literals has to be generated.

Definition 10.4 [The SCAN Algorithm] Input to SCAN is a formula $\alpha = \exists P_1, \ldots, P_n\ \psi$ with predicate variables $P_1, \ldots, P_n$ and an arbitrary first–order formula ψ.
Output of the SCAN — if it terminates — is a formula φ_α which is classically logically equivalent to α, but not containing the predicate variables $P_1, \ldots, P_n$.
SCAN performs the following three steps:

1. ψ is transformed into clause form using second order skolemization. That means the resulting formula has the form:
$$\exists P_1, \ldots, P_n \exists f_1, \ldots, f_n \psi'$$
where the f_i are the Skolem functions and ψ' is a set of clauses. From the algorithm's point of view, the quantifier prefix can be ignored. Therefore ψ' is treated as an ordinary clause set with the usual Skolem constants and functions.

2. All C–resolvents and C–factors with the predicate variables $P_1, \ldots, P_n$ have to be generated. C–resolution ('C' for constraint) is defined as follows:
$$\frac{\begin{array}{ll} P(s_1, \ldots, s_n) \vee C & P(\ldots) \text{ and } \neg P(\ldots) \\ \neg P(t_1, \ldots, t_n) \vee D & \text{are the } \textit{resolution literals} \end{array}}{C \vee D \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n}$$
and the C-factorization rule is defined analogously:
$$\frac{P(s_1, \ldots, s_n) \vee P(t_1, \ldots, t_n) \vee C}{P(s_1, \ldots, s_n) \vee C \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n}.$$
Notice that only C-resolutions between different clauses are allowed (no self resolution). A C-resolution or C-factorization can be optimized by destructively resolving literals $x \neq t$ where the variable x does not occur in t with the reflexivity equation. C–resolution and C–factorization takes into account that second order quantifiers may well impose conditions on the interpretations which must be formulated in terms of equations and inequations.

   As soon as *all* resolvents and factors between a particular literal and the rest of the clause set have been generated (the literal is 'resolved away' because self resolution is not allowed), the clause containing this literal must be deleted (purity deletion). If all clauses are deleted this way, this means that α is a tautology.

   All equivalence preserving simplifications may be applied freely. These are for example:

   - Tautologous resolvents can be deleted.
   - Subsumed clauses can be deleted.
   - Subsumption factoring can be performed. Subsumption factoring means that a factor subsumes its parent clause. This may be realized by just deleting some literals. For example $Q(x) \vee Q(a)$, where x is a variable, can be simplified to $Q(a)$.
   - Subsumption resolution can also be performed. Subsumption resolution means that a resolvent subsumes its parent clause, and this again may be realized by deleting some literals [Ohlbach and Siekmann, 1991]. For example the resolvent between $P \vee Q$ and $\neg P \vee Q \vee R$ is just $Q \vee R$ such that $\neg P$ can be deleted from the clause. (An instance of this operation is realized as so called 'unit_deletion' in the OTTER theorem prover.)

   If an empty clause is generated, this means that α is contradictory.

3. If the previous step terminates and there are still clauses left then reverse the skolemization. A method for reversing the skolemization in a set F of clauses is (1) to abstract all arguments of all occurrences of Skolem functions by variables, i.e. $f(s_1, \ldots, s_n)$ is replaced

with $f(x_1, \ldots, x_n)$ and additional literals $x_i \neq s_i$ are added to the clause where the x_i are fresh variables and (2) to consistently rename all variables such that the arguments of all occurrences of the Skolem function are the same. If this is possible and $F[f(x_1, \ldots, x_n)]$ is the result then $\forall x_1, \ldots, x_n \exists y\ F[y]$ is the solution. This process is repeated for all Skolem functions.

If it is not possible to rename the variables consistently, the only chance is to take parallel Henkin quantifiers [Henkin, 1961]

or leave the second–order quantification.

The next example illustrates the different steps of the SCAN algorithm in detail. The input is: $\exists P\ \forall x, y\ \exists z\ (\neg P(a) \vee Q(x)) \wedge (P(y) \vee Q(a)) \wedge P(z)$.

In the first step the clause form is to be computed:

$$\begin{array}{ll} C_1 & \neg P(a), Q(x)) \\ C_2 & P(y), Q(a) \\ C_3 & P(f(x, y)) \end{array}$$

f is a Skolem function. The second–order quantifier prefix is therefore $\exists P\ \exists f\ \forall x, y$. But this is only needed for the correctness proof below.

In the second step of SCAN we begin by choosing $\neg P(a)$ to be resolved away. The resolvent between C_1 and C_2 is $C_4 = Q(x), Q(a)$ which is equivalent to $Q(a)$ (this is one of the equivalence preserving simplifications). The C-resolvent between C_1 and C_3 is $C_5 = (a \neq f(x, y), Q(x))$. There are no more resolvents with $\neg P(a)$. Therefore C_1 is deleted. We are left with the clauses

$$\begin{array}{ll} C_2 & P(y), Q(a) \\ C_3 & P(f(x, y)) \\ C_4 & Q(a) \\ C_5 & a \neq f(x, y), Q(x) \end{array} .$$

Selecting the next two P-literals to be resolved away yields no new resolvents. Thus, C_2 and C_3 are simply to be deleted as well. All P-literals have now been eliminated. Restoring the quantifiers we then get

$$\forall x\ \exists z\ Q(a) \wedge (a \neq z \vee Q(x))$$

as the final result (y is no longer needed.)

Theorem 10.5 [Correctness of SCAN] If SCAN terminates for a formula α then α is logically equivalent to $SCAN(\alpha)$

Proof. The formulae under consideration contain a prefix of second–order existential quantifiers over predicate and function variables as the only second–order component. In order to prove the equivalence we can therefore take a standard first–order Tarskian model theory augmented with assignments of n-ary relations to n-place predicate variables and n-ary functions to n-place function variables.

Since we use second order skolemization, clause form generation as well as reversing the skolemization are equivalence preserving. Adding a resolvent or a factor to a clause set is also equivalence preserving. Therefore the only critical step in the SCAN algorithm is the purity deletion rule.

Removing a clause cannot make (in an interpretation) true clause sets false. Therefore every interpretation satisfying the clause set before the deletion satisfies it also after the deletion.

What we are left with to prove is that an interpretation satisfying the clause set without the pure clause also satisfies the clause set with the pure clause. And this turns out to be the really hard part of the proof where we have to exploit the second order character of the problem. What has to be exploited is that the predicate P is existentially quantified and therefore its interpretation can be chosen appropriately.

Before we come to the proof for the general case, it is useful to make some conceptual simplifications.

- Exploiting the equivalence
  $(P(s_1,\ldots,s_n)\vee C) \leftrightarrow (P(x_1,\ldots,x_n) \vee C \vee x_1 \neq s_1 \vee \ldots \vee x_n \neq s_n)$ it can be assumed that the predicate P which has been 'resolved away' in the pure clause C (C pure means that it has no resolution partner other than possibly self resolution) has only variables as arguments.

- The proof for an n-place predicate is not different to the proof for a one–place predicate. Just read x in $P(x)$ as a vector of variables. Without loss of generality we assume therefore that P is a one–place predicate.

- Since purity deletion is done after all resolvents and factors with the pure literal are generated, it can be assumed without loss of generality that there is only one resolution partner in the clause set. If there are n resolution partners in the clause set then all proof steps below can be repeated n times.

- The clauses containing no resolution partners do not contain complementary literals with the predicate P. They are not touched during the purity deletion process. In the sequel they can therefore be ignored.

- The variables in the clauses can be renamed such that different clauses *share* the same variables.

There are two cases which have to be distinguished. The first case is that the predicate P occurs in the pure clause C only with one sign, either positively or negatively. The second case is that P occurs with both signs, i.e. C is self resolving. This is the case where $SCAN$ may loop.

Let us now consider the first case and without loss of generality assume the predicate P occurs only positively in C.

Thus, the situation before and after purity deletion looks as follows:

$$\text{before:}K = \left\{ \begin{array}{l} P(x), C(x) \qquad (=_{def} \ A) \\ Factors(A) \\ \dfrac{\neg P(x_1),\ldots,\neg P(x_n), D(x_1,\ldots,x_n)}{S(x_1),\ldots,S(x_n), D(x_1,\ldots,x_n)} \\ \quad \vdots \end{array} \right.$$

$$\text{after:}K' = \left\{ \begin{array}{l} Factors(A) \\ \dfrac{\neg P(x_1),\ldots,\neg P(x_n), D(x_1,\ldots,x_n)}{S(x_1),\ldots,S(x_n), D(x_1,\ldots,x_n)} \\ \quad \vdots \end{array} \right.$$

where C and D denote the remaining literals. These literals may well contain additional variables. For the purpose of this proof these variables can be ignored. C and D may also contain additional *positive* literals with the predicate symbol P. $S(x_1),\ldots,S(x_n), D(x_1,\ldots,x_n)$ stands for the 2^n-1 resolvents which are possible between these two clauses. S denotes either $\neg P$ or C. For example if $n=2$ there are three resolvents:

$$\begin{array}{l} \neg P(x_1), \quad C(x_2), D(x_1,x_2) \\ C(x_1), \neg P(x_2), D(x_1,x_2) \\ C(x_1), \quad C(x_2), D(x_1,x_2) \end{array}$$

If tautologies are automatically eliminated those resolvents which are either themselves tautologies or which are derived from tautologies are not present. We shall see that the factors of A are needed to take over the role of those clauses which are derived from tautologies. Subsumed clauses, however, may be deleted without any effect on the proof.

Take any interpretation $\Im$ satisfying K' and mapping the symbol P to a predicate $\mathcal{P}$ and the variable x to a domain element a. If $\Im$ satisfies $C(x)$ or $P(x)$ then $\Im$ satisfies K and we are done. If $\Im$ falsifies both we move to an interpretation $\Im'$ by changing the assignment of P to a predicate

$\mathcal{P}'$ which is like $\mathcal{P}$ except that $\mathcal{P}'(a)$ is true. Then $\Im'$ satisfies $P(x), C(x)$. We have to show that $\Im'$ still satisfies all the other clauses.

Assume $\Im'$ maps the variables x_i to some a_i. Let J be the set of variables which are mapped to a, i.e. $x_i \in J$ if $a_i = a$. For these variables, the truth value of $\neg P(x_i)$ has changed from *true* under $\Im$ to $false$ under $\Im'$. For all other variables nothing has changed. Therefore if $\neg P(x_j), j \notin J$ is true under $\Im$, it is still true under $\Im'$. In this case all clauses containing this literal are still true. Now suppose $\neg P(x_j), j \notin J$ are all false under $\Im$. If for simplicity we assume $J = \{x_1, \ldots, x_j\}$, there is a clause $M = C(x_1), \ldots, C(x_j), \neg P(x_{j+1}), \ldots, \neg P(x_n)$, $D(x_1, \ldots, x_n)$ among the resolvents. In this clause, all literals with predicate C and $\neg P$ are false under $\Im$. Since $\Im$ satisfies K', it must satisfy $D(x_1, \ldots, x_n)$. The interpretation of D has not changed. Therefore $\Im'$ satisfies $D(x_1, \ldots, x_n)$ as well. Thus, $\Im'$ satisfies all clauses in K.

It remains to be shown that in case of automatic tautology deletion the critical clause M is *not* deleted. If M would either itself be a tautology or derived from a tautology, the clause A would look like $A = P(x), P(y), C'(x, y)$. In this case the structure of M would be
$M = P(y_1), C'(x_1, y_1), \ldots, P(y_1), C'(x_j, y_j), \neg P(x_{j+1}), \ldots, \neg P(x_n), D(x_1, \ldots, x_n)$. That means for example $P(y_1), C'(x_1, y_1)$ would be false for all assignments of y_1, in particular for $\Im(y_1) = a$. This assigment would also falsify the factor $P(x), C'(x, x)$ of clause A, which cannot be the case[23].

From this we conclude that both $\Im'$ and $\Im$ satisfy $\exists P\ K$.

The remaining case to be considered is the case where the clause C is self resolving. Schematically the situation looks as follows:

$$\text{before:} K = \left\{ \begin{array}{l} P(x), \neg P(y), C(x, y) \\ \neg P(x), D(x) \\ \hline \neg P(y), D(x), C(x, y) \\ \neg P(y), D(x), C(x, x'), C(x', y) \\ \quad \vdots \quad \text{(possibly infinitely many resolvents)} \end{array} \right.$$

$$\text{after:} K' = \left\{ \begin{array}{l} \\ \neg P(x), D(x) \\ \hline \neg P(y), D(x), C(x, y) \\ \neg P(y), D(x), C(x, x'), C(x', y) \\ \quad \vdots \end{array} \right.$$

To simplify things, let us assume neither $C(x, y)$ nor $D(x)$ contain negative occurrences of P. If in a given interpretation $\Im$ which maps x to some a_0, $D(x)$ is true, we can choose $P(x)$ to be true without further conflicts. If $D(x)$ is false, but $C(x, y)$ is true, where y is mapped to some a_1, $\Im$ satisfies K. If both $D(x)$ and $C(x, y)$ are false then the first resolvent enforces $\neg P(y)$ to be true. That means, $P(a_0)$ to be false enforces $P(a_1)$ to be false and therefore it has to be checked whether the first clause still remains true under the assignment $x \mapsto a_1$ etc. With the same arguments as in the base case this is proved with induction on n using the n^{th} resolvent. That means that in this case $\Im$ again satisfies $\exists P\ K$.

The case that $C(x)$ contains further negative literals with P means that there is another recursion loop which generates new branches of resolvents. Induction on the number of these further literals proves the statement.

The case that $D(x, y)$ also contains negative literals with P requires the integration of the arguments we used to prove the first case into the proof for the second case. This is technically complicated, but there are no further proof ideas needed.

Collecting everything together we can finally conclude $\alpha \leftrightarrow SCAN(\alpha)$. $\lhd$

If the formula given to SCAN contains a cycle in the P-literals, SCAN may keep on producing infinitely many clauses. In some cases the size of the clauses remains finite. According to a result of Ackermann [1935a, 1935b] which can be adapted to our case, the (possibly infinite) conjunction of the P-literal free clauses is a solution. It is one of the advantages of SCAN that in this case

[23] This argument does not imply that only binary factors are needed. Before the factor itself is operated on, its factor has to be generated. That means all factors are needed.

a subsumption test on the resolvents may terminate the process and thus compute a finite result whereas otherwise only infinite junk would be produced.

As an example where this happens, apply SCAN to the formula:

$$\exists P\ \forall x, y\ (P(x,y) \to P(y,x)) \wedge (P(x,y) \leftrightarrow Q(x,y)).$$

Since the symmetry clause $\neg P(x,y), P(y,x)$ contains only a trivial cycle which can easily be recognized, SCAN stops and returns as expected the symmetry of Q.

There is, however, no proof that SCAN stops in all cases where there is a finite solution. If this were the case then it would be decidable whether a theorem $\exists x\ P(x)$ has finitely many different proofs or not.

As mentioned earlier, formulae with universal quantifiers have to be negated before giving them to SCAN and the result has to be negated again. The question may arise whether there is a possibility to treat universally quantified variables directly. Eliminating P from formulae $\forall P\ F[P]$ in some sense means factoring out the tautological part of $F[P]$. For example $(\forall P\ (P \vee \neg P) \wedge Q)$ is equivalent to Q, i.e. the P–part is tautologous. SCAN uses resolution as the basic operation, and resolution is sensitive to contradictions and not to tautologies. Therefore it is resolution which requires negation of the formula and elimination of the contradictory part.

In applications like circumscription, the structure of the formulae is $\forall P^*\ Q[P^*] \Rightarrow R[P^*]$ with a large $Q[P^*]$ and a small $R[P^*]$. In this case it is much more convenient to negate the formula yielding $\exists P^*\ Q[P^*] \wedge \neg R[P^*]$ because the big $Q[P^*]$ remains untouched.

There is a corollary derived from the proof of SCAN which may be of some interest. The proof says that deleting a clause as soon as one literal is resolved away preserves equivalence. This may be exploited to eliminate only certain unwanted formulae. As an example, consider a PROLOG program containing a binary predicate P which is *symmetric*. Adding the symmetry clause to the program clauses PROLOG to loop. The corollary allows to eliminate the symmetry clause by generating all non redundant resolvents with the other PROLOG clauses. That means in this case that all clauses containing some $P(s,t)$ are duplicated with $P(t,s)$ replacing $P(s,t)$ in the copy. For all queries not containing P, the new PROLOG program is equivalent to the old one together with the symmetry clause.

Remark 10.6 [Comparison with other Methods] Wilhelm Ackermann gave two procedures for eliminating existential quantifiers. Both eliminate only one quantifier at a time. The first one requires to bring the formula into a form

$$\exists P\ \forall x\ (A(x) \vee P(x)) \wedge \Delta[\neg P]$$

where $\Delta[\neg P]$ is a formula containing only negative occurrences of $P(x)$. The result is then $\Delta[A]$, i.e. all occurrences of $\neg P(x)$ are replaced with $A(x)$ in Δ. This method has difficulties in handling problems with clauses containing several occurrences of P. For example

$$\exists P\ \forall x, y\ (P(x,a) \vee P(a,x) \vee C(x)) \wedge (\neg P(y,a) \vee \neg P(a,y) \vee D(y))$$

falls into this problematic class. The SCAN–solution for this case, however, is simply $C(a) \vee D(a)$. In its kernel the second method of Ackermann is actually quite similar to SCAN. Although his notation is very different to ours, it amounts to generating the conjunction of all P–free resolvents[24]. It is, however, also restricted to one–place predicates. Literals $P(s_1, \ldots, s_n)$ have therefore to be written as $First(x, s_1) \wedge \ldots \wedge \mathrm{N}^t h(x, s_n) \to P'(x)$ before this method can be applied. This transformation blows up the formulae considerably. In fact, if SCAN is reduced to formulae with one-place predicates where all arguments are variables and no further simplifications of the resolvents are applied, you obtain Ackermann's second method.

[24]Historical note: On page 401 of [Ackermann, 1935a] there is the definition of an operation

$$\mathcal{A}^{x_1,\ldots,x_n,z}_{y_1,\ldots,y_m} \wedge \mathcal{B}^{p_1,\ldots,p_n}_{q_1,\ldots,q_l,z} \to \mathcal{A}^{x_1,\ldots,x_n}_{y_1,\ldots,y_m} \vee \mathcal{B}^{p_1,\ldots,p_n}_{q_1,\ldots,q_l}$$

where the subscripts y_i stand for $P(y_i)$ and the superscripts x_i stand for $\neg P(x_i)$ in a clause containing also literals $\mathcal{A}$ or $\mathcal{B}$ respectively. Thus, contraction on z actually means resolution between $P(z)$ and $\neg P(z)$. The step to full resolution as we know it now is not that big.

Since Ackermann does not use resolution as we do, it is very difficult to integrate optimization steps like subsumption deletion etc. That means that his method would not terminate for the above example with the symmetry clause. Therefore SCAN is much easier to handle and it behaves better than Ackermann's method in such pathological cases.
The idea of generating consequences of the formulae with P and then taking the subset of P–free formulae is actually the kernel of other approaches to this problem. For example a theorem in [Kreisel and Krivine, 1966] says that it is the set of *all* consequences you have to take. This is of course too much to be of practical value. A minimal subset free of redundancies should be sufficient. We showed that the set of resolvents without tautologies and subsumed clauses is sufficient. Bachmair, Ganzinger and Waldmann [Bachmair *et al.*, 1992] have gone even one step further. Their 'hierarchical theorem proving' approach allows the formulation of redundancy criteria based on term orderings. Furthermore they have incorporated equality reasoning by superposition principles. This mechanism can be used to get rid of existentially quantified predicate *and function symbols*.

Example 10.7 [Interpolation] As mentioned in the introduction, the applications we have in mind are classes of problems where formulae with the structure $\exists P_1, \ldots, P_n \psi$ or $\forall P_1, \ldots, P_n \psi$ occur. This need not be second–order formulae in the first place. Even in standard first–order logic there might be useful applications. Suppose there is an axiomatization of something in terms of a predicate P and maybe some other predicates, and by some reason it is known that only theorems not containing P are to be proved from these axioms. That means

$$\begin{array}{ll} & Axioms(P) \Rightarrow Theorem \\ \text{iff} & \forall P\ (Axioms(P) \Rightarrow Theorem) \\ \text{iff} & (\exists P\ Axioms(P)) \Rightarrow Theorem \\ \text{iff} & SCAN(\exists P\ Axioms(P)) \Rightarrow Theorem \end{array}$$

i.e. SCAN can optimize the axioms with respect to the particular class of theorems not containing P.
Actually the situation is a special case of *interpolation*. We have

$$\begin{array}{ll} & \forall Q, R\ [\varphi(Q,P) \Rightarrow \psi(P,R)] \\ \text{iff} & (\exists Q\ \varphi(Q,P)) \Rightarrow \forall R\ \psi(P,R) \\ \text{iff} & (SCAN(\exists Q\ \varphi(Q,P))) \Rightarrow \neg SCAN(\exists R \neg \psi(P,R)) \\ \text{iff} & \varphi'(P) \Rightarrow \psi'(P) \end{array}$$

i.e. SCAN does interpolation.

We are now ready to resume our disucssion of the translation into classical logic. The SCAN algoirthm can help reduce the Ψ of Step 3 to a first-order formula if possible. Let us do one more example

Example 10.8 [Łukasiewicz formulation of the implicational fragment of classical logic]
The implicational fragment of propositional logic can be axiomatized by modus ponens: from P and $P \Rightarrow Q$ derive Q, and one more axiom $\quad ((P \Rightarrow Q) \Rightarrow R) \Rightarrow ((R \Rightarrow P) \Rightarrow (S \Rightarrow P))$.
As a Hilbert system, we do not yet have a consequence relation of the form $A \mid\!\sim B$ but only of the form $\varnothing \mid\!\sim B$. Thus at this stage, our algebraic analysis and automatic translation methods into classical logic cannot be applied. Assuming that we use the meothos of Examples 9.0.16 and 9.0.17, we are likely to automatically prove the deduction theorem and define $A \mid\!\sim B$ as $\varnothing \mid\!\sim A \Rightarrow B$.
Now we can continue with our analysis and translation. It is our aim to automate the move from a Hilbert system to a corresponding conservative singleton consequence relation more effectively than the rough search proposed in Example 9.0.16, which happens to give us a very good result for this particular example.
The purpose of this example is to show that translation via SCAN is more transparent and effective than term translation. For this purpose we can add the deduction rule to Łukasiewicz system. This will give us a proper consequence relation which is still not at all known to be classical implication.

So the force of the example does not diminish. We now assume the strongest semantics of the $\Rightarrow$–connective (material implication) is not yet known, but we do know that $\Rightarrow$ is directional and hence has a possible world semantics in terms of a ternary relation. We assume further that $\leq$ is equality and T is truth.
The truth condition for $\Rightarrow$ is:

$$x \models P \Rightarrow Q \quad \text{iff}$$
$$\forall y, z\ [\mathcal{R}(x, y, z) \text{ and } y \models P \text{ imply } z \models Q].$$

We have seen how this semantics can be used to translate the above Hilbert axiom and modus ponens into predicate logic. For example the translation of modus ponens yields

$$\forall P, Q\ \forall x\ (P(x) \wedge \forall y, z\ (\mathcal{R}(x, y, z) \wedge P(y) \rightarrow Q(z)) \rightarrow Q(x))$$

If we do this for the above axiom also, we get two second–order formulae from which SCAN can eliminate the quantifiers. The result is:

$$\begin{array}{l}\text{SCAN(Modus Ponens)} = \forall z\ \exists x, y\ \mathcal{R}(x, y, z) \\ \text{SCAN(Axiom)} = \forall a, b, c, d, e, h, k \\ \quad ((\mathcal{R}(a, b, c) \wedge \mathcal{R}(c, d, e) \wedge \mathcal{R}(e, h, k)) \rightarrow \\ \quad \exists u, v\ (\mathcal{R}(b, u, v) \wedge \mathcal{R}(d, v, k) \wedge \\ \quad \forall x, y\ (\mathcal{R}(u, x, y) \rightarrow (k = x))))\end{array}$$

It can now be proved with standard predicate logic means[25] that the conjunction of these two formulae is equivalent to

$$\forall a\ \mathcal{R}(a, a, a) \wedge \forall a, b, c\ [\mathcal{R}(a, b, c) \rightarrow a = b \wedge a = c]$$

That means the relation $\mathcal{R}$ collapses to a point relation. This fact is the key lemma from which it is trivial to show that the $\Rightarrow$ connective is actually material implication.
The above considerations mean that the semantic translation of Łukasiewicz system reduces to the following

$$x \models P \Rightarrow Q \text{ iff } (x \models P \rightarrow x \models Q)$$

where $\rightarrow$ is classical implication. Thus the translation of $P \Rightarrow Q$ into classical logic is $\forall x[P^*(x) \rightarrow Q^*(x)]$ and we have

$$\vdash A \Rightarrow A \text{ iff } \vdash \forall x(A(x) \rightarrow A(x)).$$

This semantic translation should be compared with the metalevel *term translation* of this logic. The term translation into classical predicate logic uses one binary function symbol $f_{\Rightarrow}(x, y)$ and one unary predicate $T(x)$ (Theorem (x)). The following classical theory Δ represents the logic:

- $\forall x, y\ [T(x) \wedge T(f_{\Rightarrow}(x, y)) \rightarrow T(y)]$
- $\forall x, y, z, u\ T(f_{\Rightarrow}(f_{\Rightarrow}(f_{\Rightarrow}(x, y), z), f_{\Rightarrow}(f_{\Rightarrow}(z, x), f_{\Rightarrow}(u, x))))$

To check, for example, whether $\vdash A \Rightarrow A$ we check in classical logic whether $\Delta \vdash T(f_{\Rightarrow}(x, x))$.
Now let the reader consider whether there is anything to this 'reduction' beyond a 'change of notation' and whether it can seriously be taken as supporting the universality of any logic? We should, however, note that the step tansforming the Hilbert consequence into a singleton consequence relied on the deduction theorem and was not automated. We hope to close this gap in the future.
Of course the term translation does at least show that classical logic automatic reasoning machinery can assist (blindly) in proving theorems for any logic.

That means using SCAN and standard predicate logic theorem proving, both of which can be automatized, it is possible to analyze unknown logics and to find the strongest semantics. This semantics (plus some further optimizations) in turn serve as a basis for a translation into predicate logic.

[25]Thanks to Mark Reynolds who helped find the proof.

11 Conclusion: the current state of the debate

The current state of the debate is that an almost arbitrary logic can be translated into classical logic. This translation is more or less natural (not metalevel) and is beneficial from the automated deduction point of view. We now need to put forward arguments relating to two points

- Explain why we limit our support for the universality of classical logic to the automated reasoning aspect only.
- Even from the automated reasoning point of view, we do not yet have a conclusive argument in favour of classical logic as a universal language for automated deduction because classical logic itself can be translated into other logics, say intuitionistic logic or linear logic or even modal logic. Why not use these logics as universal?

We first address the second point:

The main support comes, in my opinion, from the availability of much better and much more effective automatic deduction methods in classical logic. By effective I do not only mean algorithmically tractable (most theorem proving for non-classical logics are done on classical machines via translation). I also mean that there are certain algorithms for eliminating quantifiers such as SCAN as well as certain constructions which are logically meaningful and which can, at the current state of knowledge, be carried out only in classical logic.

To explain this latter point, we take an example. Consider

$$\varphi = (A(z) \Rightarrow \forall x(B(x,y) \Rightarrow \exists y C(x,y)))$$

This formula, in an *unknown* implicational logic, has several problems associated with it, when it comes to preparation for automated reasoning and the search for equivalent convenient normal forms

- Quantifiers cannot be moved about
- Skolem functions cannot be introduced easily. In fact, the existential on y probably depends not only on the nested arrows but also on the variable z, even though z is not in C at all.
- The subformula C is buried inside φ and is not accessible
- The rewrites allowed depend on the semantics and/or proof theory of $\Rightarrow$.

Classical logic is the *simplest* and most convenient with respect to these points. All parts of φ are accessible in classical logic on the surface via the prenex conjunctive normal form.

For application areas which use non-classical logic and which do not wish to translate into classical logic, still it is possible to perform automated deduction locally. The problem of Skolemization can be solved by what we call 'run time skolemization'. The computation proceeds and whenever there is a need to skolemise in a context (label) t, one can introduce a constant c^t, labelled by the context. Visa rules must govern the process of transporting the constants from one context to another. Such methods are natural in the framework of *Labelled Deductive Systems* and the rules of the case studies of section 3 illustrate this. Note especially that *LDS* already uses the notation $t : c(s)$ to mean the contant c was created at label (context) s and is currently available at label (context) t.

Although one may argue that linear logic also has a convenient normal form and enjoys a lot more structure in its proof theory, the extra structure is actually a disadvantage. We can record all the structure through our labels (or the term sort of the linked predicate languages), so we don't need this extra structure, which can only interact with the sort structure (if we were translating into linear logic) and cause possible confusion. In fact, effective quantificational automated deduction is a problem for the structurally richer logics and we do prefer the simplicity of classical logic. We do want the lack of structure in the logic, because we want to use the sorts for that purpose and use sort specific automated deduction.

We now address the first question, seeking evidence that classical logic cannot conveniently serve the role of a universal language beyond the aspect of automated reasoning. The main obstacle for this role is structure. The non-classical connectives are usually tailored for natural structure of the non-classical application and therefore there is a risk that when translated into classical logic all structure be lost. The problem is particularly acute if the translation is proof theoretically based and where the translation is not through subformulas. It may not matter if all we want is automated reasoning but if we actually want to use the logic then we may have to keep to the original non-classical presentation. In areas where logic is used for action (modal action logic, linear logic) the need to keep the structure is crucial. There is some work by H. J. Ohlbach [1993] on the functional translation which seems for the case of modal logic (binary accessibility) to preserve structure and be computationally effective. It is not known at this stage whether the functional translation can be automated and be kept structure preserving in general, e.g. for ternary accessibility.

A metaleval transltion into **HFP** will also preserve structure, and this can be taken as another 'good' translation into classical logic.

To summarise, it seems that the question of whether many sorted classical logic can be universal beyond the relam of automated deduction, depends very much on the state of the art of our translation methods.

References

[Ackermann, 1935a] Wilhelm Ackermann. Untersuchung über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[Ackermann, 1935b] Wilhelm Ackermann. Zum Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 111:61–63, 1935.

[Alchourrón *et al.*, 1985] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50:510–530, 1985.

[Babb, 1990] E. Babb. An incremental pure logic language with constraints and classical negation. In T. Dodd, R. Owens, and S. Torrance, editors, *Logic Programming: Expanding the Horizons*, pages 14–63. Blackwell, Oxford, 1990.

[Bachmair *et al.*, 1992] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Theorem proving for hierarchic first-order theories. In G. Levi and H. Kirchner, editors, *Algebraic and Logic Programming, Third International Conference*, pages 420–434. Springer-Verlag, LNCS 632, September 1992.

[Blok and Pigozzi, 1989] W. J. Blok and D. Pigozzi. *Algebraizable Logics.* Memoires of AMS No 396, American Mathematical Society, 1989.

[Brachman *et al.*, 1989] R. J. Brachman, H. J. Levesque, and R. Reiter (eds.). *KR '89: Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann, San Mateo, California, 1989.

[Gabbay and Ohlbach, 1992] D. M. Gabbay and H-J. Ohlbach. quantifier elimination in second order predicate logic. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of KR'92*, pages 425–435, 1992. A short version appeared in *South African Computer Journal*, **7**, 35–43, 1992.

[Gabbay *et al.*,] D. M. Gabbay, L. Giordano, and A. Martelli. Conditional logic programming. Manuscript, University of Turin and Imperial College.

[Gabbay *et al.*, 1993] D. M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects.* Oxford University Press, 1993.

[Gabbay, 1969] D. Gabbay. The Craig interpolation theorem for intuitionistic logic. In *Logic Colloquium '69*, 1969.

[Gabbay, 1976] D. M. Gabbay. *Semantical Investigations in Modal and Temporal Logics*. D. Reidel, 1976.

[Gabbay, 1985] D. Gabbay. Theoretical foundations for nonmonotonic reasoning in expert systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*. Springer-Verlag, Berlin, 1985.

[Gabbay, 1986] D. M. Gabbay. Investigations in Heyting intuitionistic logic. D. Reidel, 1986.

[Gabbay, 1987] D. M. Gabbay. Modal and temporal logic programming, part I. In A. Galton, editor, *Temporal logics and their applications*, pages 197–237. Academic Press, 1987.

[Gabbay, 1990] D. M. Gabbay. Modal and temporal logic programming, part II. In T. Dodd, R. Owens, and S. Torrance, editors, *Logic Programming: Expanding the Horizons*, pages 82–124. Blackwells, 1990.

[Gabbay, 1992a] D. Gabbay. Theory of algorithmic proof. In *Handbook of Logic in Theoretical Computer Science, Volume 1*, pages 307–408. Oxford University Press, 1992.

[Gabbay, 1992b] D. M. Gabbay. Fibred semantics and the combination of logics, August 1992. Lectures given at Logic Colloquium 1992, Veszprém, Hungary.

[Gabbay, 1992c] D. M. Gabbay. How to construct a logic for your application. In *Proceedings of the German AI Conference. GWAI 92*, pages 1–30. Lecture Notes on AI, vol 671, Springer, 1992.

[Gabbay, 1992d] D. M. Gabbay. LDS and situation theory. In *Proceedings STA III*, 1992.

[Gabbay, 1992e] D. M. Gabbay. Modal and temporal logic programming, part III. In L. F. del Cerro and M.Penttonen, editors, *Non-classical Logic Programming*, pages 85–124. Oxford University Press, 1992.

[Gabbay, 1993a] D. M. Gabbay. General theory of structured consequence relation. Studies in Logic and Computation. Oxford University Press, 1993.

[Gabbay, 1993b] D. M. Gabbay. Partially ordered algebraic logics, 1993. In preparation.

[Gabbay, 1993c] D. M. Gabbay. What is a logical system. In D. M. Gabbay and H-J. Ohlbach, editors, *What is a Logical System?* Oxford University Press, 1993. To appear.

[Gabbay, 1994] Dov M. Gabbay. *Labelled Deductive Systems. Part I.* Oxford University Press, 1994. First Draft 1989. Preprint, Department of Computing, Imperial College, London, SW7 2BZ, UK.Current Draft February 1991, 265pp. Published as CIS-Bericht-90-22, Centrum für Informations- und Sprachverarbeitung, Universität München, Germany.

[Henkin, 1961] L. Henkin. Some remarks on infinitely long formulas. In *Infinitistic Methods*, pages 167–183. Pergamon Press, Oxford, 1961.

[Köning, 1992] E. Köning. A hypothetical reasoning algorithm for linguistic analysis. Paper submitted to JLC, 1992.

[Kowalski, 1986] R. A. Kowalski. *Logic for Problem Solving*. North Holland, 1986.

[Kraus *et al.*, 1990] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990. A preliminary version, with authors Kraus and Lehmann only, was presented under the title "Nonmonotonic logics: models and proofs" to *JELIA: European Workshop on Logical Methods in Artificial Intelligence*, Roscoff France, June 1988. Another preliminary version, with all three authors, appeared under the title "Preferential models and cumulative logics", Technical Report TR 88-15, Department of Computer Science, Hebrew University of Jerusalem, November 1988.

[Kreisel and Krivine, 1966] G. Kreisel and J.L. Krivine. *Éléments de Logique Matheématique. Théorie des modèles.* Société Mathématique de France, 1966.

[Lehmann and Magidor, 1992] D. Lehmann and M. Magidor. What does a conditional knowledge base entail? To appear in *Artificial Intelligence*, 1992. This paper gathers together and extends the material of a paper by Lehmann alone, with the same title, in [Brachman *et al.*, 1989], 212-222, and Technical Report 88-16 by Lehmann and Magidor, Department of Computer Science, Hebrew University of Jerusalem, November 1988.

[Makinson, 1989] D. Makinson. General theory of cumulative inference. In M. Reinfrank et al., editors, *Non-Monotonic Reasoning*, volume 346 of *Lecture Notes on Artificial Intelligence*, pages 1–18. Springer-Verlag, Berlin, 1989.

[Ohlbach and Siekmann, 1991] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgraf Karl Refutation Procedure. In Jean Luis Lassez and Gordon Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, pages 41–112. MIT Press, 1991.

[Ohlbach, 1993] Hans Jürgen Ohlbach. Optimized translation of multi modal logic into predicate logic. In Andrei Voronkov, editor, *Proc. of Logic Programming and Automated Reasoning (LPAR)*, pages 253–264. Springer LNAI, Vol. 698, 1993.

[Perlis, 1985] D. Perlis. Languages with self reference I: foundations. *Artificial Intelligence*, 25:301–22, 1985.

[Perlis, 1988] D. Perlis. Languages with self reference II: knowledge, belief and modality. *Artificial Intelligence*, 34:179–212, 1988.

[Scott, 1974] D. Scott. Completeness and axiomatizability in many valued logics. In *Proceedings of the Tarski Symposium*, pages 411–436, Providence, Rhode Island, 1974. American Mathematical Society.

[Simmons, 1993] Harold Simmons. The monotonous elimination of predicate variables. *Journal of Logic and Computation*, 1993. Forthcoming.

[Szałas, 1992] Andrzej Szałas. On correspondence between modal and classical logic: Automated approach. Technical Report MPI–I–92–209, Max-Planck-Institut für Informatik, Saarbrücken, March 1992.

[Tarski, 1936] A. Tarski. On the concept of logical consequence. In *Logic, Semantics, Metamathematics.* Oxford University Press, 1936.

[van Benthem, 1984] Johan van Benthem. Correspondence theory. In Gabbay Dov M and Franz Guenthner, editors, *Handbook of Philosophical Logic, Vol. II, Extensions of Classical Logic, Synthese Library Vo. 165*, pages 167–248. D. Reidel Publishing Company, Dordrecht, 1984.

[van Benthem, 1992] J. van Benthem. The logic of programming. *Fundamenta Informatica*, 1992. To appear.

[Wójcicki, 1988] R. Wójcicki. *Theory of Logical Calculi.* Reidel, Dordrecht, 1988.

[Wójcicki, to appear] R. Wójcicki. An axiomatic treatment of non monotonic arguments. *Studia Logica*, to appear.