MAX-PLANCK-INSTITUT FÜR INFORMATIK

Computer Support for the Development and Investigation of Logics

Hans Jürgen Ohlbach

MPI–I–94–228 July 94

Im Stadtwald
66123 Saarbrücken
Germany

Computer Support for the Development and Investigation of Logics

Hans Jürgen Ohlbach

MPI–I–94–228 July 94

Author's Address

Hans Jürgen Ohlbach
Max–Planck–Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
F. R. Germany
email: ohlbach@mpi-sb.mpg.de

Acknowledgements

I would like to thank Dov Gabbay as well as my colleagues Andreas Nonnengart and Renate Schmidt for very fruitful discussions.

This work was supported by the ESPRIT project 6471 MEDLAR and by the BMFT funded project LOGO (ITS 9102).

Abstract

Symbolic reasoning in a logical framework becomes more and more important for computer applications such as Natural Language Processing Systems or Expert Systems. These applications usually need specifically tailored logics. Therefore we are developing methods and algorithms for supporting the designer of an application system, who is usually not a logician, to develop his own application oriented logic.

This paper gives an overview about our current state of these investigations. In particular we consider the correspondences between axiomatic and semantic specifications of a logic and the problem of finding one from the other. Correlated with this area are translation methods from the object logic into predicate logic, and methods for optimizing the translation. Other topics are investigations of the expressiveness of a logic and the axiomatizability of semantic conditions.

The basic techniques underlying our approach, so called K-transformations and quantifier elimination, are briefly discussed. They are quite general mechanisms for manipulating predicate logic formulae, and the investigation of logics is only one of their applications.

For the technical details of the methods and the proofs I refer to the original papers.

Contents

1 Introduction

Symbolic reasoning in a logical framework becomes more and more important for computer applications such as Natural Language Processing Systems or Expert Systems. Unfortunately there is not just *the* universal logic as the basis for every application of logic. Of course, higher–order logic is expressive enough to emulate all other logics, but it has so many unpleasant features that it is useless for most practical applications.

There is a tradeoff between expressiveness of a logic and the complexity of the reasoning algorithms for this logic. For obtaining an optimal result in a particular application, one has therefore to find a compromise between these two features. However, not every designer of an application program, which needs logic in some of its components, is a logician and can develop the optimal logic for his purposes, neither can he hire a trained logician to do this for him. In this situation we could either resign and live with non–optimal solutions, or we could try to make computers themselves expert logicians.

Although this is a very ambitious goal, there is some evidence that it is in fact possible, at least to a certain extent. I just want to mention the MULTLOG system [BFOZ93]. This is a Prolog program that accepts as input the truth tables of a finitely many–valued logic, and produces as output a LaTeX document describing various calculi for this logic. The first MULTLOG generated paper has already been accepted at a conference.

In this paper I give a survey on our main activities in this area. Problems we have investigated so far are:

- reasoning in Hilbert systems,
- finding model theoretic semantics for axiomatically specified logics, and vice versa
- finding corresponding axioms for semantic properties,
- investigating the expressiveness of logics,

- finding translations from the object logic to predicate logic.

The procedures for solving these problems should of course guarantee soundness and completeness. Therefore we need a precisely defined meta theory and soundness and completeness results at this level. In most cases we use first–order predicate logic (PL1) as meta–logic for encoding and manipulating the object logics. The intention is to map the problems from the object logic level to the predicate logic level and to use the well established methods and results for PL1. As long as this is possible, there is therefore no need to use more expressive systems, higher order predicate logic, λ–calculus or type theory for example. On the other hand, since PL1 has no variable binding mechanism at the term level, this means that we are restricted to the propositional versions of the object logic.

The techniques we have developed, mainly transformations of logical formulae, are of a relatively general nature. Therefore, although our main interest was the development of logics, they are defined as general predicate logic methods, and some of them have already found other interesting applications.

The problems we are investigating are explained in some detail in Section 2. In each case I demonstrate how these problems are mapped to problems at the PL1 level. In section 3 the key techniques we have developed, Killer Transformations and Quantifier Elimination, are introduced. Finally, in Section 4, the application of these techniques for finding model theoretic semantics for Hilbert systems is described.

Due to space limitations, I cannot explain all the technical details and refer to the original papers. But I present the main ideas and results.

2 Problems Asking for Computer Support

Logics can be defined in various ways. The most abstract way is by means of a Hilbert system. A Hilbert system is a kind of grammar. It consists of axioms and rules. The axioms are actually axiom schemas because all instances of the axioms are considered as theorems. The rules specify how to derive new theorems from the initial theorems and the previously derived ones. For example the axiom

$$(p \to q) \to r) \to ((r \to p) \to (s \to p)) \tag{1}$$

together with Modus Ponens: from $p \to q$ and p infer q (2)

specify the implicational fragment of propositional logic [Łuk70, p. 295]. For encoding vague notions, like "knows", "believes" and "wants", a Hilbert style axiomatization is usually the method of choice because in Hilbert systems their properties can be expressed in a very abstract and intuitive way.

2.1 Reasoning in Hilbert Systems

A Hilbert system is a forward calculus. Starting with the axioms as the initial theorems, the rules of the Hilbert system derive new theorems. Verifying that a formula is in fact a theorem in the system amounts to enumerating all theorems until the formula eventually appears. Computers can solve this problem with a well known technique.

Using first–order predicate logic as meta logic, the Hilbert system can be encoded as a Horn theory. The logical connectives are encoded as function symbols and formulae are encoded as terms. The propositional variables in Hilbert axioms are place holders for arbitrary formulae. Therefore they become universally quantified variables in the encoded axiomatization. For example the system consisting of (1) and (2) can be encoded as the predicate logic clauses

$$\forall p,q,r,s \quad Th(i(i(i(p,q),r),i(i(r,p),i(s,p)))) \tag{3}$$

$$\forall p,q \quad Th(i(p,q)) \wedge Th(p) \Rightarrow Th(q) \tag{4}$$

'Th' is the only predicate needed. $Th(p)$ means 'p is a theorem'. 'i' denotes the implication connective. A PL1 encoding of a Hilbert system together with a theorem to be proved is now a

suitable input for an *automated theorem prover*. And in fact, these kinds of problems have been used for a long time as test problems in the automated theorem proving community [MW92].

Hilbert systems as you can find them in logic textbooks usually specify each aspect of the logic explicitly. The theorems proved from such systems in textbooks are in general so simple that for current days automated theorem provers there is absolutely no problem to prove them. However, if the Hilbert system is optimized by minimizing the number of axioms, the proof of quite simple theorems can already become extremely complicated. Proofs may require hundreds of rule applications, and finding them may need hours of CPU time. The technique of clause K-transformations, presented in Section 3.3, improves the behaviour of automated theorem provers for these more complex systems.

2.2 Finding Model Theoretic Semantics for Hilbert Systems

A specification of a logic with a Hilbert system is quite intuitive, but for many purposes it is not adequate. One reason for developing an alternative to Hilbert systems is the desire to get more efficient calculi. Another reason is to understand the logic better by bringing properties to the surface which are sometimes very deeply hidden. An alternative way for describing a logic is by mapping the syntactic constructs to a (hopefully) simple and well understood mathematical structure. Typical examples for this *semantical* description of a logic are Tarski's set theoretic semantics for predicate logic or Kripke's possible worlds semantics for modal logic [Kri59, Kri63].

The usual way, the correlations between the axiomatic description and the semantics are presented is: the axioms and the semantics are defined and soundness and completeness are proved. Soundness and completeness guarantee that a formula is a theorem in the axiomatic description if and only if it is a valid formula in the semantics. *Finding* an appropriate semantic structure, however, is nontrivial and requires experience and intuition.

As examples for semantics of a logic consider the different versions of the semantics of modal logic. Common to all of them is the possible worlds framework as basic semantic structure. Each possible world determines the interpretation of the propositional variables and the classical connectives in the usual way. The interpretation of formulae with non–classical operators is defined in terms of relations or functions connecting the worlds. The weakest semantics for modal logic is the (weak) neighbourhood semantics (also called minimal model semantics [Che80]). Each world has sets of worlds as 'neighbourhoods'. A formula $\Box p$ is true in a world w iff the truth set of p, i.e. the set of worlds where p is true, is among w's neighbourhoods. This semantics satisfies the ME rule, $p \Leftrightarrow q$ implies $\Box p \Leftrightarrow \Box q$, but no stronger axiom or rule. In strong neighbourhood semantics, $\Box p$ is true in a world w iff one of w's neighbourhoods is a *subset* of p's truth set. Strong neighbourhood semantics satisfies a monotonicity property: $p \Rightarrow q$ implies $\Box p \Rightarrow \Box q$. The next stage is the well known Kripke semantics with a binary accessibility relation. But this is not the end of the story. For example, modal logic S5 has a semantics in terms of an accessibility relation with the extra condition that the accessibility condition is an equivalence relation. This condition guarantees that the S5 axioms hold. An alternative semantics for S5 has the truth condition for the $\Box$–operator: $\Box p$ is true in a world iff p is true everywhere. In this semantics without accessibility relation, all S5 axioms are tautologies.

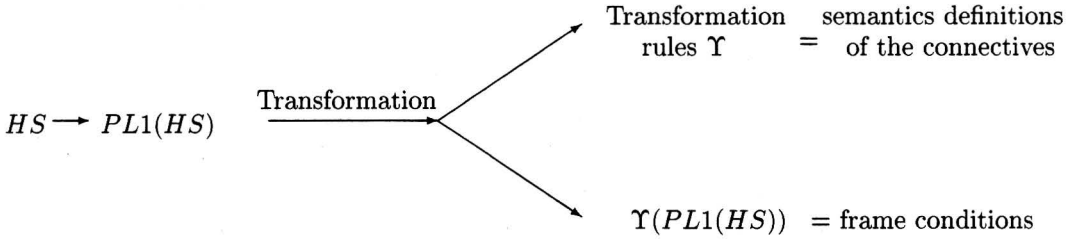
Each version of the semantics consists of two parts. The *basic semantics* contains just the definition of the primitive notions, neighbourhood relations or accessibility relations for example, and the satisfiability relation. The possible worlds together with the relations and functions operating on them are usually called *frames*.

The second part of the full specification of the semantics restricts the class of semantic structures by imposing constraints on the frames (so called *frame conditions*) and sometimes by restricting the assignment of truth values to the propositional variables. Modal logic T, for example is characterized by restricting the class of frames to those with reflexive accessibility relations. Intuitionistic logic as another example has a restriction on the assignment of propositional variables: if p is true in a world w then it remains true in all words accessible from w.

Each part of the semantics validates a certain part of the Hilbert axioms. The basic semantics of normal modal logic with binary accessibility relation for example validates the K-axiom

$\Box(p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)$ and the Necessitation rule: from p infer $\Box p$. The reflexivity condition on the accessibility relation validates the axiom $\Box p \Rightarrow p$. Obviously, there is a hierarchy of semantics. A semantics S_1 is *stronger* than a semantics S_2 if the basic part of S_1 validates more axioms than the basic part of S_2. A semantics is optimal for a Hilbert system if all axioms are validated in the basic part and no extra conditions are needed.

In [Ohl94] I addressed the problem of *finding the semantics* for a given Hilbert system. In this paper I proceed as indicated in the following picture.



We begin with an arbitrary Hilbert system HS and encode it in first–order predicate logic. The key observation underlying my approach for systematically finding a semantics comes from automated theorem proving. In order to prove a conjecture from assumptions, one need not perform the proof directly, but we can transform assumptions and conjectures and prove the transformed conjectures from the transformed assumption, provided the transformation Υ guarantees

$$assumption \Rightarrow conjecture \text{ iff } \Upsilon(assumption) \Rightarrow \Upsilon(conjecture). \tag{5}$$

The working hypothesis of my approach is therefore

> *The semantics of an axiomatically defined logic is the result of a carefully designed transformation of predicate logic formulae. The transformation rules represent the interpretation function and the non–tautologous transformed Hilbert axioms are the frame conditions.*

An optimal transformation turns all axioms into tautologies. In this case, proving a theorem φ from HS reduces to proving $\Upsilon(\varphi)$ without any additional assumptions. The guideline for finding a good transformation is therefore the intention to *turn axioms into tautologies*, or to make them in some other way redundant.

The problem of finding a semantics now reduces to two problems:

- We must find transformations for Hilbert axioms which turn them into tautologies. Ideally one would like to have a procedure *Trans_Gen* which, given a formula φ as input, computes a transformation Υ_φ with property (5) such that $\Upsilon_\varphi(\varphi) = true$. If $PL1(HS)$ consists of the formulae $\varphi_1, \ldots, \varphi_n$ one would compute $\Upsilon_{\varphi_1} = Trans\_Gen(\varphi_1)$, get $\varphi_i' = \Upsilon_{\varphi_1}(\varphi_i)$ for $i = 2, \ldots, n$, compute $\Upsilon_{\varphi_2'} = Trans\_Gen(\varphi_2')$, apply it to $\varphi_3', \ldots, \varphi_n'$ and repeat the process until all axioms are turned into tautologies. And, in fact, in [OGP94], we sketch such a procedure *Trans_Gen*. Unfortunately this idea does not work in general. The reason is that Υ may produce an infinite conjunction of formulae, and then the process does not terminate. But even if this method worked, it would not be very satisfactory because we do not want any transformation, but a transformation that gives some insight into the structure of the logic. I therefore proposed an approach which is restricted to a certain class of logics, but gives better results for them. The idea is to develop transformations for very concrete formulae or formula schemas. For each given Hilbert system, we then check which of the formulae out of our database of formulae with known transformations is a theorem. That means we decompose

  $$PL1(HS) \Leftrightarrow \varphi_1 \wedge \ldots \wedge \varphi_k \wedge rest$$

  and for the φ_i we develop transformations once and for all. The transformations for $\varphi_1 \wedge \ldots \wedge \varphi_k$ specify the basic semantics and the transformation of the *rest* gives the frame conditions.

- The transformed Hilbert axioms usually still contain formula variables. But what we are after, are frame conditions, i.e. formulae describing properties of the semantics structures, accessibility relations and the like. It turns out that in many cases the transformed Hilbert axioms can be simplified by eliminating the formula variables with a quantifier elimination algorithm. The result is in fact the axiom for the frame condition. In Section 3.4 I sketch the algorithm we have developed for eliminating formula variables.

After the introduction of the basic techniques, which are necessary to solve these problems, we come back to this discussion in Section 4.

2.3 Finding Hilbert Axioms from Semantic Properties

For axiomatizing vague notions whose mathematical structure is not clear, Hilbert systems are a good starting point. If, however, the semantic structure is clear, for example the time structure in a temporal logic, we might want to go the other direction and, starting with a semantics, develop a Hilbert system. For example we might want a linear and dense time structure and ask for the corresponding Hilbert axioms.

In order to solve this problem, one can again use PL1 as meta logic and encode the relevant information as PL1 axioms. In particular the interpretation of the connectives, which is known in this case, is written as PL1 equivalence with the binary satisfiability relation $\models$ and the PL1 encoding of the semantic notions. For example the interpretation of the modal logic $\Diamond$–operator becomes

$$\forall w, p\ w \models \Diamond p \Leftrightarrow \exists v\ R(w,v) \wedge v \models p,$$

which is an ordinary PL1 formula (in infix notation). We can encode the semantics of *all* relevant connectives this way, add the frame property we want to translate into a Hilbert axiom, and ask an automated theorem prover to enumerate all constructive proofs for a formula

$$\exists p\ \forall w\ w \models p.$$

That means we try to verify the existence of a tautology, a formula which is true in all worlds. Of course there are lots of them. Therefore each answer of the theorem prover must be checked by translating it back using the methods we have developed for the 'Hilbert system $\rightarrow$ semantics' direction. Hopefully the theorem prover eventually comes up with the right answer. In [BGO94] we have shown the details of this procedure and tested it with a lot of examples from modal and relevance logic. One of the examples was to find the Hilbert axiom corresponding to the transitivity of the accessibility relation in modal logic. In order to give an impression of the procedure, we list the protocol of a typical proof run with the OTTER theorem prover [McC89, McC90]. `s` is the satisfiability relation, `d` is the $\Diamond$–operator, `i` is the standard implication. | is OTTERS symbol for disjunction, `$ans` is a special literal for extracting variable bindings. It has no logical meaning.

```
%Interpretation of the connectives d and i.
(all z X (s(z,d(X)) <-> (exists x (R(z,x) & s(x,X))))).
(all z X (all Y (s(z,i(X,Y)) <-> (s(z,X) -> s(z,Y))))).
%Property to be translated.
(all x y z ((R(x,y) & R(y,z)) -> R(x,z))).
%Negated theorem.
-(exists f all z (s(z,f) & -$ans(f))).
end_of_list.
---------- PROOF ----------
  1  -s(z,d(x1))|R(z,f1(z,x1)).
  2  -s(z,d(x1))|s(f1(z,x1),x1).
  3   s(z,d(x1))| -R(z,x)| -s(x,x1).
  5   s(z,i(x2,x3))|s(z,x2).
  6   s(z,i(x2,x3))| -s(z,x3).
  7  -R(x,y)| -R(y,z)|R(x,z).
  8  -s(f2(x4),x4)|$ans(x4).
```

```
 12 [8,5]       $ans(i(x,y))|s(f2(i(x,y)),x).
 15 [12,2]      $ans(i(d(x),y))|s(f1(f2(i(d(x),y)),x),x).
 16 [12,1]      $ans(i(d(x),y))|R(f2(i(d(x),y)),f1(f2(i(d(x),y)),x)).
 19 [15,2]      $ans(i(d(d(x)),y))|s(f1(f1(f2(i(d(d(x)),y)),d(x)),x),x).
 20 [15,1]      $ans(i(d(d(x)),y))|
                 R(f1(f2(i(d(d(x)),y)),d(x)),f1(f1(f2(i(d(d(x)),y)),d(x)),x)).
115 [20,7,16]   $ans(i(d(d(x)),y))|
                 R(f2(i(d(d(x)),y)),f1(f1(f2(i(d(d(x)),y)),d(x)),x)).
171 [115,3,19] $ans(i(d(d(x)),y))|s(f2(i(d(d(x)),y)),d(x)).
174 [171,6]    $ans(i(d(d(x)),y))|s(f2(i(d(d(x)),y)),i(z,d(x))).
175 [binary,174,8] $ans(i(d(d(x)),d(x))).
------------ end of proof -------------
```

In the usual notation, this answer is $\Diamond\Diamond x \Rightarrow \Diamond x$, which is in fact the corresponding Hilbert axiom.

In principle we can reduce the problem of finding corresponding Hilbert axioms to a theorem proving problem. The search space, however, is in general very large. Therefore this is not yet a really satisfactory solution.

2.4 Simplifying Semantics

The semantics of a logic is usually formulated in mathematical notation. This is the most expressive formal(?) language we have. Therefore it is very easy to formulate in this mathematical language conditions on the semantic structure of our logic which have no counterpart on the syntactic side. But even if we do not exploit the full mathematical language and restrict ourselves to fragments of predicate logic, this effect may happen. For example it is well known that conditions like the irreflexivity or antisymmetry of the accessibility relation in modal logic are not axiomatizable in the corresponding Hilbert system. This means that the syntax of modal logic is so restricted that one cannot distinguish irreflexive frames from arbitrary frames. Thus, requiring irreflexivity has no effect at all on the theorems provable in modal logic. In other cases it may turn out that the syntactic side of a logic supports only weaker versions of semantic conditions as initially intended. It is of course very important to know the expressiveness of the logic, because otherwise the effects of semantic conditions are unpredictable.

A general technique for investigating the expressiveness of the logic $\mathcal{L}$ is the following: We formulate the semantics of $\mathcal{L}$ in a suitable meta logic $\mathcal{L}_t$, usually predicate logic, take a translation τ: $\mathcal{L} \rightarrow \mathcal{L}_t$ which ensures certain syntactic invariants on the translated formulae, and use these invariants to investigate the effect of the given semantic condition C on theorem proving search attempts for translated $\mathcal{L}$–formulae. It may for example turn out that C can never contribute to a proof search, or that parts of C are always redundant and do not contribute to proof search attempts. If the target logic $\mathcal{L}_t$ is PL1, all the results about proof search strategies and redundancy criteria can be used for this purpose. What predicate logic theorem provers usually do in order to get rid of irrelevant parts of the search space, now becomes valuable information about the expressiveness of $\mathcal{L}$.

A technique recently developed by Andreas Nonnengart supports these kind of investigations for modal logic. He has developed the so called semi–functional translation [Non93] from multi–modal logic to many–sorted predicate logic which produces clauses where the accessibility relation literals *occur only with negative sign.* The translation rules for the modal part are

$$\begin{aligned} \pi(\langle p\rangle c, w) &= \exists \gamma{:}AF_p\ \pi(c, w{:}\gamma) \\ \pi([p]c, w) &= \forall v\ p(w, v) \Rightarrow \pi(c, v) \end{aligned}$$

For example a formula $[p]\langle q\rangle c$ is translated into $\forall v\ p(w, v) \Rightarrow \exists \gamma{:}AF_q\ c(v{:}\gamma)$. Intuitively one can understand the sort AF_p as a set of functions mapping worlds to p–accessible worlds. The colon : is an infix function symbol which can be understood as application function, $a{:}\gamma \stackrel{\text{def}}{=} \gamma(a)$ (cf. Sec. 3.2.2). Only formulae in negation normal form can be translated this way. It is not necessary to understand the details of this translation technique here. The only fact we need is that the translated clauses contain *only negative* accessibility relation literals. From the translation there is

only one positive clause per accessibility relation, namely $\forall x\ \forall\gamma{:}AF_p\ p(x, x{:}\gamma)$. It relates the sort AF_p with the predicate p. If we do theorem proving by refutation with resolution (see below), we immediately see that there is no resolution partner at all for the irreflexivity clause $\neg p(x, x)$. This clause is redundant and cannot contribute to a proof. Thus, irreflexive accessibility relations are not characterizable in modal logic.

The semi–functional translation with the strong syntactic invariant that no positive accessibility literals ever occur in translated modal formulae, turned out to be an excellent basis for applying all kinds of redundancy criteria to eliminate or simplify frame conditions. Here we can exploit the results about resolution strategies with deletion operations [BG90], and the technique is really easy to apply.

3 Basic Techniques

3.1 Automated Theorem Proving

Some notions and notations are needed in this paper which are taken from the theorem proving literature.

A *clause* is a disjunction of literals. If A_i are the negative literals and B_j are the positive literals, we can write clauses in three different ways, either as a disjunction, $\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$ or as an implication $A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_m$ or as a set $\{\neg A_1, \ldots, \neg A_n, B_1, \ldots, B_m\}$. The variables in clauses are always considered universally quantified. Two different clauses are always considered as variable disjoint. A *ground clause* is a clause without variable symbols.

A *substitution* σ is an endomorphism in the free term algebra which changes only finitely many variables. We write substitutions as sets $\sigma = \{x_1 \mapsto s_1, x_2 \mapsto s_2, \ldots\}$. $s\sigma$ denotes the application of the substitution σ to the term s. $\sigma\tau$ denotes the composition of the two substitutions σ and τ.

$mgu(s, t)$ is the *most general unifier* for the two terms or atoms s and t. That means if $\sigma = mgu(s, t)$ then $s\sigma = t\sigma$. Since we do not consider theory unification, there is, up to variable renaming, at most one most general unifier for two terms. Two literals are *complementary unifiable* if they have different signs and the atoms are unifiable.

Resolution is the standard inference rule for many theorem provers [Rob65]. The definition of the resolution rule is

$$\frac{\begin{array}{ll} C_1\colon\ L_1 \vee L_2 \vee \ldots \vee L_n & L_1 \text{ and } K_1 \text{ are complementary unifiable.} \\ C_2\colon\ K_1 \vee K_2 \vee \ldots \vee K_m & \sigma \text{ is the most general unifier} \end{array}}{\sigma(L_2 \vee \ldots \vee L_n \vee K_2 \vee \ldots \vee K_m)}$$

L_1 and K_1 are the *resolution literals*. We say that we *resolve* the clauses C_1 and C_2 *upon* the resolution literals L_1 and K_1.

Self resolution is a resolution operation between two variable renamed copies of the same clause. For example a self resolution with the transitivity clause is

$$\frac{\begin{array}{l} \neg P(x, y) \vee \neg P(y, z) \vee \ P(x, z) \\ \qquad\qquad\qquad\qquad\quad | \\ \qquad\qquad\qquad\qquad \neg P(x', y') \vee \neg P(y', z') \vee P(x', z') \end{array}}{\neg P(x, y) \vee \neg P(y, z) \vee \neg P(z, z') \vee P(x, z')}$$

3.2 Formula K-Transformations

A formula K-transformation for a formula φ is a mapping Υ_φ from PL1 formulae to PL1 formulae such that

- for all formulae Φ and ψ: $\Phi \Rightarrow \psi$ iff $\Upsilon_\varphi(\Phi) \Rightarrow \Upsilon_\varphi(\psi)$
  (faithfulness = soundness and completeness)
- $\Upsilon_\varphi(\varphi)$ is a tautology
  (φ is 'killed' by Υ_φ, therefore the name 'K(iller)-Transfomation')

To illustrate the basic idea of formula K-transformations, suppose we have some set Φ of axioms which, among other things, axiomatize a reflexive and transitive relation R, i.e.

$$\forall x\ R(x,x) \tag{6}$$

$$\forall x,y,z\ R(x,y) \wedge R(y,z) \Rightarrow R(x,z) \tag{7}$$

are either contained in Φ or derivable from Φ, and we want to get rid of the reflexivity and transitivity of R.

In order to show that a formula ψ is entailed by Φ, one usually tries to refute $\Phi \wedge \neg\psi$. Before the refutation is actually started, every transformation on $\Phi \wedge \neg\psi$ which preserves satisfiability and unsatisfiability is allowed. Skolemization of existential quantifiers is a typical example of a routinely applied transformation which preserves satisfiability and unsatisfiability, but not logical equivalence.

The translation we propose for eliminating reflexivity and transitivity of R exploits that these two properties together imply

$$\forall x,y\ R(x,y) \Leftrightarrow (\forall w\ R(w,x) \Rightarrow R(w,y)). \tag{8}$$

To see this, suppose $R(x,y)$ and $R(w,x)$ hold. By transitivity, $R(w,y)$ also holds, i.e. the "$\Rightarrow$"–part is shown. For the "$\Leftarrow$"–part, take $w = x$ and use the reflexivity of R to derive $R(x,y)$.

Since (8) is entailed by Φ, we could add it to $\Phi \wedge \neg\psi$ without loosing satisfiability or unsatisfiability. However, instead of (8), we add

$$\forall x,y\ R(x,y) \Leftrightarrow (\forall w\ R'(w,x) \Rightarrow R'(w,y)) \tag{9}$$

to $\Phi \wedge \neg\psi$ where R' is a *new* predicate symbol. We can even go one step further and introduce a new sort symbol W for the variable w and add

$$\forall x,y\ R(x,y) \Leftrightarrow (\forall w{:}W\ R'(w,x) \Rightarrow R'(w,y)) \tag{10}$$

to $\Phi \wedge \neg\psi$.

Clearly, if $\Phi \wedge \neg\psi$ is satisfiable then $\Phi \wedge \neg\psi \wedge (10)$ is also satisfiable: the interpretation of R' can be chosen to be the same as the interpretation of R. In this case (10) is equivalent to (8), which follows from Φ. Thus, (10) is also true in the extended interpretation. On the other hand, if $\Phi \wedge \neg\psi \wedge (10)$ is satisfiable then certainly $\Phi \wedge \neg\psi$ is satisfiable as well.

But now we have a definition of R in terms of R' where R' is an uninterpreted new predicate symbol. In the next step, (10) is used as a rewrite rule from left to right, replacing all occurrences of R in $\Phi \wedge \neg\psi$ by the formula with R'. We obtain the transformed formula $\Phi' \wedge \neg\psi' \wedge (10)$ with R' in place of R. This is a terminating equivalence preserving transformation.

What happens to the reflexivity and transitivity of R? $\forall x\ R(x,x)$ becomes $\forall x\ \forall w\ R'(w,x) \Rightarrow R'(w,x)$ which is a tautology (by the reflexivity of "$\Rightarrow$"). The transitivity (7) becomes $\forall x,y,z\ (\forall w\ R'(w,x) \Rightarrow R'(w,y)) \wedge (\forall w\ R'(w,y) \Rightarrow R'(w,z)) \Rightarrow (\forall w\ R'(w,x) \Rightarrow R'(w,z))$ which is also a tautology (by the transitivity of $\Rightarrow$). Thus, R' need neither be reflexive nor transitive.

Nothing would have been gained if the definition (10) of R could not be removed afterwards. That means we have to show that $\Phi' \wedge \neg\psi' \wedge (10)$ is satisfiable if and only if $\Phi' \wedge \neg\psi'$ is satisfiable. Since $\Phi' \wedge \neg\psi'$ does not contain R any more, we can always find an interpretation for R, using (10) as definition. Therefore each model for $\Phi' \wedge \neg\psi'$ can be extended to a model for $\Phi' \wedge \neg\psi' \wedge (10)$. Thus, (10) can be eliminated. $\Phi' \wedge \neg\psi'$ is the final result of our transformation.

What has actually happened is that the role of the reflexivity and transitivity of R has been taken over by the reflexivity and transitivity of the implication connective. Many other examples of K-transformations are of a similar kind. The built–in properties of predicate logic take over the role of special properties of non–logical symbols.

The new symbols R' and W we introduced in our reflexivity and transitivity example can have some well known interpretations. If for example R is the subset relation $\subseteq$ then R' is the membership relation $\in$ and (9) is the definition of subset in terms of membership. If R is a binary consequence relation $\vdash$ of a logic, then R' can be interpreted as the satisfiability relation $\models$, and

the sort W denotes the set of worlds. (10) becomes the definition of the consequence relation in terms of the satisfiability relation.

$$\forall x, y \; x \mid\!\!\sim^2 y \Leftrightarrow (\forall w{:}W \; w \models x \Rightarrow w \models y) \tag{11}$$

3.2.1 The General Transformation Procedure

The general procedure for transforming formulae Φ in a consistent way, i.e. without loosing satisfiability or unsatisfiability, consists of the following sequence of steps

$$\Phi \xrightarrow{\text{extension}} \Phi \wedge \text{transformer} \xrightarrow{\text{transformation}} \Phi' \wedge \text{transformer} \xrightarrow{\text{elimination}} \Phi'$$

where 'transformer' is a formula of the kind

$$Left \Leftrightarrow Right. \tag{12}$$

(9) is an example for (12). The 'extension' step involves finding the transformer. In general this is still a creative step. In [OGP94], however, we describe methods for automating this step to a certain extent. The actual transformation is done in the 'transformation' step. In the simplest case the transformation strategy is just definitional replacement where (12) is used as rewrite rule from left to right. It can, however, also be a much more complex combination of rewriting, inferencing and deleting formulae. In the elimination step we delete the transformer. Since removing formulae can turn unsatisfiable formula sets into satisfiable sets, this is also a nontrivial step which has to be justified. To ensure that the transformation is satisfiability preserving, which is sufficient to do theorem proving by refutation, the following *translation lemmas* have to be proved.

- The *extension lemma* proves that satisfiability of Φ implies satisfiability of $\Phi \wedge$ transformer.
- The *transformation lemma* proves that $\Phi \wedge$ transformer is satisfiable if and only if $\Phi' \wedge$ transformer is satisfiable, Where Φ' is the transformed version of Φ.
- The *elimination lemma* proves that satisfiability of Φ' implies satisfiability of $\Phi' \wedge$ transformer.

The structure of formula K-transformations is investigated in more detail in [OGP94]. They can be classified according to certain syntactic criteria such that some of the transformation lemmas can be proved once and for all. The simple schema we have used for the reflexivity and transitivity example turned out to be quite often applicable. The schema is: in order to find a formula K-transformation Υ_φ, choose[1] an equivalence $Left \Leftrightarrow Right_0$ which is provable from φ, and then rename some of the symbols and literals in $Right_0$ to obtain the final transformer $Left \Leftrightarrow Right$.

3.2.2 Optimizing Translations into Predicate Logic

As a non–trivial application of formula K-transformations, I show how translations from nonclassical logics into predicate logic can be manipulated and hopefully optimized. I illustrate the idea with modal logic, but the principles are applicable to other systems as well.

Via their possible worlds semantics, a number of non–classical logics can be translated into predicate logic. For example the semantics of the modal operators

$$x \models \Box P \quad \text{iff} \quad \forall y \; R(x,y) \Rightarrow y \models P \tag{13}$$
$$x \models \Diamond P \quad \text{iff} \quad \exists y \; R(x,y) \wedge y \models P \tag{14}$$

gives rise to the relational translation of propositional modal logic.

$$\begin{aligned} tr_r(P,w) &= P'(w) \qquad P \text{ a predicate symbol} \\ tr_r(\Box\varphi,w) &= \forall v \; R(w,v) \Rightarrow tr_r(\varphi,v) \\ tr_r(\Diamond\varphi,w) &= \exists v \; R(w,v) \wedge tr_r(\varphi,v). \end{aligned}$$

[1] Choosing the right equivalence is still a creative step. In [OGP94] we give some heuristics for finding the equivalences.

The relation R occurs only in the typical patterns

$$\forall v\ R(w,v) \Rightarrow tr_r(\varphi, v)$$
$$\exists v\ R(w,v) \wedge tr_r(\varphi, v)$$

in the translated formulae. Additionaly it may occur in some characteristic axioms axiomatizing properties of R itself. Since this is a quite typical pattern which appears not only in this application, it is worthwhile to look for a K-transformation which eliminates this relation. The transformation we need depends on whether the relation is serial (i.e. $\forall x \exists y\ R(x,y)$ holds) or not. For the serial case the transformer is

$$\forall x, y\ R(x,y) \Leftrightarrow \exists \gamma{:}AF\ y = apply(\gamma, x). \tag{15}$$

For the non–serial case it is a bit more complicated (see [OGP94]). Intuitively, the sort AF denotes the set of *accessibility functions*, i.e. functions mapping worlds to accessible worlds. *apply* is the application function. Therefore we usually write $\gamma(x)$ instead of $apply(\gamma, x)$. With this idea in mind all transformation lemmas can be proved very easily.

From (15) we can prove

$$\forall w\ \forall v\ R(w,v) \Rightarrow \varphi(v) \quad \Leftrightarrow \quad \forall \gamma{:}AF\ \varphi(\gamma(w)) \tag{16}$$
$$\forall w\ \exists v\ R(w,v) \wedge \varphi(v) \quad \Leftrightarrow \quad \exists \gamma{:}AF\ \varphi(\gamma(w)) \tag{17}$$

The transformation strategy is now not just simple definitional replacement with (15). Instead we use the derived equivalences (16) and (17) wherever possible first for rewriting quantifications as a whole.

Applied to the modal logic case, our K-transformation turns the relational translation of modal logic into predicate logic into the functional translation (c.f. [Wal87, Ohl88a, JR88, Her89, AE92, Ohl90, Gas92, Ohl93, Zam89]):

$$\begin{aligned} tr_f(\Box\varphi, w) &= \forall \gamma{:}AF\ tr_f(\varphi, \gamma(w)) \\ tr_f(\Diamond\varphi, w) &= \exists \gamma{:}AF\ tr_f(\varphi, \gamma(w)) \end{aligned}$$

The transformer (15) can be used in exactly the same way for optimizing the treatment of varying–domains in the translation of quantified modal logics. The normal translation rules for the quantifiers in the varying–domain case are

$$\begin{aligned} tr_r(\forall x\ \varphi(x), w) &= \forall x\ Exists(w,x) \Rightarrow tr_r(\varphi(x), w) \\ tr_r(\exists x\ \varphi(x), w) &= \exists x\ Exists(w,x) \wedge tr_r(\varphi(x), w) \end{aligned}$$

where $Exists(w,x)$ intuitively means that x is in the domain of the world w. Since each domain contains at least one element, *Exists* is serial. The transformer (15) now yields an optimized translation

$$\begin{aligned} tr_f(\forall x\ \varphi(x), w) &= \forall \gamma{:}M\ tr_f(\varphi(\gamma(w)), w) \\ tr_f(\exists x\ \varphi(x), w) &= \exists \gamma{:}M\ tr_f(\varphi(\gamma(w)), w). \end{aligned}$$

The sort M[2] denotes the set of functions mapping worlds to their domain elements, i.e. $\gamma(w) \in domain(w)$. Quantification over all γ exhausts the domain of w.

Using formula K-transformations, with a minimum of effort we reconstructed the functional translation for modal logic and extended it to an optimized translation of varying–domain systems. We composed an existing translation, namely the relational translation from modal to predicate logic with the newly defined K-transformation. This is an example for a general method to modify translations. The pattern is:

[2]We choose M to distinguish it from AF in case both transformers are applied simultaneously.

test.

In order to check the faithfulness condition, suppose $R(a,b) \wedge R(b,c) \Rightarrow R(a,c)$ is a ground instance of C.

$$\vec{\forall}\Upsilon_C(R(a,b)) \wedge \vec{\forall}\Upsilon_C(R(b,c)) \Rightarrow \vec{\forall}\Upsilon_C(R(a,c))$$

becomes

$$\begin{array}{l}(R(a,b) \wedge \forall x\ R(x,a) \Rightarrow R(x,b)) \\ \wedge\ (R(b,c) \wedge \forall x\ R(x,b) \Rightarrow R(x,c)) \\ \Rightarrow (R(a,c) \wedge \forall x\ R(x,a) \Rightarrow R(x,c))\end{array}$$

which is in fact a tautology. The transformation is independent of the structure of a, b and c. Therefore the condition holds for all ground instances of C. Thus, Υ_C is sound and complete and no self resolvent of the transitivity clause needs to be considered at all.

Compared to the transformer (9) which transformes both, positive and negative literals, we got a considerable improvement: only the positive literals need to be transformed (of course without renaming R).

In general Υ_C can be obtained by successively computing self resolvents and choosing selected literals until the faithfulness test succeeds.

3.3.1 Transforming the Modus Ponens Clause

In the context of investigating logics, it is interesting to see whether clause K-transformations can improve the performance of automated theorem provers for Hilbert systems. To this end I developed a clause K-transformation for the Modus Ponens clause (4) (also called condensed detachment). Unfortunately a faithful transformer for Modus Ponens consists of *all* self resolvents between the first and third literal.

$$\begin{array}{ll}S_\Upsilon = & \{Th(i(x,y)) \wedge Th(x) \Rightarrow Th(y), \\ & Th(i(x,i(z_1,z_2))) \wedge Th(x) \wedge Th(z_1) \Rightarrow Th(z_2), \\ & \ldots, \\ & Th(i(x,i(z_1,i(\ldots z_i)))) \wedge Th(x) \wedge Th(z_1) \wedge \ldots \wedge Th(z_{i-1}) \Rightarrow Th(z_i), \\ & \ldots\ \}\end{array}$$

The first literal $Th(i(x,i(z_1,i(\ldots z_i))))$ is the selected literal in all clauses. Transforming a clause like (3) yields infinitely many clauses. Fortunately, one can find a finite encoding of these infinitely many clauses with some well known predicate logic tricks. It turned out that in the case of (3) the transformation revealed some significant redundancies caused by the lonely variable s in (3). After removing these redundencies I proved with Otter 3.0 some of the challenging problems discussed in the Automated Reasoning literature. I got the following results [OGP94].

	theorem	original	transformed	improvement
1	$Th(i(x,x))$	1.91	0.11	17.3
2	$Th(i(x,i(y,x)))$	1.94	0.13	14.9
3	$Th(i(i(i(x,y),x),x))$	4.77	0.52	9.2
4	$Th(i(i(x,y),i(i(y,z),i(x,z))))$	2520.77	49.51	50.9
5	$Th(i(x,i(i(x,y),y)))$	35.07	13.75	2.5

The numbers in the third and fourth column give the total CPU time in seconds, Otter 3.0 needed to prove the theorem (on a Solburn machine with Super Sparc processors), once from the original two axioms, and then from the transformed axioms. Not for all the examples I tried, I got these impressive improvements, but the improvement is in general better for complicated theorems than for easy ones.

3.4 Quantifier Elimination

In [GO92a] we have developed an algorithm which can compute for second–order formulae of the kind $\exists P_1, \ldots, P_k\ \Phi$ where Φ is a first–order formula, an equivalent first–order formula — if there is one. Since $\forall P_1, \ldots, P_k\ \Phi \Leftrightarrow \neg\exists P_1, \ldots, P_k\ \neg\Phi$, this algorithm can also be applied to universally

quantified predicate variables. Related methods can also be found in [Ack35a, Ack35b, Ack54, Sza92, BGW92, Sim94].

The definition of the algorithm is:

Definition 3.1 (The SCAN Algorithm)
Input to SCAN is a formula $\alpha = \exists P_1, \ldots, P_n\ \Phi$ with predicate variables $P_1, \ldots, P_n$ and an arbitrary first–order formula Φ.
Output of the SCAN — if it terminates — is a formula φ_α which is *logically equivalent* to α, but not containing the predicate variables $P_1, \ldots, P_n$.
SCAN performs the following three steps:

1. Φ is transformed into clause form.

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'), 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. 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. ◁

The SCAN algorithm is correct in the sense that its result is logically equivalent to the input formula. It cannot be complete, i.e. there may be second–order formulae which have a first–order equivalent, but SCAN cannot find it. An algorithm which is complete in this sense cannot exists, otherwise the theory of arithmetic would be enumerable.

The points where SCAN can fail to compute a first–order equivalent for α are (i) the resolution does not terminate and (ii) reversing Skolemization is not possible. In the second case there is a (again second–order) solution with existentially quantified Skolem functions.

4 Semantics for Extensions of Propositional Logics

Now we come back to the discussion we started in Section 2.2. We show how K-transformations and quantifier elmination can be used for finding a model theoretic semantics.

4.1 From Unary to Binary Consequence Relations

There are different possbilities for axiomatizing a logic in terms of consequence relations. A unary 'consequence relation' $\vdash$, which is actually not a consequence relation, but a theoremhood relation is the simplest possibility. Alternatively one can specify a logic via a binary consequence relation $\vdash^2$. $\varphi \vdash^2 \psi$ means that ψ is derivable from φ, derivable in the so specified logic. If each argument of $\vdash^2$ is not a single formula but a list or a set we have a sequent system. It turned out that a binary consequence relation is the most natural starting point for finding a semantics.

If $\mathcal{A}_0$ is a predicate logic (as meta logic) axiomatization of a logic $\mathcal{L}$ in terms of a unary consequence relation $\vdash$, we can use K-transformations to find a corresponding axiomatization with a binary consequence relation. There must be a distinguished term $\iota(x,y)$ in $\mathcal{A}_0$ which is some sort of implication. The term is distinguished in the sense that all theorems of the form $\vdash \ldots$ provable from $\mathcal{A}_0$ are actually of the form $\vdash \iota(\ldots,\ldots)$. Moreover, the ground proof of these theorems must contain also only atoms of the form $\vdash \iota(\ldots,\ldots)$. In the Łukasiewicz example this distinguished term is just $i(x,y)$ denoting $x \to y$. Only atoms of the form $\vdash i(\ldots,\ldots)$ occur in proofs of the theorems we are interested in. In general, ι need not be a function symbol. It can for example be a term $o(n(x),y)$ encoding $\neg x \vee y$. In most cases, however it will be just an implication symbol.

The following transformer turns unary into binary consequence relations:

$$\vdash \iota(x,y) \Leftrightarrow x \vdash^2 y. \tag{18}$$

This is actually a kind of deduction theorem. In [OGP94] we prove the faithfulness of this transformer.

For example the clauses (3) and (4) become

$$\forall r,p,q,s \qquad i(i(p,q),r) \vdash^2 i(i(r,p),i(s,p)) \tag{19}$$

$$\forall p,q,r,s \qquad p \vdash^2 q \wedge i(p,q) \vdash^2 i(r,s) \Rightarrow r \vdash^2 s. \tag{20}$$

4.2 Transformers for Binary Consequence Relations

In [Ohl94], I have developed a sequence of K-transformations which generate possible worlds semantics from axiomatizations of logics in terms of a reflexive and transitive binary consequence relation and arbitrary n–place connectives.

To start with, we eliminate reflexivity and transitivity of the consequence relation $\vdash^2$. The method for eliminating reflexivity and transitivity has been explained in Section 3.2. The transformer is

$$\mathcal{T}_1: \qquad p \vdash^2 q \Leftrightarrow (\forall w{:}W\ w \models p \Rightarrow w \models q). \tag{21}$$

We interpret the sort W as the set of 'worlds' in a possible worlds semantics and the predicate $\models$ as the satisfiability relation. All further transformations start from the transformed system $\mathcal{T}_1(\mathcal{A}_1)$ where $\mathcal{A}_1$ is the initial Hilbert system. For three further properties of logical connectives f I developed faithful K-transformations. The general schema is almost always the same. The property to be transformed is translated with $\mathcal{T}_1$, corresponding equivalences $Left \Leftrightarrow Right_0$ are derived and then the non–recursive parts in $Right_0$, these are the parts without the original formula variables, are renamed. Sometimes alternative formulations in terms of sets of worlds instead of worlds were possible. The following results were obtained.

Congruence Properties

$$\forall \vec{p},\vec{q}\ \textstyle\bigwedge_i (p_i \vdash^2 q_i \wedge q_i \vdash^2 p_i) \Rightarrow f(\vec{p}) \vdash^2 f(\vec{q}) \tag{22}$$

There are two alternative transformers:

$\mathcal{T}_2$: $\quad w \models f(\vec{p}) \Leftrightarrow \exists \vec{x}{:}S\ N_f(w,\vec{x}) \wedge \bigwedge_i \forall v{:}W\ R^i_f(x_i,v) \Leftrightarrow v \models p_i$ and

$\mathcal{T}_3$: $\quad w \models f(\vec{p}) \Leftrightarrow \exists \vec{X}\ N'_f(w,\vec{X}) \wedge \bigwedge_i X_i = |p_i|$

where $X_i = |p_i|$ is an abbreviation for $\forall v\ v \in X_i \Leftrightarrow v \models p_i$.

Monotonicity Properties

Upward monotonicity: $\forall p, q\ p \mid\!\sim q \Rightarrow \forall \vec{s}, \vec{s}'\ f(\vec{s}, p, \vec{s}') \mid\!\sim f(\vec{s}, q, \vec{s}')$ (23)
Downward monotonicity: $\forall p, q\ p \mid\!\sim q \Rightarrow \forall \vec{s}, \vec{s}'\ f(\vec{s}, q, \vec{s}') \mid\!\sim f(\vec{s}, p, \vec{s}')$ (24)

If $\vec{p}$ are the downward monotonic argument positions and $\vec{q}$ are the upward monotonic argument positions in f then the two alternative transformers for monotonicity are

$\mathcal{T}_4$: $w \models f(\vec{p}, \vec{q}) \Leftrightarrow \forall \vec{x}{:}S\ \bigwedge_i (\forall v{:}W\ R^i_f(x_i, v) \Rightarrow v \models p_i) \Rightarrow$
$\exists \vec{y}{:}S\ \bigwedge_j (\forall v{:}W\ R^j_f(y_j, v) \Rightarrow v \models q_j) \wedge N_f(w, \vec{x}, \vec{y})$ and

$\mathcal{T}_5$: $w \models f(\vec{p}, \vec{q}) \Leftrightarrow \forall \vec{X}\ \vec{X} \subseteq |\vec{p}| \Rightarrow \exists \vec{Y} \vec{Y} \subseteq |\vec{q}| \wedge N'_f(w, \vec{X}, \vec{Y})$.

Closure Properties

For each upward monotonic argument position of f
(to simplify notation we assume this is the last argument position):
$\forall \vec{p}, q_1, q_2, w\ w \mid\!\sim f(\vec{p}, q_1) \wedge w \mid\!\sim f(\vec{p}, q_2) \Rightarrow \exists s\ s \mid\!\sim q_1 \wedge s \mid\!\sim q_2 \wedge w \mid\!\sim f(\vec{p}, s)$, (25)
and for each downward monotonic argument position of f
(to simplify notation we assume this is the first argument position):
$\forall p_1, p_2, \vec{q}, w\ w \mid\!\sim f(p_1, \vec{q}) \wedge w \mid\!\sim f(p_2, \vec{q}) \Rightarrow \exists s\ p_1 \mid\!\sim s \wedge p_2 \mid\!\sim s \wedge w \mid\!\sim f(s, \vec{q})$. (26)
If again $\vec{p}$ are the downward monotonic argument positions and $\vec{q}$ are the upward monotonic argument positions in f then the transformer is

$\mathcal{T}_6$: $w \models f(\vec{p}, \vec{q}) \Leftrightarrow \forall \vec{x}, \vec{y}\ \mathcal{R}_f(w, \vec{x}, \vec{y}) \Rightarrow (\bigwedge_i x_i \models p_i) \Rightarrow (\bigvee_j y_j \models q_j)$

N_f and N'_f are the *neighbourhood relations* and $\mathcal{R}_f$ is the accessibility relation between worlds. We got these new relation symbols by renaming literals $v \models f(\vec{x})$ into $N_f(v, \vec{x})$ during the construction of the transformer.

There are some preconditions for the transformer for the closure properties. The original axiom set must consist of definite Horn clauses, and the neighbourhood relation N'_f introduced by $\mathcal{T}_3$ must not be empty for at least one argument position. These conditions are needed to prove that there is always a model where the closure properties also hold for infinitely many q_i's, and to construct the accessibility relation from the neighbourhood relation.

It is no coincidence that these equivalences look like semantics definitions for the connectives f. In our framework, there is actually no difference between these transformers and what we could call a semantics for a connective. *The transformers are translation rules and semantics definitions at the same time.*

If this general scheme is instantiated for the unary $\Box$–operator in modal logic, we reconstructed in fact the different versions of neighbourhood semantics (minimal model semantics) and Kripke semantics as you can find them in logic textbooks [Che80]. (The details, which are not so simple as it may appear here, can be found in [Ohl94].)

Truth Value Semantics

For a connective f with a semantics defined in terms of the accessibility relation we can prove that f is actually a classical connective by proving that the accessibility relation collapses to a point relation.

A point relation is one where general reflexivity $\forall x\ \mathcal{R}(x, \ldots, x)$ holds and all arguments of $\mathcal{R}$ 'collapse' in the following sense: $\forall \vec{x}\ \mathcal{R}(\vec{x}) \Rightarrow \bigwedge_{i,j} x_i = x_j$ holds. This is a standard predicate logic theorem proving problem which can be solved with an automated theorem prover.

4.3 Simplification by Quantifier Elimination

If the original Hilbert system entails the reflexivity and transitivity of $\models$ and the congruence properties (22) hold for *all* connectives f then the relation $p \approx q \stackrel{\text{def}}{=} (\mathcal{A}_1 \text{ entails } (p \mid\!\sim q \wedge q \mid\!\sim p))$, is a congruence relation on $\mathcal{L}$–terms. For a term s let $[s]$ be its $\approx$–congruence class. $\mathcal{L}_{|\approx} \stackrel{\text{def}}{=} \{[s] \mid s \text{ is an } \mathcal{L}\text{–term}\}$ is called the *Lindenbaum algebra* of $\mathcal{L}$. If $\mathcal{L}$ contains propositional logic, *its Lindenbaum algebra is a Boolean algebra.* We have $p \mid\!\sim q$ iff $[p] \leq [q]$ where $\leq$ is the lattice theoretic smaller

relation of Boolean algebras. By Stone's representation theorem, this Boolean algebra is isomorphic to a field of sets [Sto36]. Each element $[p]$ can therefore be identified with a set $\mathcal{P}([p])$ and we have $p \mid\!\!\sim^2 q$ iff $[p] \leq [q]$ iff $\mathcal{P}([p]) \subseteq \mathcal{P}([q])$ iff $\forall w\ w \in \mathcal{P}([p]) \Rightarrow w \in \mathcal{P}([q])$. By the construction in the proof of Stone's theorem, $\mathcal{P}([p])$ is *the set of prime ideals containing* $[p]$. (In our case these prime ideals correspond to the maximally consistent sets in the usual canonical model constructions.)

Now we have an alternative interpretation of the 'world sort' W: The worlds are the set of prime ideals of $\mathcal{L}_{|\approx}$ and the literals $w \models p$ can be interpreted as $w \in \mathcal{P}([p])$. In [Ohl94] I show that one can even go a step further and interpret literals $w \models p$ as $w \in p$ where p is quantified over the full powerset of the set of prime ideals. That means formula variables p can in fact be taken as second–order predicate variables and instead of $w \models p$ we can write $p(w)$. Transformed Hilbert axioms can now be simplified by negating them first, applying SCAN and negating the result again.

As an example, let us transform and simplify the Hilbert axiom $\Box p \mid\!\!\sim^2 \Box\Box p$ in normal modal logic (the axiom '4'). The transformer $\mathcal{T}_1$ translates this axiom into

$\forall p\ \forall w\ w \models \Box p \Rightarrow w \models \Box\Box p.$

We apply the instance of $\mathcal{T}_6$ for the $\Box$–operator to this formula.

$\forall p\ \forall w\ w(\forall u\ \mathcal{R}(w,u) \Rightarrow p(u)) \Rightarrow (\forall u\ \mathcal{R}(w,u) \Rightarrow (\forall v\ \mathcal{R}(u,v) \Rightarrow p(v))$

In order to simplify this formula we apply the SCAN algorithm. Since p is universally quantified, we have to negate the formula first. The result

$\exists p\ \exists w\ w(\forall u\ \mathcal{R}(w,u) \Rightarrow p(u)) \wedge (\exists u\ \mathcal{R}(w,u) \wedge (\exists v\ \mathcal{R}(u,v) \wedge \neg p(v))$

is the input for SCAN. The clause from of this formula is:

$$\begin{array}{llll} C_1 & \neg\mathcal{R}(w,u) \vee p(u) & C_2 & \mathcal{R}(w,u) \\ C_4 & \neg p(v) & C_3 & \mathcal{R}(u,v) \end{array}$$

u is the only universally quantified variable. There is one resolution possible with literals containg p. The resolvent is $\neg\mathcal{R}(w,v)$. If the existential quantifiers are reconstructed for this clause together with C_2 and C_3, and the formula is negated again, we obtain, as expected, the transitivity of $\mathcal{R}$: $\forall u,v,w\ \mathcal{R}(w,u) \wedge \mathcal{R}(u,v) \Rightarrow \mathcal{R}(w,v)$.

4.4 The Semantics Generation Procedure

We define a procedure for developing the semantics for Hilbert systems.

We start with an arbitrary formula set $\mathcal{A}_1$ containing the $\mid\!\!\sim^2$–predicate. Naturally $\mathcal{A}_1$ should be consistent, otherwise the following steps have no meaning.

Step 1 Prove that $\mid\!\!\sim^2$ is reflexive and transitive.

Step 2 Prove the congruence properties, (22) for all connectives. If there are connectives where this is not possible, the logic is outside the scope of the current theory. If the congruence properties hold for all connectives and $\mid\!\!\sim^2$ is the only predicate in $\mathcal{A}_1$, and all formula variables are universally quantified then both transformers $\mathcal{T}_2$ and $\mathcal{T}_3$ define a sound and complete semantics.

Step 3 Try proving for each connective f the monotonicity properties (23) and (24) from $\mathcal{A}_1$. In this step we determine for each argument position of f, whether it is upward or downward monotonic. We assume that each argument position is monotonic.

The actual semantics is now determined by one of the transformers $\mathcal{T}_4$ or $\mathcal{T}_5$, for each monotonic connective, and one of the previous transformers for the non–monotonic connectives. All can be mixed freely.

Step 4 Check whether $\mathcal{A}_1$ contains an axiomatization of standard propositional logic. Find a model for $\mathcal{A}_1 \wedge (\exists p,q\ \neg(p \mid\!\!\sim^2 q))$ to ensure that the next transformations do not produce inconsistencies. If both conditions are fulfilled, one can simplify the previously transformed axioms with the quantifier elimination algorithm SCAN. Notice that this is an optional simplification step. If the quantifier elimination algorithm does not succeed, then the

original translated formula is the final result of the transformation. In this way we can also deal with so–called incomplete systems. The transformed axioms specify restrictions on the assignment of predicate variables.

Step 5 If the conditions of the previous step are fulfilled, and $\mathcal{A}_1$ consists of definite Horn clauses only, and the neighbourhood relation is not empty for at least one argument position, then check for the closure properties. For the connectives which are either upward or downward monotonic for each argument position, try proving (25) for each upward monotonic argument position and (26) for each downward monotonic argument position. If this is successful then the transformer $\mathcal{T}_6$ yields the classical relational possible worlds semantics. Translate $\mathcal{A}_1$ again using the strongest possible transformer for each connective. As in the previous step, these translated axioms can be simplified using quantifier elimination.

Notice that this step and the corresponding completeness proof subsumes the Sahlquist technique frequently used in modal logic to get frame properties from Hilbert axioms [Sah75, vB84].

Step 6 For connectives with relational semantics (the previous step was successful) try proving that the accessibility relation collapses to a point relation. If this is provable then the connective has a standard truth value semantics.

The completeness results for the K-transformation technique guarantee that a theorem can be proved from the original Hilbert axioms $\mathcal{A}_1$ if and only if the translated theorem can be proved from the resulting translated axiom system.

All steps in this procedure can be fully automated by using automated theorem provers for PL1, model generation algorithms like John Slaney's FINDER [PS90], and the SCAN algorithm.

5 Summary

I have sketched methods for supporting the development and investigation of application oriented logics with computers. In particular we have considered the problems of

- reasoning in Hilbert systems.
- finding model theoretic semantics for axiomatically specified logics, and vice versa
- finding corresponding axioms for semantic properties,
- investigating the expressiveness of logics,
- finding translations from the object logic to predicate logic.

Using predicate logic as meta logic, we were able to map these problems to formula manipulation problems in PL1. With automated theorem provers, a quantifier elimination algorithm, and the special technique of K-transformations we found partially automatable solutions of these problems. As an example, we were able to reconstruct the various semantics versions for modal logic, but for arbitrary n–place connectives, and with very general completeness results for computing corresponding frame conditions for Hilbert axioms.

This work can, and needs to be extended in so many directions, that it is impossible to name them all here. Of course, automated support for the development of calculi for a given logic would be very welcome. Serious limitations, which have to be overcome, are the restriction to propositional logics. What we have not yet considered at all are many–valued logics, probabilistic logics and nonmonotonic logics. Conditional logics, however should be within the scope of the current techniques.

Although in this paper we considered only specifications of logics, the basic ideas and techniques are independent of this particular application. There might be other areas where similar manipulations of logical specifications also yield interesting results. In particular, the area of representation theorems for algebraic systems is related. Transferring the ideas and methods to algebraic logic and to general algebra seems promising.

References

[Ack35a] Wilhelm Ackermann. Untersuchung über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[Ack35b] Wilhelm Ackermann. Zum Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 111:61–63, 1935.

[Ack54] Wilhelm Ackermann. *Solvable Cases of the Decision Problem.* North–Holland Pu. Co., 1954.

[AE92] Yves Auffray and Patrice Enjalbert. Modal theorem proving: An equational viewpoint. *Journal of Logic and Computation*, 2(3):247–297, 1992.

[BFOZ93] Matthias Baaz, Christian G. Fermüller, Arie Ovrutcki, and Richard Zach. MULTLOG: A system for axiomatizing many-valued logics. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning, Proceedings of LPAR 93, Lecture Notes in AI 698*, pages 345–347. Springer Verlag, 1993.

[BG90] Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In *CADE-10: 10th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 427–441, Kaiserslautern, FRG, 1990. Springer-Verlag. Copy filed.

[BGO94] Chris Brink, Dov Gabbay, and Hans Jürgen Ohlbach. Towards automating duality. *Journal of Computers and Mathematics with Applications*, 1994. Forthcoming in a special issue. A longer version appeared as a technical report MPI-I-93-220 of the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[BGW92] 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.

[Bra75] Daniel Brand. Proving theorems with the modification method. *SIAM Journal on Computing*, 4(4):412–430, 1975.

[Che80] B. F. Chellas. *Modal Logic: An Introduction.* Cambridge University Press, Cambridge, 1980.

[Gas92] Olivier Gasquet. Deduction for multimodal logics. In *Proc. of Applied Logic Conference (Logic at Work). Amsterdam*, December 1992.

[GO92a] Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second–order predicate logic. *South African Computer Journal*, 7:35–43, July 1992. also published in [GO92b].

[GO92b] Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second–order predicate logic. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Principles of Knowledge Representation and Reasoning (KR92)*, pages 425–435. Morgan Kaufmann, 1992. also published as a technical report MPI-I-92-231 of the Max-Planck-Institut für Informatik, Saarbrücken and in the South African Computer Journal, 1992.

[Her89] Andreas Herzig. *Raisonnement automatique en logique modale et algorithmes d'unification.* PhD thesis, Université Paul-Sabatier, Toulouse, 1989.

[JR88] Peter Jackson and Han Reichgelt. A general proof method for modal predicate logic without the Barcan Formula or its converse. DAI Research Report 370, Department of Artificial Intelligence, University of Edinburgh, 1988.

[Kri59] S. A. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–14, 1959.

[Kri63] S. A. Kripke. Semantical analysis of modal logic i, normal propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[Łuk70] J. Łukasiewicz. *Selected Works.* North Holland, 1970. Edited by L. Borkowski.

[McC89] William W. McCune. *OTTER User's Guide.* Mathematical and Computer Science Devision, Argonne National Laboratory, april 1989.

[McC90] William McCune. OTTER 2.0. In Mark Stickel, editor, *Proc. of* 10^{th} *Internation Conference on Automated Deduction, LNAI 449*, pages 663–664. Springer Verlag, 1990.

[MW92] Williman McCune and Larry Wos. Experiments in automated deduction with condensed detachment. In Deepak Kapur, editor, *Autmated Deduction - CADE 11, Lecture Notes in AI, vol. 607*, pages 209–223. Springer Verlag, 1992.

[Non93] Andreas Nonnengart. First-order modal logic theorem proving and functional simulation. In Ruzena Bajcsy, editor, *Proceedings of the 13th IJCAI*, volume 1, pages 80 – 85. Morgan Kaufmann Publishers, 1993.

[OGP94] Hans Jürgen Ohlbach, Dov Gabbay, and David Plaisted. Killer transformations. Technical Report MPI-I-94-226, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1994. To be published in Proc. of the 1993 Workshop on Proof Theory in Modal Logic, Hamburg.

[Ohl88a] Hans Jürgen Ohlbach. A resolution calculus for modal logics. In Ewing Lusk and Ross Overbeek, editors, *Proc. of* 9^{th} *International Conference on Automated Deduction, CADE-88 Argonne, IL*, volume 310 of *Lecture Notes in Computer Science*, pages 500–516, Berlin, Heidelberg, New York, 1988. Springer-Verlag. Extended version appeared in [Ohl88b].

[Ohl88b] Hans Jürgen Ohlbach. A resolution calculus for modal logics. SEKI Report SR-88-08, FB Informatik, Universität Kaiserslautern, Germany, 1988. PhD Thesis, short version appeared in [Ohl88a].

[Ohl90] Hans Jürgen Ohlbach. Semantics based translation methods for modal logics. SEKI Report SR-90-11, FB. Informatik, Univ. of Kaiserslautern, 1990. Finally published in [Ohl91].

[Ohl91] Hans Jürgen Ohlbach. Semantics based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.

[Ohl93] Hans Jürgen Ohlbach. Optimized translation of multi modal logic into predicate logic. In Andrei Voronkov, editor, *Proc. of Logic Programming and Automated Reasoning (LPAR)*, volume 698 of *Lecture Notes in Artificial Intelligence*, pages 253–264. Springer Verlag, 1993.

[Ohl94] Hans Jürgen Ohlbach. Synthesizing semantics for extensions of propositional logic. Technical Report MPI-I-94-225, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1994.

[PS90] P. Pritchard and J. Slaney. Computing models of propositional logics. In *10th International Conference on Automated Deduction, CADE-10, LNCS 449*, page 685. Springer Verlag, 1990.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery (JACM)*, 12(1):23–41, 1965.

[Sah75] H. Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logics. In S. Kanger, editor, *Proceedings of the 3rd Scandinavian Logic Symposium, 1973*, pages 110–143, Amsterdam, 1975. North Holland.

[Sim94] Harold Simmons. The monotonous elimination of predicate variables. *Journal of Logic and Computation*, 4(1), 1994.

[Sto36] M. H. Stone. The theory of representations for boolean algebras. *Transactions of American Mathematical Society*, 40:37–111, 1936.

[Sza92] 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.

[vB84] 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.

[Wal87] Lincoln A. Wallen. Matrix proof methods for modal logics. In *Proc. of* 10^{th} *IJCAI*, pages 917–923. IJCAI, Morgan Kaufmann Publishers, 1987.

[Zam89] N.K. Zamov. Modal resolutions. *Izvestiya VUZ. Mathematika*, 33(9):22–29, 1989. Also published in Soviet Mathematics, Allerton Press.

Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server **ftp.mpi-sb.mpg.de** under the directory pub/papers/reports. If you have any questions concerning ftp access, please contact **reports@mpi-sb.mpg.de**. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Regina Kraemer
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: **kraemer@mpi-sb.mpg.de**

MPI-I-94-226	H. J. Ohlbach, D. Gabbay, D. Plaisted	Killer Transformations
MPI-I-94-225	H. J. Ohlbach	Synthesizing Semantics for Extensions of Propositional Logic
MPI-I-94-224	H. Aït-Kaci, M. Hanus, J. J. M. Navarro	Integration of Declarative Paradigms Proceedings of the ICLP'94 Post-Conference Workshop Santa Margherita Ligure, Italy
MPI-I-94-223	D. M. Gabbay	LDS – Labelled Deductive Systems Volume 1 — Foundations
MPI-I-94-218	D. A. Basin	Logic Frameworks for Logic Programs
MPI-I-94-216	P. Barth	Linear 0-1 Inequalities and Extended Clauses
MPI-I-94-209	D. A. Basin, T. Walsh	Termination Orderings for Rippling
MPI-I-94-208	M. Jägér	A probabilistic extension of terminological logics
MPI-I-94-207	A. Bockmayr	Cutting planes in constraint logic programming
MPI-I-94-201	M. Hanus	The Integration of Functions into Logic Programming: A Survey
MPI-I-93-267	L. Bachmair, H. Ganzinger	Associative–Commutative Superposition
MPI-I-93-265	W. Charatonik, L. Pacholski	Negativ set constraints: an easy proof of decidability
MPI-I-93-264	Y. Dimopoulos, A. Torres	Graph theoretical structures in logic programs and default theories
MPI-I-93-260	D. Cvetković	The logic of preference and decision supporting systems
MPI-I-93-257	J. Stuber	Computing Stable Models by Program Transformation
MPI-I-93-256	P. Johann, R. Socher	Solving simplifications ordering constraints
MPI-I-93-250	L. Bachmair, H. Ganzinger	Ordered Chaining for Total Orderings
MPI-I-93-249	L. Bachmair, H. Ganzinger	Rewrite Techniques for Transitive Relations
MPI-I-93-243	S. Antoy, R. Echahed, M. Hanus	A needed narrowing strategy
MPI-I-93-237	R. Socher-Ambrosius	A Refined Version of General E-Unification