Computer Support for the Development and Investigation of Logics

Hans Jürgen Ohlbach, *Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany.*
E-mail: ohlbach@mpi-sb.mpg.de

Abstract

The development and investigation of application–oriented logics comprises many aspects and problems. For a few of them some computer support is possible which frees the investigator from sometimes quite complex computations.

This paper gives an overview about some developments in this area. In particular, we consider the correspondences between axiomatic and semantic specifications of a logic and the problem of finding one from the other by means of automated theorem provers and quantifier elimination algorithms. Other topics adressed in this paper are reasoning in Hilbert systems, the investigation of the expressiveness of a logic and the axiomatizability of semantic conditions.

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

1 Introduction

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 many practical applications.

There is a tradeoff between expressiveness of a logic and the complexity of the reasoning algorithms for this logic. Therefore for obtaining an optimal result in a particular application, one has 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 give more or less automated support and guidance for developing new logics.

Although this is a very ambitious goal, there is some evidence that it is in fact possible, at least to a certain extent. MULTLOG [4] for example 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.

This paper is a survey of our advances in providing automated support for logical investigations. Considered are:

- reasoning in Hilbert systems,
- finding model-theoretic semantics for axiomatically specified logics, and vice versa,
- finding axioms corresponding to a given semantics, and

- investigating the expressiveness of logics.

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.

In Section 2 it is explained how reasoning in Hilbert systems can be realized with automated theorem provers and how the proof search can be improved by transforming the axioms in a certain way. In Section 4 we consider the correspondence problem, i.e. the problem of finding corresponding frame properties for given Hilbert axioms. This problem is reduced to a quantifier elimination problem. A particular quantifier elimination algorithm for predicate variables and its implementation is presented. The inverse problem of finding Hilbert axioms for semantic properties is discussed in section 5. The correlation between an axiomatic presentation of a logic and a semantic specification is not straightforward. Correspondence between Hilbert axioms and semantic properties is only one aspect, and it does not always guarantee completeness of the semantics with respect to the axioms. We discuss these problems in an algebraic context in Section 3. Finally in Section 6 a new approach for investigating the expressiveness of a logic and simplifying its semantics is presented. In this approach this problem is reduced to the redundancy problem of certain predicate logic axioms.

In this paper we want to convey the main ideas and results leaving out the technical details and proofs.

2 Reasoning in Hilbert Systems

The most abstract way a logic can be defined 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 theorems. The rules specify how to derive new theorems from the initial theorems and the previously derived ones. For example the axiom

$$\vdash ((p \to q) \to r) \to ((r \to p) \to (s \to p)) \tag{2.1}$$

together with Modus Ponens: $$\text{from } \vdash p \to q \text{ and } \vdash p \text{ infer } \vdash q \tag{2.2}$$

specify the implicational fragment of propositional logic [19, p. 295]. For encoding vague notions, like "knows", "believes" and "wants", a Hilbert-style axiomatization is usually the preferred method because in Hilbert systems their properties can be expressed in a very abstract and intuitive way.

A Hilbert system is a forward reasoning calculus. Starting with the axioms as the initial theorems, with the rules of the Hilbert system new theorems can be derived. Verifying that a given formula is in fact a theorem in the system amounts to enumerating all theorems until eventually the formula appears as one of the derived theorems. 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 (2.1) and (2.2) can be encoded as the predicate logic clauses

$$\forall p,q,r,s \qquad Th(i(i(i(p,q),r),i(i(r,p),i(s,p)))) \tag{2.3}$$

$$\forall p,q \qquad Th(i(p,q)) \wedge Th(p) \Rightarrow Th(q) \tag{2.4}$$

'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 [22].

Hilbert systems as we find them in logic textbooks usually specify each aspect of the logic explicitly. For example, if a binary connective is supposed to be commutative, there is usually an axiom expressing the commutativity 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 as in the above case (2.1), 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 below, can improve the behaviour of automated theorem provers for these more complex systems.

2.1 Clause K-Transformations

Clauses like Modus Ponens (2.4) are *self–resolving.* This means there are resolvents between renamed copies of the clause. Such self–resolving clauses generate a very large search space in an automated theorem prover. Big search spaces can be controlled either with strategies and heuristics for eliminating redundant parts and searching through the remaining parts in a hopefully promising way, or by reformulating the problem. Reformulating the problem in this case means changing the axiomatization in such a way that the same theorems can still be proved, but hopefully in less time.

For the Hilbert system examples we are, in particular, interested in changing or even deleting the clause responsible for the big search space, the Modus Ponens clause. To this end we have been investigating the general problem: given a particular clause C in a set Φ of clauses, is it possible to find a transformation $\Upsilon_C(\Phi)$ of Φ such that the clause C becomes superfluous in Φ and can be deleted *without changing the set of provable theorems*? That means φ is provable from Φ if and only if $\Upsilon_C(\varphi)$ is provable from $\Upsilon_C(\Phi)$. Let us call this a *faithful* transformation.

A first idea for a transformation that makes C superfluous comes from the following consideration. A clause $C = A_1 \wedge \ldots \wedge A_n \Rightarrow B$ can be seen as a procedure that, given instances of the antecedent literals A_i as input, computes the appropriate instances of B as output. Well, if the only thing, we were interested in, is in producing B from the A_i, an alternative way to obtain this 'output' would be to ensure that the transformed 'input' $\bigwedge_i \Upsilon_C(A_i)$ implies B, i.e. $\bigwedge_i \Upsilon_C(A_i) \Rightarrow B$ is a tautology. If we transform Φ such that instead, or in addition to A_i itself, $\Upsilon_C(A_i)$ is available, we do not need C anymore.

Unfortunately it may be the case (and necessary) that the 'output' B of the clause C needs to be used as 'input' for C again. This is typical for self–resolving clauses C where a resolvent B with C may be resolvable with the same clause C again. But after eliminating C, B will not be transformed. Thus, the 'output' B is not available anymore as input. Therefore our first guess for a condition on Υ_C has to be strengthened: We must ensure that the transformed 'input' implies the *transformed* 'output' $\Upsilon_C(B)$. That means $\bigwedge_i \Upsilon_C(A_i) \Rightarrow \Upsilon_C(B)$ must be a tautology.

On the technical level this condition must be refined because for non-ground clauses where variables may become instantiated we cannot just transform clauses by transforming their literals separately. Instead of this we must require that *for all ground instances* $C\rho$ of C:

$$\bigwedge_{i=1}^n \vec{\forall}\Upsilon_C(A_i\rho) \Rightarrow \bigvee_{j=1}^m \vec{\forall}\Upsilon_C(B_j\rho)$$

is a tautology. $\vec{\forall}\Upsilon_C(\ldots)$ means that all variables introduced by Υ_C are universally quantified.

This is the main faithfulness criterion, Υ_C must fulfill. The other more technical criteria and the faithfulness proof can be found in [25]. The faithfulness criterion requires a test for, in general, infinitely many ground instances, and it does not yet say, *how to find* Υ_C. Fortunately it is possible to derive a procedure for finding Υ_C and for testing the faithfulness criterion. The basic idea is: Υ_C is represented by a set S_{Υ_C} of *self–resolvents* of C (including C itself) where for each self–resolvent D a single literal L_D is selected. Υ_C then transforms a clause set Φ by adding the resolvents between the clauses E in Φ and the clauses D in S_{Υ_C} to Φ, but choosing only L_D as the resolution literal in D.

As an example, consider the transitivity clause $C = R(x,y) \wedge R(y,z) \Rightarrow R(x,z)$. We choose $S_{\Upsilon_C} = \{C\}$. (It turns out that no self–resolvent at all is necessary in this case.) As selected literal L_C we may choose either of the two antecedent literals. Lets choose the second one $L_C = R(y,z)$. The transformation Υ_C specified herewith transforms a clause set by adding for each clause containing a *positive* literal $R(s,t)$ the resolvent with C where the second literal is the resolution literal. For example, $\Upsilon_C(R(a,b)) = (R(a,b) \wedge (\forall x\ R(x,a) \Rightarrow R(x,b)))$. It is important to notice that the literals in the resolvents which originate from C itself are never again resolved with C. Therefore this is a terminating procedure.

Notice that in the presence of the reflexivity axiom $R(x,x)$, the original literal $R(a,b)$ can be derived from $\forall x\ R(x,a) \Rightarrow R(x,b)$ and the reflexivity clause. Therefore the original clauses in Φ can be deleted and only the transformed clauses, and of course also the reflexivity clause, need to be kept. This gives precisely the transformation, Brand used in his modification method to eliminate the transitivity of the equality predicate [8].

The important question is now, whether this transformation is faithful. One can show that the selected literals determine a sufficient set of ground instances of C to be tested in the faithfulness 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.

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

is

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

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.

In general Υ_C can be obtained by successively computing self–resolvents and choosing selected literals until the faithfulness test succeeds. We have a prototypic PROLOG implementation of this search process. The program was implemented by Jan Smaus and can find many of the finitely representable transformations.

2.1.1 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. Unfortunately, a faithful transformer for Modus Ponens consists of *all* self–resolvents between the first and the third literal.

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

The first literal $Th(i(x,i(z_1,i(\ldots z_i)))$ is the selected literal in the clause i. Transforming a clause like (2.3) now means adding all resolvents between (2.3) and the first literal of the clauses in S_Υ, which yields infinitely many clauses. Fortunately, with some well known predicate logic tricks one can find a finite encoding[1] of these infinitely many clauses [25]:

$$\begin{array}{ll} Th(i(i(i(x,y),z),i(i(z,x),i(u,x)))) & q_2(x) \Rightarrow q_3(x)\\ Th(i(i(x,y),z) \Rightarrow q_1(x,z) & q_3(i(x,y)) \wedge Th(x) \Rightarrow q_4(y)\\ q_1(x,z) \Rightarrow Th(i(i(z,x),i(u,x))) & q_4(x) \Rightarrow q_3(x)\\ q_1(x,z) \wedge Th(i(z,x)) \Rightarrow q_2(x) & q_4(x) \Rightarrow Th(x)\\ q_2(x) \Rightarrow Th(x) & \end{array}$$

The auxiliary predicates q_i have been introduced for abbreviating repeatetly occurring terms. The infinitely many clauses are finitely encoded with a cycle involving the predicates q_3, q_4 and Th. This way the cycle represented by Modus Ponens reappears, but, as the experiments show, it is less dangerous. I proved from the transformed axioms some of the challenging problems discussed in the Automated Reasoning literature. The following results [25] were obtained with the theorem prover Otter 3.0.

[1] A small Prolog program that performs this transformation is available as a gzipped tar file by anonymous ftp from ftp.mpi-sb.mpg.de, file /pub/guide/staff/ohlbach/modus-ponens.tar.gz.

	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 that Otter needed to prove the given theorem (on a Solburn machine with Super Sparc processors), first from the original two axioms, and then from the transformed axioms. For other examples of Hilbert systems we observed mixed results. There are examples where a proof was found only in the transformed system and there are examples where the transformed system behaves much worse than the original system. This means this transformation is a heuristic which may help or which may not.

3 Semantics for Hilbert Systems

The specification of a logic as a Hilbert system is quite intuitive, but for many purposes this presentation is not adequate. One reason for developing an alternative to Hilbert systems is to improve the efficiency of the calculus. Another reason is to understand the logic better by bringing properties to the surface which are sometimes very deeply hidden. One 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 [17, 18].

For example consider the different versions of the semantics for 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. The classical connectives are interpreted 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 [10]). 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, namely $\vdash p \Leftrightarrow q$ implies $\vdash \Box p \Leftrightarrow \Box q$, but no stronger axioms or rules. In a stronger version of 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: $\vdash p \Rightarrow q$ implies $\vdash \Box p \Rightarrow \Box q$. The next stage in strengthening the semantics is the well known Kripke semantics determined by 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 in the hierarchy 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 properties*) 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 $\vdash \Box(p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)$ and the Necessitation rule: from $\vdash p$ infer $\vdash \Box p$. The reflexivity condition on the accessibility relation validates the axiom $\vdash \Box p \Rightarrow p$.

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.

Since finding an appropriate semantics for a given Hilbert system is essential for understanding the logic and finding better calculi, the question is, can we give some computer support for solving this usually quite hard problem? Here is a first idea for supporting the search for the second part of the semantics, the frame properties. Let us consider as an example again a normal modal logic with standard Kripke semantics and the axiom T: $\vdash \Box p \Rightarrow p$. This axiom is taken to be true for all formulae p in all worlds w, i.e.

$$\forall p\ \forall w\ w \models \Box p \Rightarrow p. \tag{3.1}$$

Taking the semantic definitions for $\Rightarrow$

$$w \models p \Rightarrow q \text{ iff } w \models p \Rightarrow w \models q \tag{3.2}$$

and for $\Box$

$$w \models \Box p \text{ iff } \forall v\ R(w, v) \Rightarrow v \models p \tag{3.3}$$

as rewrite rules for (3.1) yields

$$\forall p\ \forall w\ (\forall v\ R(w, v) \Rightarrow v \models p) \Rightarrow w \models p \tag{3.4}$$

If it was possible to interpret the quantifier $\forall p$ which quantifies over all formulae as a quantifier over all subsets of the possible worlds and to write $p(v)$ instead of $v \models p$ then (3.4) becomes the second–order formula

$$\forall p\ \forall w\ (\forall v\ R(w, v) \Rightarrow p(v)) \Rightarrow p(w). \tag{3.5}$$

This can be proved to be equivalent to $\forall w\ R(w, w)$. And in fact reflexivity is the frame property corresponding to the T-axiom. (3.5) is the *second–order translation* SO of the T-axiom (cf. [31]).

In general, the way the second–order translation of a Hilbert axiom $\vdash \varphi$ is defined varies for different logics. First of all it depends on the truth condition in the given logic. In modal logic as well as in many other logics, a formula is considered a theorem if it is true in *all* worlds. In this case $\vdash \varphi$ is first turned into a formula $\forall w\ w \models \varphi$. If theoremhood means truth in particular worlds or even only in a single world then this first step has to be changed in an appropriate way. In the second step the semantic definitions of the connectives are used exhaustively as rewrite rules for literals of the form $v \models \psi$. Finally in the third step, the formula variables are turned into universally quantified predicate variables. Literals $v \models p$ are turned into literals $p(v)$. In logics with assignment restrictions, like for example in intuitionistic logic, the assignment restriction has to be added as an antecedent in an implication. That means, for example, instead of $\forall p\ \varphi$ we get a formula of the kind $\forall p\ restr(p) \Rightarrow \varphi$ where $restr(p)$ captures the assignment restriction. In intuitionistic logic this is the formula $\forall u, v\ p(u) \wedge u \leq v \Rightarrow p(v)$ where $\leq$ is the accessibility relation in intuitionistic frames.

The second–order translation of Hilbert axioms specifies a class of frames or a class of models. It is *sound* if all formulae provable from the Hilbert axioms are true in this class of frames or models respectively, and it is *complete* (to be be more precise, *weakly complete*) if all formulae which are true in this class are provable from the Hilbert axioms and rules.

Now we have two problems: (i) when is it possible to turn a quantification over formulae into a quantification over predicate variables, (which is the same as a quantification over subsets of the set of worlds), and (ii) given a second–order formula like (3.5), can we mechanically compute the corresponding first–order equivalent, if there is some?

3.1 Basic Semantics and Representation Theorems

Let us first concentrate on problem (i) which is in fact a very deep mathematical problem. This problem may become clearer if we formulate it algebraically. It is well known that by the Lindenbaum–Tarski construction in many cases a logic corresponds to a certain algebra [7]. This algebra is usually given by a set of congruence classes of equivalent formulae. For extensions of propositional logic, for example, with operators that are invariant under equivalences, we can define a congruence relation $\sim$ on formulae by

$$p \sim q \text{ iff } \vdash p \Rightarrow q \text{ and } \vdash q \Rightarrow p \tag{3.6}$$

and factorize the set of all formulae with $\sim$. An n–place logical connective f' is associated with a function f on the congruence classes $[p]$, defined by $f([p_1], \ldots, [p_n]) \stackrel{\text{def}}{=} [f'(p_1, \ldots, p_n)]$. The classical logical connectives are associated with functions (operators) in Boolean algebras: $\wedge$ is associated to $\sqcap$ (meet), $\vee$ is associated to $\sqcup$ (join), $\neg$ is associated to $'$ (inverse). Now we have an algebra in the usual sense, except that its elements are congruence classes of formulae. In the case of normal modal logics, the Lindendbaum–Tarksi algebra is a Boolean algebra with one operator, which is traditionally associated with the $\Diamond$–operator.

Since in most logics there is explicitly or implicitly some kind of reflexive transitive binary consequence relation $\vdash'$ between formulae, the corresponding algebras are usually lattices where the consequence relation corresponds to the $\leq$ relation between the lattice elements. There is a one-to-one correspondence between the Hilbert axioms of the logic and equations in the Lindenbaum-Tarski algebra. As an example, consider the T-axiom $\vdash \Box p \Rightarrow p$. An equivalent formulation with the $\Diamond$–operator is $\vdash p \Rightarrow \Diamond p$.

Since $\Rightarrow$ is classical implication, we get a corresponding formulation in the lattice language: $p \leq f(p)$ where f corresponds to $\Diamond$. Equivalent to this is the equation $\forall p\ (p \sqcap f(p)) = p$. In the sequel we assume that instead of Hilbert axioms, we have a set Σ of equations specifying the class of algebras corresponding to the given logic.

One of the fundamental achievements in lattice theory is the discovery of set representations, i.e. isomorphisms between elements of the lattice and certain subsets of some basic set. For example, in the case of Boolean algebras (without operators) there is the famous Stone–isomorphism [29] between an element x of the algebra $\mathcal{A}$ and the set of ultrafilters[2] in $\mathcal{A}$ containing x. Moreover, this representation maps the Boolean algebra function $\sqcap$ (meet) to $\cap$, $\sqcup$ (join) to $\cup$, $'$ (inverse) to the complement and $\leq$ to the subset relation on sets.

Notice that if the Boolean algebra is in fact the Lindenbaum–Tarski algebra of some logic, then its ultrafilters correspond to the maximally consistent sets of formulae used in the canonical model constructions and these in turn are the worlds in the canonical frame. Furthermore, there is the straightforward correspondence between the logical connectives (e.g. $\wedge$), the functions in the algebra (e.g. $\sqcap$) and the operations on sets (e.g. $\cap$).

The general setting is now: We are given a lattice $\mathcal{A}$ (without extra functions), and we are given an isomorphism ξ that maps an element x of the lattice to some set $\xi(x)$ and the basic lattice functions g to functions $\xi(g)$ operating on the sets in $\xi(\mathcal{A})$. (From distributive lattices upward in the hierarchy of lattics, the basic lattice functions are mapped to the usual set operations $\cap$, $\cup$, complement and $\subseteq$. Without distributivity of meet and join, the join $\sqcup$ cannot be mapped to $\cup$.) Then we add some extra operations, for example to a Boolean algebra we may add a function f corresponding to the modal $\Diamond$–operator, and we know that some set Σ of equations hold for f. f is defined for all elements of $\mathcal{A}$ and therefore, by the isomorphism, $f_\xi \stackrel{\text{def}}{=} \xi(f)$ is defined for all *representable* sets $\xi(x)$. The definition of f_ξ for representable sets is simply $f_\xi(\xi(x)) \stackrel{\text{def}}{=} \xi(f(x))$. The main question is now, can we extend f_ξ to some function f_ξ^+ that is defined for all subsets X in $\xi(\mathcal{A})$, and not only for those which are ξ–images of some x in $\mathcal{A}$? The function f_ξ^+ must in some sense interpolate f_ξ for the non-representable sets. To be useful, this extension f_ξ^+ must satisfy two conditions (a) f_ξ^+ must agree with f_ξ on the representable sets, i.e. $f_\xi^+(\xi(x)) = f_\xi(\xi(x))$ for all x in $\mathcal{A}$ (*soundness*), and (b) the equations in Σ which hold for f must hold for f_ξ^+ as well (*completeness*)[3].

Jónsson and Tarski have investigated this problem for Boolean algebras with operators [15]. It turns out that under certain conditions a definition for f_ξ^+ can be found which at least satisfies condition (a) and is therefore sound. If f is one-place (the extension to n–place functions is straightforward), *normal*, i.e. $f(0) = 0$, and

[2]A *filter* F in a lattice $\mathcal{A}$ is a subset of $\mathcal{A}$ which is upwards closed, i.e. if $x \in F$ and $x \leq y$ then $y \in F$, and closed under meet, i.e. $x \sqcap y \in F$ iff $x \in F$ and $y \in F$. An ultrafilter is a maximal filter. In the case of Boolean algebras an ultrafilter is a filter which is closed under join, i.e. $x \sqcup y \in F$ iff $x \in F$ or $y \in F$, and F either contains x or the the inverse x' for each $x \in \mathcal{A}$.

[3]This notion of completeness corresponds to the notion of weak completeness on the logical side. It means that if a formula φ holds in all frames of the class characterized by Σ then φ is provable in the Hilbert system corresponding to Σ.

additive, that means $f(x \sqcup y) = f(x) \sqcup f(y)$ holds[4], then

$$x \in f_\xi^+(X) \text{ iff } \exists y\ y \in X \wedge \forall Z_\xi\ y \in Z_\xi \Rightarrow x \in f_\xi(Z_\xi) \tag{3.7}$$

turns out to be a suitable definition for f_ξ^+. Here Z_ξ means that Z_ξ is a representable set, i.e. $Z_\xi = \xi(z)$ for some z in $\mathcal{A}$, such that $f_\xi(Z_\xi)$ is defined.

The second part of the conjunction in (3.7) depends only on x and y. If we abbreviate this conjunct by $R(x, y)$ we can write (3.7) as

$$x \in f_\xi^+(X) \text{ iff } \exists y\ y \in X \wedge R(x, y). \tag{3.8}$$

This is in fact the algebraic counterpart of the semantics definition for the modal $\diamond$–operator and R is the accessibility relation.

ξ maps elements x of the algebra $\mathcal{A}$ to the set of ultrafilters containing x. This means for an ultrafilter y and an element x we have

$$y \in \xi(x) \text{ iff } x \in y.$$

This allows us to reformulate the definition of R and to bring it into a more convenient form:

$$\begin{aligned} R(x,y) &\Leftrightarrow \forall Z_\xi\ y \in Z_\xi \Rightarrow x \in f_\xi(Z_\xi) \\ &\Leftrightarrow \forall z\ y \in \xi(z) \Rightarrow x \in \xi(f(z)) \\ &\Leftrightarrow \forall z\ z \in y \Rightarrow f(z) \in x. \end{aligned}$$

If f is again the modal $\diamond$–operator, this is the definition of R in the usual canonical model construction, and this brings us back to the modal logic level.

We have not yet checked whether f_ξ^+ as defined in (3.7) satisfies the equations in Σ. If this can be guaranteed, we are licensed to assume that the algebra $\xi(\mathcal{A})$ is in fact the full power set algebra and a quantification over all the elements of $\mathcal{A}$ is equivalent to a quantificiation over all subsets of the domain of $\xi(\mathcal{A})$. This is just what we wanted to achieve in order to solve Problem (i), replacing quantifiers over formula variables with quantifiers over predicate variables.

The question when equations Σ holding for f continue to hold for f_ξ^+ has been investigated and various *preservation theorems* for certain classes of equations have been proved [14]. For the case of modal algebras, i.e. Boolean algebras corresponding to modal logics, the results of correspondence theory [31, 32, 6] and in particular the Sahlqvist theorem [27] guarantee that Σ holds for f_ξ^+ if the axioms in Σ have a particular syntactic form.

In general there is no satisfactory solution so far. There is no unique way to get a definition of f_ξ^+ and there is no general result which characterizes precisely which equations Σ are preserved. From a logical point of view this means there is no unique way to get a basic semantics for a logic and there is no guarantee that given a basic semantics, the second–order translation of the Hilbert axioms specifies a class of frames or models which is complete. The model class specified by the second–order translation may be too restricted. Formulae may be true in all models of this class, but they may not be provable from the axioms.

[4]normality and additivity of the function f correspond to the presence of the necessitation rule and the K-axiom in the corresponding modal logic. Therefore Boolean algebras with operators correspond to *normal* modal logics in the sense of Chellas [10].

4 Computing Frame Properties by Quantifier Elimination

Often a class of logics (like the class of normal modal logics) has one basic semantics and the different members of the class are determined by different frame properties. As we have seen, we can obtain the corresponding frame property for a given Hilbert axiom by translating the Hilbert axiom into a second–order formula and by then trying to find an equivalent first–order formula. This method guarantees at least soundness. Completeness has to be checked separately.

In this section we consider the problem: given a formula $\exists P_1, \ldots, P_k\ \Phi$ where Φ is a first–order formula and P_i are the predicate variables occurring in Φ, find a first–order formula Φ' such that $(\exists P_1, \ldots, P_k\ \Phi)\ \Leftrightarrow \Phi'$.

Since $\forall P_1, \ldots, P_k\ \Phi \Leftrightarrow \neg\exists P_1, \ldots, P_k\ \neg\Phi$, an algorithm that solves the problem for existentially quantified predicate variables can also be applied to universally quantified predicate variables, and vice versa.

To my knowledge, Wilhelm Ackermann was the first one who considered this problem [1, 2, 3]. He gave two procedures for eliminating existential quantifiers over one–place predicate variables. Both eliminate only one quantifier at a time. The first one requires to bring the formula into a form

$$\exists P\ \forall x\ (A(x) \vee P(x)) \wedge \Delta[\overline{P}]$$

where $\Delta[\overline{P}]$ is a formula containing only negative occurrences of $P(x)$. The result is then $\Delta[A]$, i.e. all occurrences of $\overline{P}(x)$ are replaced with $A(x)$ in Δ. We have problems if we try to apply this method to clauses with several occurrences of P. These problems are overcome in Ackermann's second method.

For Ackermann's second method it is necessary to bring the formulae into a kind of clause form. In his notation he writes clauses as disjunctions in the form $\mathcal{A}^{x_1,\ldots,x_n}_{y_1,\ldots,y_m}$. $\mathcal{A}$ is a formula free of P–literals, the subscripts y_i are short for $P(y_i)$ and the superscripts x_i are short for $\neg P(x_i)$. A *contraction* operation

$$\mathcal{A}^{x_1,\ldots,x_n,z}_{y_1,\ldots,y_m} \wedge \mathcal{B}^{p_1,\ldots,p_n}_{q_1,\ldots,q_l,z} \rightarrow \mathcal{A}^{x_1,\ldots,x_n}_{y_1,\ldots,y_m} \vee \mathcal{B}^{p_1,\ldots,p_n}_{q_1,\ldots,q_l}$$

generates a new clause from the two old ones. Thus, contraction on z actually means resolution between $P(z)$ and $\neg P(z)$. (The step to full resolution with unification is small.) Ackermann showed that generating exhaustively all contractions and taking the conjunction of the P–free clauses yields a formula which is equivalent to the original second–order formula. His formulation of the second method is still quite restrictive and not very practical. n–place predicates have to be turned into one–place predicates and all arguments have to be abstracted to variables. Furthermore there is no redundancy checking like subsumption or tautology elimination. But the basic idea is strong enough to be turned into a powerful method.

This basic idea was still further abstracted by Kreisel and Krivine [16] who proved a theorem showing that a set of formulae with a predicate variable P which contains all consequences of the formulae with P is equivalent to its P–free part. That means, as soon as you have all consequences with P, you don't need the P anymore. Since in general these are infinitely many consequences, for practical purposes this theorem has to be further refined.

Different variations and improvements of these basic ideas have been published in recent years [27, 30, 28, 11]. The only algorithm which, to my knowledge, has been implemented is the SCAN Algorithm.

4.1 The SCAN Algorithm

In [13] 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.

The definition of the algorithm is:

Definition 4.1 (The SCAN Algorithm)
The 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.
The output of SCAN — if it terminates — is a formula φ_α which is *logically equivalent* to α, but does not contain 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 $P_1, \ldots, P_n$ are generated. C–resolution ('C' for constraint) is defined as follows:

$$\begin{array}{ll} P(s_1, \ldots, s_n) \vee C & P(\ldots) \text{ and } \neg P(\ldots) \\ \neg P(t_1, \ldots, t_n) \vee D & \text{are the } \textit{resolution literals} \\ \hline C \vee D \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n & \end{array}$$

   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 steps 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 (we say the literal is 'resolved away'), the clause containing this literal must be deleted (purity deletion). If all clauses are deleted this way, 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.

There are two stages where SCAN can fail at computing a first–order equivalent for α: (i) the resolution does not terminate and (ii) reversing Skolemization is not possible. If (ii) occurs the output is a second–order solution involving existentially quantified Skolem functions.

The SCAN Implementation

Our SCAN implementation is a modified version of the Otter theorem prover developed by Bill McCune at Argonne National Laboratory. The main modifications in Otter itself are the integration of the constrained resolution rule, the purity deletion operation, some further simplifications and the particular resolution strategy SCAN requires. The unskolemization routine was implemented by Thorsten Engel as a separate module.

SCAN/Otter simply reads an input file containing the full specification of the quantifier elimination problem and it generates an output file with the protocol of the run and the final result. Besides this simple interface there are two further levels of interfaces. The next level of interfaces is a Prolog interface for applying SCAN to different problems which can be reduced to quantifier elimination problems. Currently two such applications are implemented. The first one uses SCAN to compute the corresponding frame properties for Hilbert axioms as explained above.

The second interface realizes circumscription. Circumscription was proposed by John McCarthy as a logically simple and clear means to doing default reasoning. In his most general form the circumscription of a given formula φ with some predicates P and some other predicates Z is the second–order formula

$$circ(\varphi[P,Z],P,Z) = \varphi[P,Z] \wedge \forall P^*, Z^* \; (\varphi[P^*,Z^*] \wedge (P^* \Rightarrow P)) \Rightarrow (P \Rightarrow P^*)$$

This formula minimizes the extension of the predicates P, possibly at the cost of the predicates Z whose extension are allowed to vary freely. The SCAN circumscription interface computes this formula and then invokes the basic SCAN algorithm to get rid of the second–order quantifiers. All this is accessible by WWW in

http://www.mpi-sb.mpg.de/pub/guide/staff/ohlbach/scan/scan.html.

We have prepared a html form that can be completed by the user to furnish the input to our SCAN implementation. The program will be run on our machines at the MPI and the output will automatically forwarded back to the user.

4.2 *Quantifier Elimination with Fixpoint Formulae*

The fact that the resolution operation in the SCAN algorithm may loop inspired Nonnengart and Szałas [24] to develop a new approach that attempts to keep better control of the loops. Their approach is an improved version of Ackermann's first method. It uses formulae with a least fixpoint operator $\mu P.\Phi[P]$ for representing possibly infinite conjunctions and disjunctions:

$$\mu P.\Phi[P] = \bigvee_{\beta \in \alpha} \Phi^{\beta}[\bot] \tag{4.1}$$

α is an ordinal number (the least such ordinal is called the *closure ordinal for* $\Phi(P)$). $\bot$ is falsity.

Theorem 4.2 (Nonnengart & Szałas)
Assume that all occurrences of the predicate symbol P in the formula Ψ bind only variables.

- If Φ and Ψ have only negative occurrences of P then the closure ordinal for $\Phi(\overline{P})$[5] is less than or equal to ω, and

$$\exists P \forall \vec{y}\,(P(\vec{y}) \vee \Phi[\overline{P}]) \wedge \Psi[\overline{P}] \;= \Psi[\nu \overline{P(\vec{y})}.\Phi[\overline{P}]] \tag{4.2}$$

- If Φ and Ψ have only positive occurrences P then the closure ordinal for $\Phi(P)$ is less than or equal to ω, and

$$\exists P \forall \vec{y}(\neg P(\vec{y}) \vee \Phi[P] \wedge \Psi[P]) \;=\; \Psi[\nu P(\vec{y}).\Phi[P]] \tag{4.3}$$

where the above substitutions replace the variables bound by fixpoint operators by the corresponding actual variables of the substituted predicate[6]. $\lhd$

Here $\nu P.\Phi(P)$ is an abbreviation for $\neg\mu\overline{P}.\neg\Phi(\overline{P})$. For eliminating P first we have to transform the formula into a form as required by the theorem above, do the substitution with the fixpoint formula, and then expand the fixpoint formula using its representation (4.1). After each expansion step one has to check whether the new part of the formula follows form the old parts or not. This way redundant loops can be detected and non-redundant loops can be approximated. In [24] many examples of this transformation are presented.

5 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. Usually 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 [9] we have shown the details of this procedure

[5]$\overline{P}$ denotes a *negated* occurrence of P in a formula.

[6]Observe that the assumption that P's in Ψ bind only variables is necessary here.

and tested it with a lot of examples from modal and relevance logic. In one of the examples we tried to find the corresponding Hilbert axiom for *transitive* accessibility relations 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 [21, 20]. `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.

6 Simplifying Semantics and Expressiveness of the Logic

The semantics of a logic is usually formulated in set–theoretic notation. This is one of the most expressive mathematical languages we have. Therefore it is very easy to formulate in this language conditions on the semantic structure of our logic which have no counterpart on the syntactic (axiomatic) 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 properties 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 properties as initially intended. It is of course very important to know the expressiveness of the logic, because otherwise the effects of requirements to the semantics are unpredictable.

This problem has also been investigated in correspondence theory for modal logic and a number of results and techniques have been developed [6]. An alternative 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, we take a translation τ: $\mathcal{L} \to \mathcal{L}_t$ which exhibits certain syntactic invariants on the translated formulae, and use these invariants to investigate the effect of a 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 relevant 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 [23] from 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 for formulae in negation normal form are

$$\begin{array}{rcl}\pi(\Diamond\varphi, w) & = & \exists\gamma{:}AF\ \pi(\varphi, w{:}\gamma)\\ \pi(\Box\,\varphi, w) & = & \forall v\ R(w,v) \Rightarrow \pi(\varphi, v)\end{array}$$

For example a formula $\Box\,\Diamond c$ is translated into $\forall v\ R(w,v) \Rightarrow \exists\gamma{:}AF_q\ c(v{:}\gamma)$. Intuitively one can understand the sort AF as a set of functions mapping worlds to R–accessible worlds. The colon : is an infix function symbol which can be understood as application function, $a{:}\gamma \overset{\text{def}}{=} \gamma(a)$. It is not necessary to understand the details of this translation technique here. The important 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 $\forall x\ \forall\gamma{:}AF\ R(x, x{:}\gamma)$. It relates the sort AF with the predicate R. If we do theorem proving by refutation with resolution, we immediately see that there is no resolution partner at all for the irreflexivity clause $\neg R(x,x)$. This clause is redundant and cannot contribute to a proof. Thus, requiring irreflexivity or not has no effect on the provability of modal theorems; it can not be characterized 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 properties. Here we can exploit the results about resolution strategies with deletion operations [5], and the technique is really easy to apply.

7 Summary

The development and investigation of application oriented logics comprises many aspects and problems. For a few of them some computer support is possible which frees the investigator from sometimes quite complex computations. We have considered the problems of

- reasoning in Hilbert systems,
- finding corresponding frame properties for given Hilbert axioms, and vice versa
- finding corresponding axioms for semantic properties, and
- investigating the expressiveness of logics,

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 can find solutions by semi–automatic procedures.

As an example for a complex task which definitely requires computer support, we developed a translation of the logic of graded modalities into predicate logic [26]. The logic of graded modalities has operators M_n which are interpreted as 'in more than n accessible worlds' [33]. In contrast to the usual tableau systems for this logic, which work only for small numbers, the translation allows for an inference system using some kind of symbolic arithmetic instead of counting Skolem constants. We first translated the logic of graded modalities into a normal multi–modal logic and then further into predicate logic. The axiomatization of the multi–modal logic turned out to be quite complex and only with the SCAN algorithm we were able to find its semantics. This semantics was then used as a basis for an optimized translation into predicate logic.

Acknowledgments

I would like to thank Dov Gabbay as well as my colleagues Andreas Nonnengart and Renate Schmidt for very fruitful discussions. Thorsten Engel implemented the SCAN algorithm by modifying the Otter theorem prover and integrating an unskolemization algorithm.

References

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

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

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

[4] 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.

[5] 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.

[6] Patrick Blackburn, Maarten de Rijke, and Yd Venema. The algebra of modal logic. Report CS-R9463, Centrum voor Wiskunde en Informatica / Computer Science, Department of Software Technology, 1994.

[7] W.J. Blok and Don Pigozzi. *Algebraizable Logics*, volume 77, 396 of *Memoirs of the American Mathematical Society.* American Mathematical Society, Procidence, Rhodes Island, USA, 1989.

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

[9] Chris Brink, Dov Gabbay, and Hans Jürgen Ohlbach. Towards Automating Duality. *Journal of Computers and Mathematics with Applications*, 29(2):73–90, 1994. A longer version appeared as a Technical Report MPI-I-93-220 of the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

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

[11] Patrick Doherty, Witold Lukaszewics, and Andrzej Szalas. Computing circumscription revisited: A reduction algorithm. Technical Report LiTH-IDA-R-94-42, Institutionen för Datavetenskap, University of Linköping, 1994.

[12] 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, Max-Planck-Institut für Informatik, Saarbrücken, and in the *South African Computer Journal*, 1992.

[13] 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 [12].

[14] B. Jónsson. A survey of Boolean algebras with operators. In Rosenberg and Sabidussi, editors, *Algebra and Orders*, pages 239–286. 199?

[15] B. Jónsson and A. Tarski. Boolean algebras with operators, Part I. *American Journal of Mathematics*, 73:891–939, 1951.

[16] G. Kreisel and J.L. Krivine. *Éléments de Logique Matheématique. Théorie des modèles.* Société Mathématique de France, 1966.

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

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

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

[20] 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.

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

[22] 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.

[23] 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.

[24] Andreas Nonnengart and Andrzej Szalas. A fixpoint approach to second-order quantifier elimination with applications to correspondence theory. Technical report, Saarbrücken, 1995.

[25] Hans Jürgen Ohlbach, Dov Gabbay, and David Plaisted. Killer Transformations. Technical Report MPI-I-94-226, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1994.

[26] Hans Jürgen Ohlbach, Renate A. Schmidt, and Ullrich Hustadt. Translating Graded Modalities into predicate logic. To appear in *H. Wansing (ed.): Proof Theory in Modal Logics*, Oxford University Press, 1995.

[27] 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.

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

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

[30] 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.

[31] Johan van Benthem. *Modal Logic and Classical Logic.* Bibliopolis, Naples, 1983.

[32] 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 Vol. 165*, pages 167–248. D. Reidel Publishing Company, Dordrecht, 1984.

[33] Wiebe van der Hoek. *Modalities for Reasoning about Knowledge and Quantities.* PhD thesis, Vrije Universiteit Utrecht, 1992.

Received 31 August 1994. Revised 24 October 1995