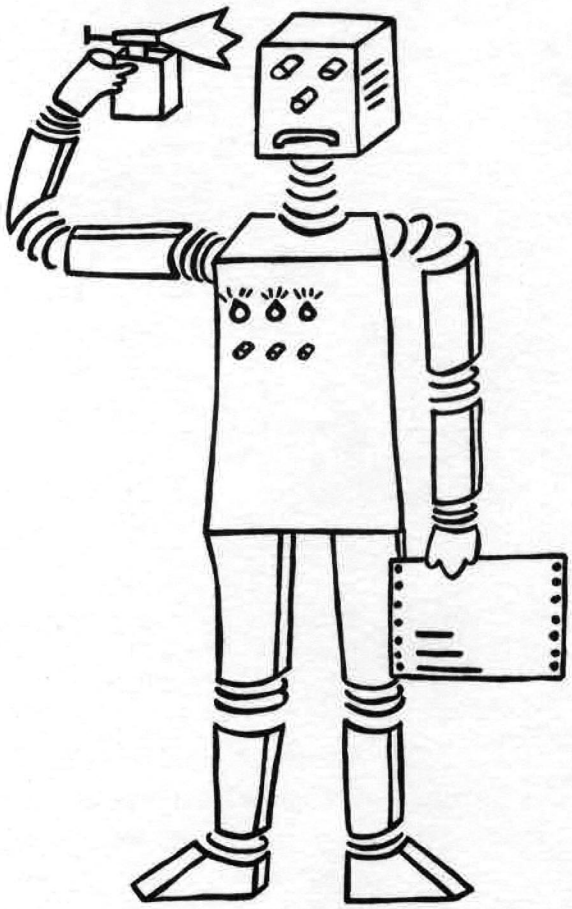
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

SEKI-REPORT

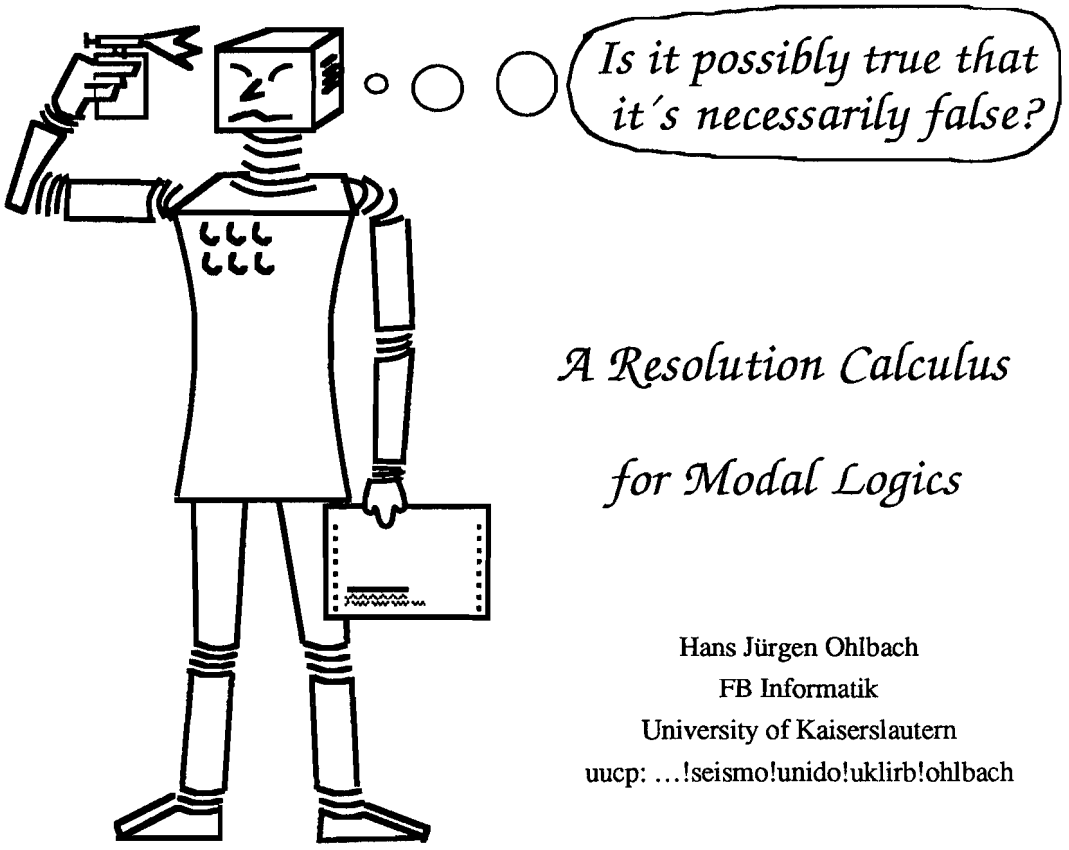
Artificial
Intelligence
Laboratories



A Resolution Calculus for Modal Logics

Hans-Jürgen Ohlbach

SEKI Report SR-88-08



A Resolution Calculus for Modal Logics

Hans Jürgen Ohlbach
FB Informatik
University of Kaiserslautern
uucp: ...!seismo!unido!uklirb!ohlbach

Abstract A resolution calculus for the quantified versions of the modal logics K, T, K4, KB, S4, S5, B, D, D4 and DB is presented. It presupposes a syntax transformation, similar to the skolemization in predicate logic, that eliminates the modal operators from modal logic formulae and shifts the modal context information to the term level. The formulae in the transformed syntax can be translated into conjunctive normal form such that a clause based modal resolution calculus is definable without any additional inference rule, but with special modal unification algorithms. The method can be applied to first-order modal logics with the two operators $\Box$ and $\Diamond$ and with standard constant-domain possible worlds semantics with flexible constant and function symbols, where the accessibility relation may have any combination of the following properties: reflexivity, symmetry, transitivity, seriality or non-seriality. While extensions to other systems seem possible, they have not yet been investigated.

Table of Contents

Chapter One

Introduction

The invention of the resolution principle for predicate logic by John Alan Robinson [Robinson 65] was an important step in the long lasting attempt of rendering formal reasoning to automated treatment. The resolution rule is simple, clear and easy to implement. Its efficiency - in terms of the branching rate in the search space, which is always finite for resolution - surpasses considerably other calculi like natural deduction or tableau systems with an unrestricted instantiation rule. Therefore the resolution rule has become the favorite inference rule, for automated theorem proving as well as for PROLOG style logic programming. In more than 25 years of work with resolution based theorem proving sophisticated refinements, control and implementation techniques have been developed that enable today's theorem provers to solve undoubtedly nontrivial problems [Wos&Winker 84].

Unfortunately the application of the clause based resolution principle has been restricted to standard predicate logic. For nonclassical logics - modal logics, temporal logics, epistemic logics, relevance logics etc. - completely different calculi have been developed that require different implementations for their respective theorem proving systems with hardly a chance to apply results and techniques of the traditional work.

With this monograph we intend to make a first step towards a situation where nonstandard logics, at least a large class of modal logics, are amenable to standard techniques. The method to bring this change about is to "skolemize" the modal operators. For example the formula $\Box\mathcal{F}$ is translated into a formula $\forall w\, \mathcal{F}[w]$ where w is quantified over all accessible worlds and [w] is attached as an additional argument to the terms and literals in $\mathcal{F}$. Thus, we eliminate the modal operators and obtain a standard predicate logic like syntax that still represents the modal semantics. The modal context information is recorded as a "world-path", a new kind of terms for which a special unification algorithm is to be defined. For each particular modal logic like S4, S5 etc., we need a particular unification algorithm; and this is the only change that is necessary to turn any resolution theorem prover for predicate logic, no matter if clause based or not, and even logic programming systems, into a theorem prover for modal logic.

An Example that Demonstrates the Basic Ideas

Consider the formula

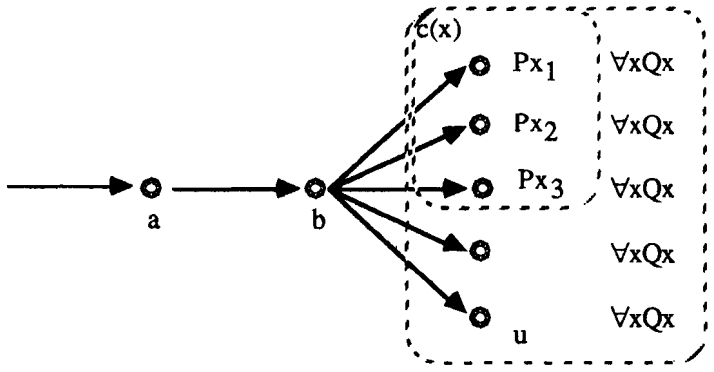
$$\Diamond\,\Diamond\forall x(\Diamond Px \wedge \Box\, Qx) \Rightarrow \Diamond\,(\forall y Py \wedge \forall z Qz) \qquad (*)$$

The meaning of such a formula can be described in terms of possible worlds which are connected by an accessibility relation [Hughes&Cresswell 68]. A possible world is an interpretation in the predicate logic sense. It determines how the function and predicate symbols are to be interpreted in that world. For instance the symbol P may be interpreted in world 'a' as the predicate 'even', and in world 'b' as the predicate 'odd'. The nesting of the modal operators in a formula determines which world or which interpretation respectively is actually meant. $\Diamond\mathcal{F}$ means "there is a world b which is accessible from the actual world a, such that $\mathcal{F}$ holds in b". $\Box\mathcal{F}$ means "for every world b which is accessible from the actual world a, $\mathcal{F}$ holds in b".

The premises $\Diamond\,\Diamond\forall x\,(\Diamond Px \wedge \Box Qx)$ of the formula (*) can therefore be expressed in words as:

$\Diamond$	From the actual world there is an accessible world a,
$\Diamond$	from a there is an accessible world b,
$\forall x$	such that for all x
($\Diamond$	there is a world c, accessible from b (but depending on x, therefore c(x))
Px	such that Px holds, where P is interpreted in world c(x)
$\wedge$	and
$\Box$	for all worlds u which are accessible from b
Qx)	Qx holds, where Q is interpreted in world u.

Graphically:



The syntax transformation we are going to present in this paper records these worlds a, b, c(x) and u explicitly and attaches them as an additional "world-path" argument to the predicate and function symbols.

The above formula $\Diamond\,\Diamond\forall x\,(\Diamond Px \wedge \Box Qx)$, for instance, is translated into $\forall x(P[abc(x)]x \wedge \forall u\, Q[abu]x)$ with the intuitive meaning:

$\forall x$	| For all x
(P([abc(x)] x)	| Px holds in all worlds which are accessible via the paths a b c(x)
$\wedge$	| and
$\forall u$	| for all admissible worlds u (which worlds are actually admissible depends on the world-paths in the subformulae of the quantifier.)
Q([abu] x))	| Qx holds in all worlds which are accessible via the paths a b u.

In order to prove the formula (❄) by contradiction the consequence of the implication must be negated, yielding $\neg \Diamond (\forall y Py \wedge \forall z Qz)$, and after moving the negation sign inside: $\Box(\exists y \neg Py \vee \exists z \neg Qz)$.
The transformed version is $\forall v(\neg P([v]\ f[v]) \vee \neg Q([v]\ g[v]))$ with the intuitive meaning:

$\forall v$	For all admissible worlds v
$(\neg P([v]\ f[v])$	$\neg P(f[v])$ holds in all accessible worlds.
	f is a skolem function that denotes the original y "that must exist",
	but in each world v, there may exist another y.
$\vee$	or
$\neg Q([v]\ g[v]))$	$\neg Q(g[v])$ holds in all accessible worlds.
	g is the second skolem function that depends also on v.

(The first world-path [v] in P([v] f[v]) determines the modal context for the predicate symbol P whereas the second [v] determines the modal context for f. It is a coincidence that both are the same.)

Eliminating the universal quantifiers, the transformed negated formula (❄) can be written in clause form:

C1:	P([abc(x)] x)
C2:	Q([abu] x)
C3:	$\neg P([v]\ f[v]) \vee \neg Q([v]\ g[v])$

Let us try to find a resolution proof.
Case 1: Assume the accessibility relation is serial, i.e. from each world there is an accessible world.
There are two candidates for resolution operations, C2 with C3,2 and C1 with C3,1. Consider the first candidate, C2 with C3,2. Before generating a resolvent the atoms Q([abu] x) and Q([v] g[v]) must be unified, i.e. the problem of unifying the two world-paths [abu] and [v] has to be solved. It is easy to see that the unification is impossible unless the accessibility relation of the underlying logic is transitive or symmetric. Let us assume transitivity. In this case $\{v \mapsto [abu]\}$ is a (most general) unifier for [abu] and [v]. Combining this substitution with the unifier for x and g[v], the final unifier $\{v \mapsto [abu], x \mapsto g[abu]\}$ is obtained.
The modal resolution step is now:

C2:	Q([abu] x)	
C3:	$\neg P([v]\ f[v]) \vee \neg Q([v]\ g[v])$	unifier: $\sigma = \{v \mapsto [abu]\ ,\ x \mapsto g[abu]\}$
Resolvent:	$\neg P([abu]\ f[abu])$	(in modal syntax: $\Diamond\Diamond\Box\exists y\ \neg Py$)

Consider now the second resolution possibility which gives rise to the unification problem P([abc(x)] x) and P([v] f[v]). This time the world-paths [abc(x)] and [v] can be unified only in logics with a transitive accessibility relation. In this case the unifier is $\{v \mapsto [abc(x)]\}$. This substitution can be applied to the remaining terms before the unification proceeds, and the second unification problem, x with f[abc(x)], fails with an occur check failure. The atoms P([abc(x)] x) and P([v] f[v]) are not unifiable. Since the first resolvent is also not unifiable with any other atom, and there is no other possibility for resolution, the proof fails; and in fact, the formula (❄) is *not* a theorem.

Case 2: Assume the accessibility relation is *not* serial.

In the non-serial case there may be worlds from which there are no accessible worlds at all. A formula $\Box\mathcal{F}$ is true in such worlds, not because the formula $\mathcal{F}$ evaluates to a truth value, but because the quantification "for all accessible worlds ..." in the semantics of the $\Box$-operator is empty. This has two consequences for the resolution rule in the transformed syntax: The first consequence is that a world-variable v in a formula $\forall v \neg P([v]\ f[v]) \vee \neg Q([v]\ g[v])$ cannot be instantiated safely with a non-variable term. The interpretations with no accessible worlds would satisfy the original formula, but not its instance. The second consequence is that two syntactically complementary literals like $\forall v\ R[v]$ and $\forall v \neg R[v]$ are not necessarily semantically contradictory. Both formulae are simultaneously satisfiable when the quantification $\forall v$ is empty. They can't therefore be used as resolution literals without further provision.

To overcome these difficulties we must introduce explicit reasoning about inhabited and not inhabited worlds. To this end a special predicate 'End(p)' is introduced which is true in an interpretation when 'p' is evaluated to the "last" world, i.e. a world without accessible worlds. The right instance of $\forall v \neg P([v]\ f[v]) \vee \neg Q([v]\ g[v])$ with the substitution $\sigma = \{v \mapsto [abu], x \mapsto g[abu]\}$ can now be expressed:

$\neg End([]) \wedge \neg End([a]) \wedge \neg End([ab]) \Rightarrow \forall u \neg P([abu]\ f[abu]) \vee \neg Q([abu]g[abu])$ or in clause form

$End([]) \vee End([a]) \vee End([ab]) \vee \neg P([abu]\ f[abu]) \vee \neg Q([abu]g[abu])$

with the meaning "when neither the initial world, nor the world [a] nor the world [ab] is the last world then $\forall u \neg P([abu]\ f[abu]) \vee \neg Q([abu]\ g[abu])$ is a correct instance. (Actually the last condition $\neg End([ab])$ can be omitted because when [ab] is the last world then the quantification $\forall u$ is empty and the clause is true anyway.)

The "conditioned instances" of the two resolution literals in our example

C2: $Q([abu]\ x)$

C3: $\neg P([v]\ f[v]) \vee \neg Q([v]\ g[v])$ unifier: $\sigma = \{v \mapsto [abu]\ ,\ x \mapsto g[abu]\}$

are

$\sigma\downarrow$C2: $End([]) \vee End([a]) \vee Q([abu]\ g[abu])$

$\sigma\downarrow$C3: $End([]) \vee End([a]) \vee \neg P([abu]\ f[abu]) \vee \neg Q([abu]\ g[abu])$

To complete the resolution operation for this example we create the resolvent in the usual way, but insert an additional literal End([ab]) into the resolvent to ensure that the variable u denotes a nonempty set and the literals Q([abu] g[abu]) and $\neg$Q([abu]g[abu]) are really contradictory. The resolvent is therefore

$End([]) \vee End([a]) \vee End([ab]) \vee \neg P([abu]\ f[abu])$

The presence of the literal $\neg P([abu]\ f[abu])$ in this particular example makes the literal End([ab]) superfluous such that the final resolvent is

$End([]) \vee End([a]) \vee \neg P([abu]\ f[abu])$.

A literal End(p) can be resolved with a literal containing a world-path [p c:...] where c is a nonvariable term denoting a world which is accessible from the world denoted by p. In the above example, both literals End([]) and End([a]) can be resolved against C2 because the world-path [abu] in C2 denotes a world [a], accessible from the initial world - which contradicts End([]) - and a world [ab] - which contradicts End([a]). Two more resolution steps yield therefore the same clause $\neg P([abu]\ f[abu])$ as in the serial case.

Thus, the resolution rule in the non-serial case is a partial theory resolution in the sense of Mark Stickel [Stickel 85] where the End-literals form the *residue* which is implied by the conjunction of the two resolution literals.

A Short Summary of the Subsequent Chapters

Chapter 2: M-Logic

We define the modal logics that are considered in this thesis. The formal definition of syntax and semantics serves as a point of reference for the following soundness and completeness proofs.

Chapter 3: P-Logic

The modal operators are translated into "world-paths" as mentioned above. We call the resulting logic "P-logic" and give a precise definition of the syntax and semantics of this logic.

Chapter 4: Translation from M-Logic to P-Logic

An algorithm for translating modal logic formulae into P-logic syntax is presented in this chapter. It is shown that a modal logic formula is satisfiable if and only if the corresponding translated formula is satisfiable in P-logic. Beyond this point, there is no need to ever consider the original modal logic syntax and instead we concentrate on the development of algorithms for the P-logic syntax.

Chapter 5: Tools for P-Logic

This chapter contains technical preparations for the subsequent chapters, the generation of a conjunctive normal form for P-formulae, some invariants on the structure of terms and the definition of substitutions.

Chapter 6: Modal Unification

The unification algorithms for different variants of the accessibility relation are defined in this chapter. We prove the termination, soundness and completeness as well as some helpful invariants on the structure of the unified terms. It turns out that some of the unification problems are special cases of well known theory unification problems (c.f. [Siekmann 88]).

Chapter 7: Modal Resolution

The two versions of the modal resolution rule are defined. Resolution for logics with a serial accessibility relation is just like ordinary resolution with the only difference that the unification algorithm may produce more than one, but at most finitely many most general unifiers. When the accessibility relation is not serial, additional literals must be inserted into the resolvent which express the condition "if the worlds in question are inhabited then ...". We show the soundness of these two resolution rules. For the completeness proofs, an additional device is needed which will be provided in the next two chapters.

Chapter 8: Term Frames

Completeness of the resolution rule says that whenever a clause set is unsatisfiable, i.e. false in *all* interpretations, the empty clause can be derived by a finite sequence of resolution steps. Standard completeness proofs take advantage of the fact that not all interpretations, but only interpretations over the set of ground terms (in predicate logic usually called Herbrand Interpretations, although originally introduced by Löwenheim and Skolem) need to be considered. We therefore define term interpretations or term frames respectively for P-logic and show that every satisfiable clause set has a term model.

Chapter 9: Semantic Trees

Semantic trees are a standard datastructure for giving an exhaustive survey of all possible term interpretations. In this chapter it is shown for P-logic that an unsatisfiable clause set has a finite closed semantic tree, which is then used in the completeness proof.

Chapter 10: Completeness of Modal Resolution

Collecting the results of the two previous chapters, we can now prove that for every unsatisfiable clause set, the empty clause can be derived by a finite sequence of resolution steps. Hence, we have a semidecision procedure for first-order modal logic formulae based on the resolution rule.

Chapter 11: Conclusion

The final conclusion of this work is that modal logic theorems can be proved using an ordinary clause based resolution theorem prover. The efficiency, i.e. the branching rate in the search space for modal logics with serial accessibility relations is not worse than the efficiency of predicate logic resolution theorem proving with, say, an associative and commutative function symbol, where we have more than one most general unifier. Although the technical aspects of the resolution rule for logics with non-serial accessibility relations can be compared with Digricoli's RUE-resolution for equality handling [Digricoli 79], the final search space is much smaller. Therefore logics with non-serial accessibility relations are also tractable.

We compare the new method with other deduction methods for modal logic and conclude with a discussion of possible extensions of the ideas presented in this work to more complex modal and epistemic logics.

Although most of our logical notions are formally defined within this thesis, we assume some familiarity with standard predicate and modal logic as well as some knowledge of automated theorem proving. Standard references are [Chang&Lee 73], [Loveland 78], [Hughes&Cresswell 68], [Smullyan 68].

Chapter Two

M-Logic

Since there is a large variety of modal logics, in syntax as well as in semantics, it is necessary to firmly establish the particular kind of logic we are interested in. M-logic (M for Modal) is the name for a syntactically restricted version of the "classical" modal logics with the two modal operators $\Box$ (necessarily) and $\Diamond$ (possibly). The restrictions are not principal in nature, but concern the elimination of some "syntactic sugar" in order to keep the still extensive formalism in manageable proportions. Although our modal resolution calculus is not based on the original modal logic syntax, the formal definition of the syntax and semantics of our M-logic serves as a point of reference for the translation from M-logic into a more appropriate syntax, and for proving the soundness and completeness of the translation.

2.1 Syntax of M-Logic

The formulae of most classical modal logics are usually built - unlike more recent extensions in temporal or dynamic logics - just as predicate logic formulae with two additional modal operators. We shall use the following logical connectives, quantifiers and operators:

$\wedge$	(and)	$\forall$	(for all)
$\vee$	(or)	$\exists$	(there exists)
$\neg$	(not)	$\Box$	(necessarily)
$\Rightarrow$	(implies)	$\Diamond$	(possibly)
$\Leftrightarrow$	(is equivalent)		

We consider modal logic formulae in *negation normal form* without the implication and equivalence sign, where all negation signs are moved in front of the atoms. Arbitrary modal logic formulae can be brought into this normal form using the following validity preserving rewrite rules:

$\mathcal{F} \Leftrightarrow \mathcal{G}$	$\rightarrow$	$\mathcal{F} \Rightarrow \mathcal{G} \wedge \mathcal{G} \Rightarrow \mathcal{F}$	$\neg(\mathcal{F} \wedge \mathcal{G})$	$\rightarrow$	$\neg\mathcal{F} \vee \neg\mathcal{G}$
$\mathcal{F} \Rightarrow \mathcal{G}$	$\rightarrow$	$\neg\mathcal{F} \vee \mathcal{G}$	$\neg(\mathcal{F} \vee \mathcal{G})$	$\rightarrow$	$\neg\mathcal{F} \wedge \neg\mathcal{G}$
$\neg\forall x\, \mathcal{F}$	$\rightarrow$	$\exists x\, \neg\mathcal{F}$	$\neg\Box\mathcal{F}$	$\rightarrow$	$\Diamond\neg\mathcal{F}$
$\neg\exists x\, \mathcal{F}$	$\rightarrow$	$\forall x\, \neg\mathcal{F}$	$\neg\Diamond\mathcal{F}$	$\rightarrow$	$\Box\neg\mathcal{F}$.

In the sequel "M" will be used as the index and prefix for Modal logic, because in subsequent chapters we must distinguish between the original and the translated formulae, which then will be indexed and prefixed with "P" for Predicate logic style. Additional notions like D-variables and D-valued function symbols (D for Domain) are also defined this way, to distinguish them from other variable and function types to be added later on.

Definition 2.1.1 (The Signature of M-Logic)

Besides of the fixed set of logical connectives and operators $\{\wedge, \vee, \neg, \Diamond, \Box\}$, the alphabet for building M-logic terms and formulae consists of the following disjoint sets of symbols:

$\mathbb{V}_D$	is a set of *D-variable* symbols.
$\mathbb{F}_{D,n}$	is a set of n-place *D-valued function* symbols where constants are in $\mathbb{F}_{D,0}$.
$\mathbb{F}_D$	is the union of all D-valued function symbols.
$\mathbb{P}_n$	is a set of n-place *predicate* symbols.
$\mathbb{P}$	is the union of all predicate symbols.

$\Sigma_M := (\mathbb{V}_D, \mathbb{F}_D, \mathbb{P})$ is an *M-signature* . ■

The prefix "D" in D-variable symbols has been introduced to emphasize that these variables are interpreted in the Domain of discourse. In P-logic we shall introduce W-variable symbols which are interpreted in the set of Worlds. Hence, the prefix "D" in D-terms below emphasizes that the terms denote domain elements, in contrast to W-terms in P-logic which concern worlds.

It is noted that an M-signature does not only supply symbols for building formulae, but we shall also speak of the M-signature of a set of formulae, i.e. the particular set of symbols which occur just in these formulae.

As a point of reference we state the standard definitions for terms, atoms, literals and formulae.

Definition 2.1.2 (Terms, Atoms, Literals and Formulae)

Given an M-signature $\Sigma_M := (\mathbb{V}_D, \mathbb{F}_D, \mathbb{P})$,

- the set of *D-terms* $\mathbb{T}_D$ over Σ_M is defined as the least set such that:
  - (i) Each D-variable symbol is a D-term.
  - (ii) If $f \in \mathbb{F}_{D,n}$ and $t_1,\ldots,t_n$ are D-terms then $f(t_1,\ldots,t_n)$ is a D-term.
- If $P \in \mathbb{P}_n$ and $t_1,\ldots,t_n$ are D-terms then $P(t_1,\ldots,t_n)$ is an *M-atom*.
- An *M-literal* is either an M-atom (positive literal) or a negated M-atom (negative literal). Let $\pm P(t_1,\ldots,t_n)$ denote a literal which is either positive or negative.
- The set of *M-formulae* over Σ_M is defined as the least set such that:
  - (i) An M-literal is an M-formula.
  - (ii) If $\mathcal{F}$ and $\mathcal{G}$ are M-formulae and x is a D-variable symbol then
    $\mathcal{F} \wedge \mathcal{G}$, $\mathcal{F} \vee \mathcal{G}$, $\forall x \mathcal{F}$, $\exists x \mathcal{F}$, $\Box \mathcal{F}$ and $\Diamond \mathcal{F}$ are M-formulae.

  For convenience we assume that the quantified variables in a formula are standardized apart (renamed), i.e. formulae like $\forall x \exists x \mathcal{F}$ or $\forall x \mathcal{G} \vee \exists x \mathcal{F}$ do not occur. ■

2.2 Semantics of M-Logic

A common model theory for modal logics is Kripke's "possible worlds semantics" [Kripke 59, 63]. A possible world determines how the function and predicate symbols are to be interpreted in that world. Within each world the interpretation is defined in the classical predicate logic sense. Different possible worlds may assign different meanings to the same symbol. For instance, the symbol f may be interpreted in one world as addition on integers and in another world as multiplication. Some authors distinguish between "rigid" function and predicate symbols, whose interpretation is the same in every world, and "flexible" function symbols whose interpretation may differ from world to world. We do not make this distinction, because it is a trivial exercise to extend a calculus with flexible symbols to one with rigid symbols as well. Just ignore the modal context information for rigid symbols.

When the domain of the interpretation is the same in every possible world, we speak of a *constant-domain* interpretation, otherwise we speak of a *varying-domain* interpretation. Constant-domain interpretations are characterized by the *Barcan formula*: $\forall x \Box \mathcal{F} \Rightarrow \Box \forall x \mathcal{F}$ which expresses the fact that the universal quantifier does not depend on the modal context. Our modal resolution calculus will be defined for constant-domain interpretations only; a slight modification for varying-domain interpretations, however, will be presented in chapter 11.

Worlds are connected by an "accessibility relation" $\Re$. A possible worlds structure can be displayed as a transition graph, where the nodes are labeled with the worlds and the edges represent the accessibility relation $\Re$. Two main classes of accessibility relations can be distinguished: serial and non-serial ones. An accessibility relation is said to be serial if from every world there exists an accessible world; the relation is non-serial otherwise. Accessibility relations with the following properties are considered in this work. (The letters in parentheses denote the traditional name of the corresponding logic.)

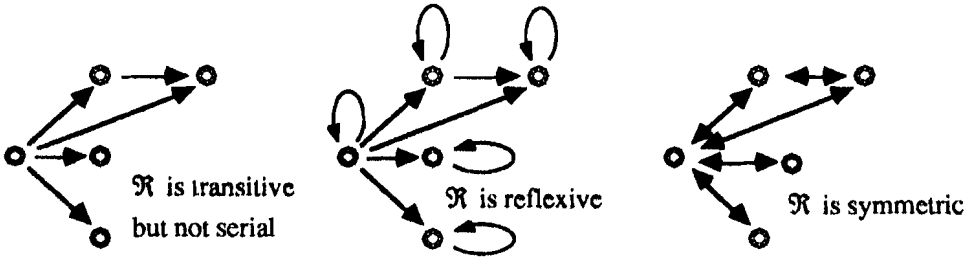
non-serial	serial		
no special properties (K)	seriality only	(D)	
	reflexivity	(T or M)	(reflexivity implies seriality)
	symmetry	(DB)	(symmetry implies seriality)
transitivity (K4)	transitivity	(D4)	
	reflexivity and symmetry	(B)	
	reflexivity and transitivity	(S4)	
	reflexivity, symmetry and transitivity	(S5).	

The two missing combinations symmetry and non-seriality (KB) as well as symmetry and transitivity can be reduced to simpler ones as follows:
If the accessibility relation $\Re$ is symmetric, either $\Re$ is not serial and there are only isolated worlds (let us call this the *predicate logic interpretations*) or $\Re$ is serial because whenever a world b can be accessed from a world a, there is a world accessible from b, namely a. In order to check a formula for unsatisfiability in symmetric non-serial interpretations, it is therefore possible to split the proof into two cases:

1. Check the predicate logic interpretations. The unsatisfiability check for predicate logic interpretations can be performed by evaluating all subformulae $\Diamond\mathcal{F}$ to False and all subformulae $\Box\mathcal{F}$ to True and then proving the remaining first-order predicate logic formula with a theorem prover for predicate logic.
2. Check the other interpretations assuming symmetry *and* seriality of $\mathfrak{R}$.

If the accessibility relation $\mathfrak{R}$ is symmetric and transitive, we have for all worlds a, b: If b is accessible from a, then by symmetry, a is also accessible from b, hence by transitivity, a is accessible by itself. Thus, either a is isolated or $\mathfrak{R}$ is reflexive in a. In order to check a formula for unsatisfiability in symmetric and transitive interpretations, it is therefore again possible to split the proof into two cases:
1. Check the predicate logic interpretations.
2. Check the other interpretations assuming $\mathfrak{R}$ being an equivalence relation.

Examples 2.2.1: Possible worlds structures. ⊙ represents a world and ⇀ the accessibility relation:



■

In the sequel expressions like "serial interpretations" mean "logics with serial accessibility relations".

We define the semantics of our M-logic in the usual three steps:
1. An "interpretation" defines the meaning of the individual symbols.
2. The interpretation is turned into an interpreter, which assigns domain elements to terms.
3. A satisfiability relation is defined that assigns truth values to formulae.

Following authors like [Fitting 83], we introduce "frames" as the kernel of an interpretation. A frame describes the possible worlds structure, but it says nothing about the interpretation of variables and the actual world that has to be used for the interpretation of a formula. This information is added in so called "M-interpretations".

Definition 2.2.2 (M-Frames and M-Interpretations)

By an *M-frame* $\mathfrak{F}_M$ for the signature Σ_M we understand any triple $(\mathbb{D}, \mathfrak{I}, \mathfrak{R})$ where

- $\mathbb{D}$ is a non-empty set, the *domain* of discourse.
- $\mathfrak{I}$ is a set of *signature interpretations*. Each signature interpretation is an assignment of "values" to each function symbol and predicate symbol in Σ_M as follows:
  To each n-place function symbol a mapping from $\mathbb{D}^n$ to $\mathbb{D}$ is assigned.
  To each n-place predicate symbol an n-place relation over $\mathbb{D}^n$ is assigned.
- $\mathfrak{R}$ is a relation over $\mathfrak{I} \times \mathfrak{I}$.

By an *M-interpretation* $\mathfrak{I}_M$ for the signature Σ_M we understand any triple $(\mathfrak{F}_M, \mathfrak{I}, \mathfrak{a})$ where

- $\mathfrak{F}_M = (\mathbb{D}, \mathfrak{I}, \mathfrak{R})$ is an M-frame.
- $\mathfrak{I}$ is an element of $\mathfrak{I}$.
- $\mathfrak{a}$ is a *D-variable assignment*, i.e. a mapping $\mathbb{V}_D \rightarrow \mathbb{D}$. ■

Remarks. Since a "possible world" determines the interpretation of the function and predicate symbols, it is convenient to identify a possible world and the corresponding signature interpretation. Therefore we use in the sequel the notions 'world' and 'signature interpretation' as synonyms to denote the elements of $\mathfrak{I}$. Hence, $\mathfrak{R}$ is still the usual accessibility relation on worlds.
The variable assignment $\mathfrak{a}$ is irrelevant for the interpretation of closed formulae. It is used for recording the binding of quantified variables during a recursive descent into a formula. The world $\mathfrak{I}$ in an M-interpretation denotes the "actual" or "current" world that reflects the modal context for a subformula inside a formula (see def. 2.2.5).

As a notational convention we define:

Definition 2.2.3

Let $\mathfrak{a}$ be a variable assignment. We define $\mathfrak{a}[x/a]$ as:

$$\mathfrak{a}[x/a](y) := \begin{cases} \mathfrak{a}(y) & \text{if } y \neq x \\ a & \text{if } y = x \end{cases}$$

I.e. $\mathfrak{a}[x/a]$ is like $\mathfrak{a}$, except that it maps x to a.
Let $\aleph = (\ldots, \mathfrak{a}, \ldots)$ be any tuple containing a variable assignment. We use $\aleph[x/a]$ as an abbreviation for $(\ldots, \mathfrak{a}[x/a], \ldots)$. If $\aleph$ is a set of such tuples we use $\aleph[x/a]$ to denote the corresponding set where in each element $\mathfrak{a}$ is replaced by $\mathfrak{a}[x/a]$. ■

Definition 2.2.4 (Evaluation of D-Terms)

Let $\mathfrak{I}_M := (\mathfrak{F}_M, \mathfrak{I}, \mathfrak{a})$ be an M-interpretation for Σ_M.
$\mathfrak{I}_M$ can be turned into an homomorphism that evaluates D-terms in the actual world $\mathfrak{I}$ by:

$$\mathfrak{I}_M(t) := \begin{cases} \mathfrak{a}(t) & \text{if t is a D-variable symbol} \\ \mathfrak{I}(f)(\mathfrak{I}_M(t_1), \ldots, \mathfrak{I}_M(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Definition 2.2.5 (Satisfiability)

The satisfiability relation $\Vdash_M$ that determines the logical value of a formula in an M-interpretation is defined inductively over the structure of M-formulae as follows:

Let $\mathfrak{F}_M := (\mathbb{D}, \mathfrak{I}, \mathfrak{R})$ be an M-frame and let $\mathfrak{I}_M := (\mathfrak{F}_M, \mathfrak{I}, \mathfrak{a})$ be an M-interpretation, then

$\mathfrak{I}_M \Vdash_M P(t_1,\ldots,t_n)$	iff	$\mathfrak{I}(P)(\mathfrak{I}_M(t_1),\ldots,\mathfrak{I}_M(t_n))$.	(P is a predicate symbol.)
$\mathfrak{I}_M \Vdash_M \neg\mathcal{F}$	iff	not $\mathfrak{I}_M \Vdash_M \mathcal{F}$.	
$\mathfrak{I}_M \Vdash_M (\mathcal{F} \wedge \mathcal{G})$	iff	$\mathfrak{I}_M \Vdash_M \mathcal{F}$ and $\mathfrak{I}_M \Vdash_M \mathcal{G}$.	
$\mathfrak{I}_M \Vdash_M (\mathcal{F} \vee \mathcal{G})$	iff	$\mathfrak{I}_M \Vdash_M \mathcal{F}$ or $\mathfrak{I}_M \Vdash_M \mathcal{G}$.	
$\mathfrak{I}_M \Vdash_M \forall x\, \mathcal{F}$	iff	for every $a \in \mathbb{D}$: $\mathfrak{I}_M[x/a] \Vdash_M \mathcal{F}$.	
$\mathfrak{I}_M \Vdash_M \exists x\, \mathcal{F}$	iff	for some $a \in \mathbb{D}$: $\mathfrak{I}_M[x/a] \Vdash_M \mathcal{F}$.	
$\mathfrak{I}_M \Vdash_M \Box\mathcal{F}$	iff	for every $\mathfrak{I}' \in \mathfrak{I}$ with $\mathfrak{R}(\mathfrak{I}, \mathfrak{I}')$: $(\mathfrak{F}_M, \mathfrak{I}', \mathfrak{a}) \Vdash_M \mathcal{F}$.	
$\mathfrak{I}_M \Vdash_M \Diamond\mathcal{F}$	iff	there is some $\mathfrak{I}' \in \mathfrak{I}$ with $\mathfrak{R}(\mathfrak{I}, \mathfrak{I}')$ and $(\mathfrak{F}_M, \mathfrak{I}', \mathfrak{a}) \Vdash_M \mathcal{F}$.	

$\mathfrak{I}_M$ *satisfies* $\mathcal{F}$ iff $\mathfrak{I}_M \Vdash_M \mathcal{F}$ ($\mathfrak{F}_M$ satisfies $\mathcal{F}$ *in the world* $\mathfrak{I}$)

$\mathfrak{F}_M$ *satisfies* $\mathcal{F}$ iff it satisfies $\mathcal{F}$ in every world ■

Definition 2.2.6 (M-Models)

An M-frame is an *M-model* for an M-formula $\mathcal{F}$ iff it satisfies $\mathcal{F}$ in *some* world.

An M-formula $\mathcal{F}$ is a tautology iff every M-frame satisfies $\mathcal{F}$ in every world.

It is *satisfiable* iff there exists an M-model for $\mathcal{F}$.

It is *unsatisfiable* iff no M-model for $\mathcal{F}$ exists. ■

It is noted that in the definition of the semantics of the $\Box$-operator: "*for every* $\mathfrak{I}' \in \mathfrak{I}$ with $\mathfrak{R}(\mathfrak{I}, \mathfrak{I}')$..." the quantification 'for every' may be empty if $\mathfrak{R}$ is not serial and if there is no accessible world from $\mathfrak{I}$ at all. In this case $\Box\mathcal{F}$ is true for every $\mathcal{F}$, not because an atom evaluates to a truth value, but because a quantification quantifies over an empty set. This phenomenon which is not known in standard predicate logic causes the introduction of several new technical concepts and notions.

A Normal Form for Formulae in S5 Interpretations.

When the accessibility relation is an equivalence relation (modal logic S5), there is a certain normal form for M-formulae [Fitting 83]. Roughly speaking all modal operators except the innermost nested ones can be eliminated and a formula with at most one nested modal operator ("modal degree" one) is kept. The reason is that in S5 every world is accessible from every other world, hence $\Box\Box$ denotes the same set of worlds as $\Box$. Also the worlds accessed by $\Box\Diamond$ can all be taken to be the same as the one accessed by $\Diamond$ alone. In the sequel we shall assume for S5 interpretations that all formulae have been reduced to this normal form.

Chapter Three

P-Logic

P-logic ("P" for Predicate logic style) is a syntactic variant of M-logic where the modal operators are replaced by "world-paths". A world-path represents the modal context, i.e. the sequence of nested modal operators of the corresponding M-formula, and is attached as an additional argument to the function and predicate symbols. It records the world in which the term or formula is to be interpreted. Thus a world-path denotes a mapping from the initial world to the actual world and not to a domain element. This suggests to formulate P-logic as a two-sorted logic with the two basic disjoint sorts D (for Domain) and W (for Worlds).

All in all we need the following syntactic constituents:

1. The D-variables in P-logic are the same as in the M-logic.
   (Actually this is a consequence of our restriction to constant-domain interpretations.)
2. To each function symbol in M-logic there is a corresponding P-logic function symbol with an additional world-path argument. If the type of the M-logic symbol is $(D^n \rightarrow D)$, the type of the new function symbol is therefore $(W \times D^n \rightarrow D)$.
3. To each n-place predicate symbol in the M-logic there is a corresponding P-logic predicate symbol with an additional world-path argument.
   The type of this predicate symbol is therefore $((W \times D^n) \rightarrow \{True, False\})$.
4. The $\Diamond$-operator is translated into Skolem functions which are intended to map worlds to accessible worlds, possibly depending on some domain variables,
   Therefore we need function symbols of type $(D^n \rightarrow (W \rightarrow W))$.
5. The $\Box$-operator is translated into universally quantified variables. The objects that are to be assigned to such a variable are functions which map worlds to accessible worlds.
   The type of such a "W-variable" is therefore $(W \rightarrow W)$.

Since W-variables are functional variables, P-logic is actually a two-sorted monadic second order logic.

There is a one to one correspondence between the M-logic function and predicate symbols and the corresponding P-logic function and predicate symbols with the additional world-path argument. For convenience we do not use different names, such that we can use the same symbol for building M-logic and P-logic terms and atoms.

3.1 Syntax of P-Logic

As usual we begin with the definition of the signature.

Definition 3.1.1 (Signature of P-Logic)
The alphabet for building P-logic terms and formulae consists of the logical connectives $\wedge$, $\vee$, $\forall$ and the following disjoint sets of symbols:

$\mathbb{V}_D$ is a set of *D-variable* symbols, i.e. variable symbols of type D.
$\mathbb{F}_{D,n}$ is a set of *D-valued function symbols*, i.e. function symbols of type $(W \times D^n \rightarrow D)$.
$\mathbb{F}_D$ is the union of all D-valued function symbols.
$\mathbb{P}_n$ is a set of *predicate* symbols of type $((W \times D^n) \rightarrow \{True, False\})$
$\mathbb{P}$ is the union of all predicate symbols.

$\mathbb{V}_W$ is a set of *W-variable* symbols, i.e. variable symbols of type $(W \rightarrow W)$.
$\mathbb{F}_{W,n}$ is a set of *W-valued function* symbols, i.e. function symbols of type $(D^n \rightarrow (W \rightarrow W))$.
$\mathbb{F}_W$ is the union of all W-valued function symbols.

In case the signature is used to build terms, which are to be interpreted in logics with a symmetric accessibility relation we assume $\mathbb{V}_W$ and $\mathbb{F}_{W,n}$ to contain for each symbol s an associated "inverse" symbol s^{-1}. These symbols denote functions that move "backward" in the possible worlds structure. They do not occur in formulae, but only in substitutions. If for example there is a constant symbol $a \in \mathbb{F}_{W,0}$ denoting a function that maps for example a world w_1 to w_2, then a^{-1} denotes the inverse function which maps w_2 to w_1. We shall use these inverse functions only when the original functions are injective such that their inverse exists.

$\Sigma_P := (\mathbb{V}_D, \mathbb{F}_D, \mathbb{P}, \mathbb{V}_W, \mathbb{F}_W)$ is a *P-signature* . ■

Again Σ_P may also be used to denote the particular symbols which occur in a set of P-formulae.

Definition 3.1.2 (Terms, Atoms, Literals and Formulae)
Given a P-signature $\Sigma_P := (\mathbb{V}_D, \mathbb{F}_D, \mathbb{P}, \mathbb{V}_W, \mathbb{F}_W)$,

- the set of *D-terms* over Σ_P is defined as the least set such that:
  - (i) D-variable symbols are D-terms.
  - (ii) If $f \in \mathbb{F}_{D,n}$, $t_1,\ldots,t_n$ are D-terms and p is a world-path then $f(p, t_1,\ldots,t_n)$ is a D-term.
- The set of *W-terms* over Σ_P is defined as the least set such that:
  - (i) W-variable symbols are W-terms.
  - (ii) If $g \in \mathbb{F}_{W,n}$ and $t_1,\ldots,t_n$ are D-terms then $g(t_1,\ldots,t_n)$ is a W-term.
- A *world-path* is a (possibly empty) string of W-terms.
- If $P \in \mathbb{P}_n$, $t_1,\ldots,t_n$ are D-terms and wp is a world-path then $P(wp, t_1,\ldots,t_n)$ is a *P-atom.*
- A *P-literal* is either a P-atom or a negated P-atom.
- A *ground term* or *ground literal* is a term (literal) without variables.
- The set of *P-formulae* over Σ_P is defined to be the least set such that:
  - (i) A P-literal is a P-formula.

(ii) If $\mathcal{F}$ and $\mathcal{G}$ are P-formulae and x is a D- or W-variable symbol then

$\mathcal{F} \wedge \mathcal{G}$, $\mathcal{F} \vee \mathcal{G}$, $\forall x \mathcal{F}$ are P-formulae.

(No existential quantifier is necessary because the translation into P-logic syntax contains a skolemization step which eliminates all existential quantifiers.)

For convenience we assume again that the variables in the scope of quantifiers are appropriately renamed such that formulae like $\forall x\,(\mathcal{G} \wedge \forall x \mathcal{F})$ or $\forall x \mathcal{G} \vee \forall x \mathcal{F}$ do not occur.

Some auxiliary notions:

- A variable is *bound*, if it is in the scope of a quantifier.
- A variable is *free* in a term, atom or formula if it is not bound.
- $s \in t$ is true either if the term s is a subterm of the term, atom or formula t or if s is a prefix (leading part) of some world-path occurring in t, i.e. for example $[uv] \in [uvwa]$. ($\in$ is different from the membership predicate $\in$.)
- $Vars(s_1 \ldots s_n)$:= set of all free variables occurring in the objects (terms etc.) $s_1,\ldots,s_n$.
- $W\text{-}vars(s_1 \ldots s_n)$:= $Vars(s_1 \ldots s_n) \cap \mathbb{V}_W$ is the set of W-variables occurring in $s_1,\ldots,s_n$.
- $D\text{-}vars(s_1 \ldots s_n)$:= $Vars(s_1 \ldots s_n) \cap \mathbb{V}_D$ is the set of D-variables occurring in $s_1,\ldots,s_n$. ■

Examples for P-formulae and their M-logic counterparts: (Some parentheses are omitted.)

M-logic	P-logic	Signature
$\Box P$	$\forall w\ P[w]$	$P \in \mathbb{P}_0$, $w \in \mathbb{V}_W$
$\Diamond P$	$P[g]$	$P \in \mathbb{P}_0$, $g \in \mathbb{F}_{W,0}$
$\forall x \Diamond Q(x,a)$	$\forall x\ Q([h(x)], x, a[h(x)])$	$Q \in \mathbb{P}_2$, $x \in \mathbb{V}_D$, $a \in \mathbb{F}_{D,0}$, $h \in \mathbb{F}_{W,1}$
$\Box\forall x\,(S(x) \wedge \Box\, \exists y \Diamond S(y))$	$\forall v\ \forall x\,(S([v], x) \wedge \forall w\ S([vwk(x)], r([vw], x))$	
$\uparrow$ $\uparrow$ $\uparrow\uparrow$		$S \in \mathbb{P}_1$, $v,w \in \mathbb{V}_W$, $k \in \mathbb{F}_{W,1}$, $r \in \mathbb{F}_{D,1}$.
v w r k		

Notational Conventions

In the sequel the letters u, v, w, x, y, z from the end of the alphabet will be used to denote variables. In most cases, but not always, the letters x, y, z denote D-variables whereas u,v,w denote W-variables. Capital letters P, Q and S are predicate symbols. All other letters are used to denote terms or function symbols. In the particular context, it should be clear, what kind of objects are meant. Some letters are written outlined in order to emphasize that they denote sets.

It is noted that we are not interested in all possible P-formulae which can be built according to the above definition. We are only interested in a subclass whose formulae reflect the meaning of a corresponding M-formula. We shall see that this restriction finds its expression in the fact that all occurrences of a particular W-variable symbol in a formula have identical prefixes in the world-path strings. This is so, because a particular W-variable symbol w corresponds to a particular occurrence of a $\Box$-operator, and the prefix of w corresponds to the modal context (surrounding modal operators) of this occurrence of $\Box$.

In order to characterize the subclass of P-formulae, which correspond to M-formulae, we need some additional notation.

Definition 3.1.3 (The Prefix of a W-Variable)

Let t be a term, termlist, formula or list of formulae and let w be a W-variable.

The prefix of the variable w in the term t is:

$\text{prefix}(w, t) := \{[s\ .w] \mid [s\ .w] \in t\}$ ($[s\ .w] := [s_1 \ldots s_k w]$ in case $s = [s_1 \ldots s_k]$)

$\text{prefix}^*(w, t) := \{s \mid [s\ .w] \in t\}$

We shall omit the embracing set parentheses when the prefix of a W-variable is unique. ■

Examples for prefixes of W-variables.

prefix $(w, \forall w,u\ P[wu]) = [w]$. $\text{prefix}^*(w, \forall w,u\ P[wu]) = []$.

prefix $(w, \forall w,u\ P[wu] \vee Q[uw]) = \{[w], [uw]\}$ $\text{prefix}^*(w, \forall w,u\ P[wu] \vee Q[uw]) = \{[], [u]\}$. ■

The nesting of the universal quantifiers in a P-formula which corresponds to some M-formula is correlated with the nesting of the variables in W-terms: When a variable x occurs in the prefix of a W-variable v, the quantifier $\forall x$ must precede the quantifier $\forall u$. If this condition is violated, we cannot evaluate P-formulae properly in logics with non-serial accessibility relations. We therefore define an ordering relation on variables, which expresses the nesting of variables in world-paths and can be used to compare the nesting of variables with the nesting of quantifiers.

Definition 3.1.4 (An Ordering Relation on Variables.)

Let z be a (W- or D-) variable, w a W-variable and let $\mathcal{F}$ be a P-formula.

$z \leq_{\mathcal{F}} w$ iff $z \in \text{prefix}(w, \mathcal{F})$. ■

Examples for the $\leq_{\mathcal{F}}$ relation:

$\mathcal{F} = \forall w,u\ P[wu]$: $w \leq_{\mathcal{F}} u$

$\mathcal{G} = \forall w,x,u\ P[wg(x)u]$: $w \leq_{\mathcal{G}} u$ and $x \leq_{\mathcal{G}} u$, but not $w \leq_{\mathcal{G}} x$ ■

Lemma 3.1.5: $\leq_{\mathcal{F}}$ is an ordering relation for W-variables in a formula $\mathcal{F}$ with unique prefixes.

Proof: Reflexivity: Clearly $w \in \text{prefix}(w, \mathcal{F}) = \{[s\ .w] \mid [s\ .w] \in \mathcal{F}\}$ if w occurs in $\mathcal{F}$, i.e. $w \leq_{\mathcal{F}} w$.

Antisymmetry: Let $w \leq_{\mathcal{F}} u$ and $u \leq_{\mathcal{F}} w$, i.e. $w \in \text{prefix}(u, \mathcal{F})$ and $u \in \text{prefix}(w, \mathcal{F})$.

For some s, $[s\ .w]$ occurs as a subterm in $\text{prefix}(u, \mathcal{F})$, i.e. in $[s.w \ldots u]$. s cannot contain u as a proper subterm, otherwise $\text{prefix}(u, \mathcal{F})$ would consist of at least two elements, which contradicts the precondition that the prefixes are unique. Therefore w and u must be identical, i.e. $w = u$.

Transitivity: The transitivity of $\leq_{\mathcal{F}}$ follows immediately from the transitivity of $\in$ ("is subterm of") ■

Now we are ready to give a characterization of the subclass of "M-adjusted" P-formulae which correspond to M-formulae.

Definition 3.1.6 (M-Adjusted P-Formulae.)

A P-formula $\mathcal{G}$ is said to be *M-adjusted* iff

a) the prefixes of all W-variables occurring in $\mathcal{G}$ are unique and

b) the nesting of the quantifications is in accordance with the $\leq_{\mathcal{F}}$ relation, i.e.
for every subformula $\forall z\ \mathcal{F}$ occurring in $\mathcal{G}$: $\neg\, z \leq_{\mathcal{F}} v$ for every free variable v occurring in $\mathcal{F}$. ■

Examples and Counterexamples for M-adjusted P-formulae.

M-adjusted	not M-adjusted	
$\forall u,w\ P[uw]$	$\forall w,u\ P[uw]$	(violates condition b)
$\forall u,x,w\ P[ug(x)w]$	$\forall u,w\ P[uw] \vee P[w])$	(violates condition a)
$\forall x,u,w\ P[ug(x)w]$	$\forall u,w,x\ P\ [ug(x)w]$	(violates condition b) ■

3.2 Semantics of P-Logic

The semantics for P-logic is constructed as usual, except for the meaning of the W-valued symbols. W-valued function symbols are intended to convey the meaning of the $\Diamond$-operator whose interpretation is: From a given world $\mathfrak{I}$ there exists an accessible world $\mathfrak{I}'$ (possibly depending on some surrounding $\forall$-quantifiers) such that .... Thus, the object that is to be assigned to a W-valued function symbol must be a function - possibly depending on some domain arguments - which maps worlds to worlds.

The W-variables are intended to convey the meaning of the $\Box$-operator whose interpretation is: For all worlds $\mathfrak{I}'$ which are accessible from a given world $\mathfrak{I}$ .... A quantification $\forall w\ldots$ over a W-variable must therefore be restricted to some worlds; moreover the restriction has a dynamic character - it depends on the actual world. The only way to incorporate this restriction is to assign functions to W-variable symbols, which map worlds to accessible worlds.

A serious complication arises in non-serial models, where a world may have no accessible world at all. Functions mapping worlds to accessible worlds are not total in such models. To handle partial functions we use the notion of a strict ω-extension of a function (see for instance [Loeckx 84]). The idea is to add an artificial bottom element $\perp$ to the domain of the function and to change partial functions f into corresponding total functions f' (ω-extensions), which return $\perp$ if f is not defined for some argument value. f' is said to be strict if it returns $\perp$ whenever one of the arguments is $\perp$.

Definition 3.2.1 (World-Access Functions)

Given an M-frame $\mathfrak{F}_M = (\mathbb{D}, \mathfrak{I}, \mathfrak{R})$, a function $\phi : \mathfrak{I} \rightarrow \mathfrak{I}$ is called a *world-access function* iff

- ➤ It is a strict ω-extension.
- ➤ For every $\mathfrak{I} \in \mathfrak{I}$: $\phi(\mathfrak{I}) \neq \perp \Rightarrow \mathfrak{R}(\mathfrak{I}, \phi(\mathfrak{I}))$. ($\phi$ maps worlds to accessible worlds.)
- ➤ For every $\mathfrak{I} \in \mathfrak{I}$: Whenever there is at least one world accessible from $\mathfrak{I}$ then $\phi(\mathfrak{I}) \neq \perp$. ($\phi$ is *maximally defined.*)

Let $\mathfrak{I}_{\rightarrow}$ be the set of all world-access functions for $\mathfrak{I}$. ■

Definition 3.2.2 (P-Frames and P-Interpretations)

By a *P-frame* $\mathfrak{F}_P$ for a P-signature Σ_P we understand any tuple $(\mathfrak{F}_M, \mathfrak{I}_W)$ where

- $\mathfrak{F}_M = (\mathbb{D}, \mathfrak{I}, \mathfrak{R})$ is an M-frame.
  (I.e. an M-frame is the kernel of P-logic's semantics.)
- $\mathfrak{I}_W$ is a *W-signature interpretation* that assigns to each W-valued function symbol g of type $D^n \to (W \to W)$ a function γ: $\mathbb{D}^n \to \mathfrak{I}_{\to}$ that maps domain elements to world-access functions.
  In case $\mathfrak{R}$ is symmetric we need an additional property of $\mathfrak{I}_W$:
  Whenever $\mathfrak{I}_W$ assigns a function γ to a W-valued function symbol g such that $\gamma(a_1,\ldots,a_n)$ is injective for every $a_1,\ldots,a_n \in \mathbb{D}$ then $\mathfrak{I}_W$ must assign to the associated inverse symbol g^{-1} an *associated inverse* function γ^{-1} such that $\gamma(a_1,\ldots,a_n) \circ \gamma^{-1}(a_1,\ldots,a_n)$ is the identity mapping.

By a *P-interpretation* $\mathfrak{I}_P$ for the signature Σ_P we understand any tuple $(\mathfrak{F}_P, \mathfrak{I}_0, \mathfrak{a}, \mathfrak{w})$ where

- $\mathfrak{F}_P = ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$ is a P-frame.
- $\mathfrak{I}_0$ is an element of $\mathfrak{I}$. ($\mathfrak{I}_0$ is the "initial world".)
- $\mathfrak{a}$ is a D-variable assignment, i.e. an assignment of domain elements to D-variable symbols.
- $\mathfrak{w}$ is a *W-variable assignment*, i.e. an assignment of world-access functions to W-variable symbols.
  Whenever $\mathfrak{w}$ assigns to a W-variable symbol u an injective world-access function ϕ then $\mathfrak{w}$ must assign the inverse function ϕ^{-1} to the associated inverse W-variable symbol u^{-1}. ■

Now that the meaning of the signature has been defined, we have to say how a term is to be evaluated in a given interpretation: D-terms must be evaluated to domain elements, W-terms to world-access functions and world-paths to compositions of world-access functions.

Definition 3.2.3 (Evaluation of P-Logic Terms)

Let $\mathfrak{I}_P = ((\mathfrak{F}_P, \mathfrak{I}_W), \mathfrak{I}_0, \mathfrak{a}, \mathfrak{w})$ be a P-interpretation for the signature Σ_P, where $\mathfrak{F}_P = (\mathbb{D}, \mathfrak{I}, \mathfrak{R})$.
$\mathfrak{I}_P$ can be turned into a strict ω-extension of a function from D-terms to $\mathbb{D}$ and W-terms to $\mathfrak{I}_{\to}$ and world-paths to $\mathfrak{I} \to \mathfrak{I}$.
We define the evaluation $\mathfrak{I}_P(t)$ inductively on the structure of the term t.

1. W-terms: t = w and w is a W-variable symbol: $\mathfrak{I}_P(w) := \mathfrak{w}(w)$
   $t = g(t_1,\ldots,t_n)$: $\mathfrak{I}_P(g(t_1,\ldots,t_n)) := \mathfrak{I}_W(g)\ (\mathfrak{I}_P(t_1),\ldots,\mathfrak{I}_P(t_n))$
2. World-paths: t = [] $\mathfrak{I}_P([]) :=$ identity mapping
   t = [s . r] $\mathfrak{I}_P([s\,.\,r]) := \mathfrak{I}_P(s) \circ \mathfrak{I}_P(r)$
3. D-terms: t = x and x is a D-variable symbol: $\mathfrak{I}_P(x) := \mathfrak{a}(x)$
   $t = f(wp,t_1,\ldots,t_n)$: $\mathfrak{I}_P(f(wp,t_1,\ldots,t_n)) := \mathfrak{I}_P(wp)(\mathfrak{I}_0)(f)\ (\mathfrak{I}_P(t_1),\ldots,\mathfrak{I}_P(t_n))$ ■

Since W-terms are interpreted as world-access functions, the interpretation of a world-path - as the composition of the world-access functions assigned to its components - is a function that maps worlds $\mathfrak{I}$ to other worlds $\mathfrak{I}'$ which are accessible from $\mathfrak{I}$ in the transitive closure of $\mathfrak{R}$.

The world $\mathfrak{I}_0$ determines the "initial world" which is used to "start" the interpretation of a formula. The initial world is mapped by the function assigned to a world-path to the actual world. Thus for a given world-path wp, $\mathfrak{I}_P(wp)(\mathfrak{I}_0)$ denotes the world which corresponds to the actual world in M-logic.

Definition 3.2.4 (Satisfiability of M-Adjusted P-Formulae)

The satisfiability relation $\Vdash_P$ for the initial world of a P-interpretation is defined inductively over the construction of M-adjusted P-formulae.

Let $\mathfrak{F}_P := ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$ be a P-frame and let $\mathfrak{I}_P := (\mathfrak{F}_P, \mathfrak{I}_0, \mathfrak{a}, \mathfrak{w})$ be a P-interpretation:

$\mathfrak{I}_P \Vdash_P P(wp, t_1,\ldots,t_n)$ iff $\mathfrak{I}_P(wp)(\mathfrak{I}_0) \neq \perp$ and for i = 1...n $\mathfrak{I}_P(t_i) \neq \perp$ and $\mathfrak{I}_P(wp)(\mathfrak{I}_0)(P)\ (\mathfrak{I}_P(t_1),\ldots,\mathfrak{I}_P(t_n))$.

$\mathfrak{I}_P \Vdash_P \neg P(wp, t_1,\ldots,t_n)$ iff $\mathfrak{I}_P(wp)\ (\mathfrak{I}_0) \neq \perp$ and for i = 1...n $\mathfrak{I}_P(t_i) \neq \perp$ and not $\mathfrak{I}_P(wp)(\mathfrak{I}_0)(P)\ (\mathfrak{I}_P(t_1),\ldots,\mathfrak{I}_P(t_n))$.

$\mathfrak{I}_P \Vdash_P (\mathcal{F} \wedge \mathcal{G})$ iff $\mathfrak{I}_P \Vdash_P \mathcal{F}$ and $\mathfrak{I}_P \Vdash_P \mathcal{G}$.

$\mathfrak{I}_P \Vdash_P (\mathcal{F} \vee \mathcal{G})$ iff $\mathfrak{I}_P \Vdash_P \mathcal{F}$ or $\mathfrak{I}_P \Vdash_P \mathcal{G}$.

$\mathfrak{I}_P \Vdash_P \forall x\ \mathcal{F}$ where x is a D-variable iff for every $a \in \mathbb{D}$: $\mathfrak{I}_P\ [x/a] \Vdash_P \mathcal{F}$.

$\mathfrak{I}_P \Vdash_P \forall u\ \mathcal{F}$ where u is a W-variable

iff for wp := prefix*(u, $\mathcal{F}$): (The prefix is unique.)

either $\mathfrak{I}_P(wp)(\mathfrak{I}_0) = \perp$ and $\mathfrak{I}_P \Vdash_P \mathcal{F}$.

or $\mathfrak{I}_P(wp)(\mathfrak{I}_0) \neq \perp$ and for every $\phi \in \mathfrak{I}_{\rightarrow}$ with $(\mathfrak{I}_P(wp) \circ \phi)(\mathfrak{I}_0) \neq \perp$: $\mathfrak{I}_P[u/\phi] \Vdash_P \mathcal{F}$.

$\mathfrak{I}_P$ *satisfies* $\mathcal{F}$ iff $\mathfrak{I}_P \Vdash_P \mathcal{F}$ ($\mathfrak{F}_P$ satisfies $\mathcal{F}$ *in the world* $\mathfrak{I}$)

$\mathfrak{F}_P$ *satisfies* $\mathcal{F}$ iff it satisfies $\mathcal{F}$ in every world ■

Remarks

When the accessibility relation is serial, $\mathfrak{I}_P$ is a total function on terms. In this case we can greatly simplify the satisfiability relation:

$\mathfrak{I}_P \Vdash_P P(wp, t_1,\ldots,t_n)$ iff $\mathfrak{I}_P(wp)(\mathfrak{I}_0)(P)\ (\mathfrak{I}_P(t_1),\ldots,\mathfrak{I}_P(t_n))$.

$\mathfrak{I}_P \Vdash_P \neg\mathcal{F}$ iff not $\mathfrak{I}_P \Vdash_P \mathcal{F}$

$\mathfrak{I}_P \Vdash_P (\mathcal{F} \wedge \mathcal{G})$ iff $\mathfrak{I}_P \Vdash_P \mathcal{F}$ and $\mathfrak{I}_P \Vdash_P \mathcal{G}$.

$\mathfrak{I}_P \Vdash_P (\mathcal{F} \vee \mathcal{G})$ iff $\mathfrak{I}_P \Vdash_P \mathcal{F}$ or $\mathfrak{I}_P \Vdash_P \mathcal{G}$.

$\mathfrak{I}_P \Vdash_P \forall x\ \mathcal{F}$ where x is a D-variable iff for every $a \in \mathbb{D}$: $\mathfrak{I}_P[x/a] \Vdash_P \mathcal{F}$.

$\mathfrak{I}_P \Vdash_P \forall u\ \mathcal{F}$ where u is a W-variable iff for every $\phi \in \mathfrak{I}_{\rightarrow}$: $\mathfrak{I}_P[u/\phi] \Vdash_P \mathcal{F}$.

Since in serial interpretations all world access functions map worlds to accessible worlds, there need be no restriction on the W-variable assignments: "for *every* $\phi \in \mathfrak{I}_{\rightarrow}$: $\mathfrak{I}_P[u/\phi] \Vdash_P \mathcal{F}$." This is different from the interpretation of the $\Box$-operator in M-logic where an explicit restriction to accessible worlds must be included in the interpretation of a formula $\Box\mathcal{F}$.

If the accessibility relation is not serial, satisfiability as defined above would not work for P-logic formulae which are not M-adjusted. Consider as an example the P-formula $\forall v \forall u\ P[uv]$ which is not M-adjusted. The prefix of the outermost variable v is [u]. In order to check $\mathfrak{I}_P([u])(\mathfrak{I}_0)$ the value of u must be known, but no value has been assigned to u so far. Even if some default value could be assumed, one can easily construct examples where the two formulae $\forall v \forall u\ P[uv]$ and $\forall u \forall v\ P[uv]$ with exchanged quantifiers yield different truth values; and this is of course not acceptable for a decent logic.

The next lemma states that the P-satisfiability relation works as expected when applied to M-adjusted formulae.

Lemma 3.2.5 ($\Vdash_P$ **is Well Defined for M-Adjusted P-Formulae.**)
Let $\forall u\ \forall v\ \mathcal{F}$ and $\forall v\ \forall u\ \mathcal{F}$ be two M-adjusted P-formulae and let $\Im_P$ be a P-interpretation.

$\Im_P \Vdash_P \forall u\ \forall v\ \mathcal{F}$ iff $\Im_P \Vdash_P \forall v\ \forall u\ \mathcal{F}$.

Proof: The proof is based on a rather technical case analysis. The main arguments are: Since both formulae are M-adjusted, neither u can occur in the (unique) prefix of v (if v is a W-variable) nor v can occur in the (unique) prefix of u (if u is a W-variable). Therefore the evaluation of both prefixes is independent of the actual order of the variable assignments for u and v in the definition of $\Vdash_P$. ■

The previous lemma does not imply that quantifiers can be arbitrarily exchanged: they can only be exchanged when the accessibility relation is serial or the new formula is still M-adjusted. A definition of the P-satisfiability relation that avoids this effect, would have to ignore quantifications completely and extract the necessary information from the world-paths directly. This is possible, but much more complicated. In particular it is unnecessary, as the translation from M-logic to P-logic produces M-adjusted formulae automatically, and it can be shown that the resolution rule preserves this property.

Definition 3.2.6 (P-Model)
A P-frame is a *P-model* for a P-formula $\mathcal{F}$ iff it satisfies $\mathcal{F}$ in some world $\Im$.
(The corresponding P-interpretation that satisfies $\mathcal{F}$ will sometimes also be called a P-model for $\mathcal{F}$.)
A P-formula $\mathcal{F}$ is a tautology iff it is satisfied by every world in every P-frame.
It is *satisfiable* iff a P-model for $\mathcal{F}$ exists.
It is *unsatisfiable* iff no P-model for $\mathcal{F}$ exists. ■

Chapter Four

Translation from M-Logic to P-Logic

In this chapter the translation algorithm from M-logic to P-logic is defined and soundness and completeness results are presented. Soundness means that whenever an M-formula is satisfiable then the translated P-formula is satisfiable too. Completeness means that whenever the translated formula is satisfiable, the corresponding original M-logic version of the formula is also satisfiable. Together, these results are the basis for a complete proof procedure: In order to prove that an M-logic formula is unsatisfiable, it is sufficient to prove that the translated P-logic formula is unsatisfiable.

4.1 The Translation Algorithm

Definition 4.1.1 (Translation of M-Formulae into P-Logic Syntax)

1. Transformation of the signature:

Given an M-formula with the M-signature $\Sigma_M = (\mathbb{V}_D, \mathbb{F}_D, \mathbb{P})$ we construct an initial P-signature $\Sigma_P := (\mathbb{V}_D, \mathbb{F}_D, \mathbb{P}, \emptyset, \emptyset)$ for the translated formula. That is, we identify the D-variables of the M-signature and the D-variables of the P-signature, the D-valued function symbols of the M-signature and the D-valued function symbols of the P-signature, and the predicate symbols of the M-signature and the predicate symbols of the P-signature. The W-valued function symbols which replace the $\Diamond$-operator, the W-variable symbols which replace the $\Box$-operator as well as the Skolem functions for the existential quantifier are then added to Σ_P during the translation of the formula.

2. Translation of terms and formulae:

We define a translation function Π that takes an M-formula $\mathcal{F}$ and translates it into a P-formula $\Pi(\mathcal{F})$. The function Π also updates the P-signature Σ_P with the generated W-variables that replace the $\Box$-operator and the skolem functions for the $\exists$-quantifier and the $\Diamond$-operator. Π needs an auxiliary function π for the recursive descent into the M-formulae and terms. The function π has three arguments: The first is the actual formula to be translated, the second argument records the modal context in form of a world-path wp and the third argument collects the universally quantified D-variables in order to generate the appropriate Skolem terms.

Some notational conventions:

D-vars + x	denotes the concatenation of a list D-vars = $(x_1 \ldots x_n)$ with x yielding $(x_1 \ldots x_n\ x)$.
f(... D-vars)	denotes the term $f(\ldots x_1, \ldots, x_n)$ where D-vars = $(x_1 \ldots x_n)$
$\mathcal{F}[x \leftarrow s]$	means the replacement of all occurrences of x by s in the formula $\mathcal{F}$.
|D-vars|	denotes the length of the list D-vars.

Now we are ready to give the translation rules.

The toplevel call is: $\Pi(\mathcal{F}) := \pi(\mathcal{F}, [], ())$ where () is the empty list.

The translation rules for π are:

$\pi(\ \mathcal{F} \wedge \mathcal{G},\ \text{wp, D-vars}) := \pi(\mathcal{F}, \text{wp, D-vars}) \wedge \pi(\mathcal{G}, \text{wp, D-vars})$

$\pi(\ \mathcal{F} \vee \mathcal{G},\ \text{wp, D-vars}) := \pi(\mathcal{F}, \text{wp, D-vars}) \vee \pi(\mathcal{G}, \text{wp, D-vars})$

$\pi(\ \forall x \mathcal{F},\ \text{wp, D-vars}) := \forall x\ \pi(\mathcal{F}, \text{wp, D-vars} + x)$

$\pi(\ \Box \mathcal{F},\ \text{wp, D-vars}) := \forall u\ \pi(\mathcal{F}, [\text{wp} . u], \text{D-vars})$

u is added as a new W-variable symbol to $\mathbb{V}_W$ in Σ_P.

$\pi(\ \exists x \mathcal{F},\ \text{wp, D-vars}) := (\pi(\mathcal{F}, \text{wp, D-vars}))[x \leftarrow f(\text{wp, D-vars})]$

f is added as a new D-valued function symbol of type $(W \rightarrow (D^{|\text{D-vars}|} \rightarrow D))$ to $\mathbb{F}_D$ in Σ_P.

$\pi(\ \Diamond \mathcal{F},\ \text{wp, D-vars}) := \pi(\mathcal{F}, [\text{wp} . g(\text{D-vars})], \text{D-vars})$

g is added as a new W-valued function symbol of type $(D^{|\text{D-vars}|} \rightarrow (W \rightarrow W))$ to $\mathbb{F}_W$ in Σ_P.

$\pi(\ \pm P(t_1,\ldots,t_n),\ \text{wp, D-vars})$ where P is an n-place predicate symbol

$:= \pm P(\text{wp}, \pi(t_1, \text{wp, D-vars}),\ldots,\ \pi(t_n, \text{wp, D-vars}))$

$\pi(\ f(t_1,\ldots,t_n),\ \text{wp, D-vars})$ where f is an n-place function symbol

$:= f(\text{wp}, \pi(t_1, \text{wp, D-vars}),\ldots,\ \pi(t_n, \text{wp, D-vars}))$

$\pi(\ x,\ \text{wp, D-vars}) := x$ where x is a D-variable symbol. ■

Lemma 4.1.2

When $\mathcal{F}$ is a wellformed M-formula then $\Pi(\mathcal{F})$ is a wellformed M-adjusted P-formula.

Proof: The wellformedness of $\Pi(\mathcal{F})$ can be shown by structural induction. The M-adjustedness is an immediate consequence of the fact that a new W-symbol is introduced only once in the translation algorithm. The new symbol is used to build just one new world-path, and although it may be used as a prefix in different world-paths, its own prefixes never change. ■

The following technical lemma establishes a syntactic invariant for the translation function π, which is exploited in the soundness proof for π. It states that the wp-argument really collects the prefixes of the newly generated W-variables which are then shifted into the translated formula.

Lemma 4.1.3

Let $\mathcal{F}$ be an M-logic term or formula, wp a world-path and D-vars a list of D-variables generated by π during the translation.

For every W-variable symbol v occurring in $\pi(\mathcal{F}, \text{wp}, \text{D-vars})$:

$v \in wp$ implies $prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = prefix(v, wp)$ and

$v \notin wp$ implies $prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = [wp\ .\ v]$.

Proof: By induction on the structure of M-formulae.

Let v be a W-variable symbol occurring in $\pi(\mathcal{F}, \text{wp}, \text{D-vars})$.

Actually if $v \notin wp$ then $\mathcal{F} = \Box\mathcal{G}$.

In the remaining case analysis it is sufficient to consider only the first case $v \in wp$.

Base Case: $\mathcal{F}$ is a D-variable symbol.

This case is trivial because no W-variable symbol occurs in $\mathcal{F}$.

Induction Step: Let $\mathcal{F}$ be a compound term or formula.

By a case analysis on the structure of $\mathcal{F}$:

Case $\mathcal{F} = f(t_1,\ldots,t_n)$ is a D-term.

$prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = prefix(v, f(wp, \pi(t_1, wp, D\text{-vars}), \ldots, \pi(t_n, wp, D\text{-vars})))$

$= prefix(v,wp) \cup prefix(v, \pi(t_1, wp, D\text{-vars})) \cup \ldots \cup prefix(v, \pi(t_n, wp, D\text{-vars}))$

$= prefix(v,wp) \cup prefix(v,wp) \cup \ldots \cup prefix(v,wp)$ (induction hypothesis)

$= prefix(v,wp)$.

Case $\mathcal{F} = \pm P(t_1,\ldots,t_n)$. This case is identical with the previous one.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$ or $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$.

$prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = prefix(v, \pi(\mathcal{F}_1, wp, D\text{-vars})) \cup prefix(v, \pi(\mathcal{F}_2, wp, D\text{-vars}))$

$= prefix(v,wp) \cup prefix(v,wp)$ (induction hypothesis)

$= prefix(v,wp)$.

Case $\mathcal{F} = \Box\mathcal{G}$.

$prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = prefix(v, \forall u\ \pi(\mathcal{G}, [wp\ .\ u], D\text{-vars}))$

$= prefix(v, [wp\ .\ u])$ (induction hypothesis)

Subcase 1: $v = u$: $prefix(u, [wp\ .\ u]) = [wp\ .\ u]$

Subcase 2: $v \neq u$: $prefix(v, [wp\ .\ u]) = prefix(v, wp)$.

Case $\mathcal{F} = \Diamond\mathcal{G}$.

$prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = prefix(v, \pi(\mathcal{G}, [wp\ .\ g(D\text{-vars})], D\text{-vars}))$

$= prefix(v, [wp\ .\ g(D\text{-vars})])$ (induction hypothesis)

$= prefix(v, wp)$. ($v \notin$ D-vars)

Case $\mathcal{F} = \forall x\ \mathcal{G}$.

$prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = \forall x\ \pi(\mathcal{G}, wp, D\text{-vars} + x))$

$= prefix(v, wp)$. (induction hypothesis)

Case $\mathcal{F} = \exists x\ \mathcal{G}$.

$prefix(v, \pi(\mathcal{F}, wp, D\text{-vars})) = prefix(v, \pi(\mathcal{G}, wp, D\text{-vars})[x \leftarrow f(wp, D\text{-vars})])$

$= prefix(v, \pi(\mathcal{G}, wp, D\text{-vars})) \cup prefix(v, f(wp, D\text{-vars}))$

$= prefix(v, wp)$ (induction hypothesis, $x \notin wp$) ■

Our second main technical lemma below establishes another invariant for the translation function π, which is exploited in the completeness proof for π. It states that any P-interpretation $\Im_P$ satisfying the translated formula $\mathcal{F}_P := \pi(\mathcal{F}, wp, D\text{-vars})$ evaluates the world-path wp to a function that maps the initial world $\Im_0$ to an actual world, and not to $\perp$. The main argument is that wp is a prefix of *all* world-paths occurring in $\mathcal{F}_P$. Therefore not a single atom in $\mathcal{F}_P$ could be satisfied if $\Im_P(wp)(\Im_0) = \perp$.

Lemma 4.1.4

Let $\mathcal{F}$ be an M-formula, wp a world-path and D-vars a list of D-variables as generated during the translation performed by π.

For every P-interpretation $\Im_P$ with initial world $\Im_0$ which satisfies $\mathcal{F}_P := \pi(\mathcal{F}, wp, D\text{-vars})$:

$\Im_P(wp)(\Im_0) \neq \perp$.

Proof: By induction on the structure of M-formulae. Assume $\Im_P \Vdash_P \mathcal{F}_P$.

Base Case: $\mathcal{F} = \pm P(t_1,\ldots,t_n)$ and P is a predicate symbol.

$\Im_P \Vdash_P \mathcal{F}_P = \pm P(wp, \pi(t_1, wp, D\text{-vars}), \ldots, \pi(t_n, wp, D\text{-vars}))$

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. (def. 3.2.4)

Induction Step: We perform a case analysis according to the structure of $\mathcal{F}$.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$

$\Im_P \Vdash_P \pi(\mathcal{F}_1, wp, D\text{-vars}) \wedge \pi(\mathcal{F}_2, wp, D\text{-vars})$

$\Rightarrow \Im_P \Vdash_P \pi(\mathcal{F}_1, wp, D\text{-vars})$ (and $\Im_P \Vdash_P \pi(\mathcal{F}_2, wp, D\text{-vars})$) (def. 3.2.4)

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. (induction hypothesis)

Case $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$.

$\Im_P \Vdash_P \pi(\mathcal{F}_1, wp, D\text{-vars}) \vee \pi(\mathcal{F}_2, wp, D\text{-vars})$

$\Rightarrow \Im_P \Vdash_P \pi(\mathcal{F}_1, wp, D\text{-vars})$ or $\Im_P \Vdash_P \pi(\mathcal{F}_2, wp, D\text{-vars})$ (def. 3.2.4)

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. (induction hypothesis applied to the positive case)

Case $\mathcal{F} = \Box\mathcal{G}$. The translation rule is: $\pi(\mathcal{F}, wp, D\text{-vars}) = \forall u\, \pi(\mathcal{G}, [wp\,.\,u], D\text{-vars})$.

Case 1: $\Im_P(prefix^*(u, \pi(\mathcal{G}, [wp\,.\,u], D\text{-vars})))(\Im_0) = \perp$

$\Rightarrow \Im_P \Vdash_P \pi(\mathcal{G}, [wp\,.\,u], D\text{-vars}))$ (def. 3.2.4)

$\Rightarrow \Im_P([wp\,.\,u])(\Im_0) \neq \perp$ (induction hypothesis)

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. (world-access functions are strict)

Case 2: $\Im_P(prefix^*(u, \pi(\mathcal{G}, [wp\,.\,u], D\text{-vars})))(\Im_0) \neq \perp$

$\Rightarrow \Im_P(prefix^*(u, [wp\,.\,u]))(\Im_0) = \Im_P(wp)(\Im_0) \neq \perp$ (lemma 4.1.3)

Case $\mathcal{F} = \Diamond\mathcal{G}$. The translation rule is: $\pi(\mathcal{F}, wp, D\text{-vars}) = \pi(\mathcal{G}, [wp\,.\,g(D\text{-vars})], D\text{-vars})$

$\Im_P([wp\,.\,g(D\text{-vars})])(\Im_0) \neq \perp$ (induction hypothesis)

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. (world-access functions are strict)

Case $\mathcal{F} = \forall x\, \mathcal{G}$. The translation rule is: $\pi(\mathcal{F}, wp, D\text{-vars}) = \forall x\, \pi(\mathcal{G}, wp, D\text{-vars}+ x)$

$\Im_P[x/a] \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars}+ x)$ for any $a \in \mathbb{D}$ (def. 3.2.4)

$\Rightarrow \Im_P[x/a](wp)(\Im_0) \neq \perp$ (induction hypothesis)

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. ($x \notin wp$)

Case $\mathcal{F} = \exists x\, \mathcal{G}$. The translation rule is: $\pi(\mathcal{F}, wp, D\text{-vars}) = (\pi(\mathcal{G}, wp, D\text{-vars}))[x \leftarrow f(wp, D\text{-vars})]$

Assume $\Im_P(wp)(\Im_0) = \perp$, i.e. $\Im_P(f(wp, D\text{-vars})) = \perp$

$\Rightarrow$ The fact $\Im_P \Vdash_P \pi(\exists x\, \mathcal{G}, wp, D\text{-vars})$ does not depend on the assignment to x.

$\Rightarrow \Im_P \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars})$

$\Rightarrow \Im_P(wp)(\Im_0) \neq \perp$. $\lightning$ (induction hypothesis)

$\Rightarrow$ The assumption is wrong, i.e. $\Im_P(wp)(\Im_0) \neq \perp$ must hold. ■

4.2 Soundness of the Translation

In order to establish the soundness of the translation function Π, we show that whenever an M-formula $\mathcal{F}$ has an M-model, the translated P-formula $\Pi(\mathcal{F})$ has a P-model. The essential idea in the (constructive) proof is to augment the functionality of the translation functions Π and π such that in parallel with the translation of an M-formula to a P-formula, an M-frame is translated to a P-frame. If we can then show that an M-frame satisfying $\mathcal{F}$ is translated into a P-frame satisfying $\Pi(\mathcal{F})$ we are done.

Since a P-frame $\mathbb{F}_P = (\mathbb{F}_M, \mathfrak{I}_W)$ consists of an M-frame and the interpretation of the W-valued function symbols, the only thing the augmented translation functions have to do, is to define the interpretation of the generated Skolem functions. Just as for the generation of the Skolem terms, information about the embracing universal quantifiers was made available with the D-vars argument of the function π, for the definition of their semantics we need in the augmented translation function an additional argument $\mathcal{K}$ that takes information about the assignment of values to the variables in the current world. This $\mathcal{K}$-argument takes for a particular world $\mathfrak{I}_0$ in the M-frame (the world that satisfies the formula) a set with elements $(\mathfrak{I}, \mathfrak{d}, \mathfrak{w})$ where $\mathfrak{I}$ is the actual world that is determined by the nesting of the modal operators in $\mathcal{F}$, $\mathfrak{d}$ is the assignment of values to the D-variables and $\mathfrak{w}$ is the assignment of values to the generated W-variables.

In the sequel we use the following notational convention:
If $\mathbb{F}_P = (\mathbb{F}_M, \mathfrak{I}_W)$ is the generated P-frame, $\mathcal{K} := (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathcal{K}$ then
let $\mathcal{K}_M := (\mathbb{F}_M, \mathfrak{I}, \mathfrak{d})$ denote the corresponding M-interpretation and
let $\mathcal{K}_P := ((\mathbb{F}_M, \mathfrak{I}_W), \mathfrak{I}_0, \mathfrak{d}, \mathfrak{w})$ denote the corresponding P-interpretation.

Definition 4.2.1 (The augmented translation functions)
Let $\mathcal{F}$ be a closed M-formula and let $\mathbb{F}_M := (\mathbb{D}, \mathfrak{I}, \mathfrak{R})$ be an M-frame over the signature of $\mathcal{F}$.

We define the augmented toplevel translation function Π as follows:
The arguments are $\Pi(\mathcal{F}, \mathbb{F}_M, \mathfrak{I}_0)$.
Π creates an initial P-frame $\mathbb{F}_P := (\mathbb{F}_M, \emptyset)$ and calls the augmented recursive translation function π:
$\pi(\mathcal{F}, [], (), \{(\mathfrak{I}_0, \emptyset, \emptyset)\})$
The initial P-frame is updated in parallel with the generation of the skolem functions.
Π returns the translated formula and the generated and updated P-frame.

The augmented translation rules realized by π are:
$\pi(\mathcal{F} \wedge \mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}) := \pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K}) \wedge \pi(\mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K})$.
$\pi(\mathcal{F} \vee \mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}) := \pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K}_1) \vee \pi(\mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}_2)$
where $\mathcal{K}_1 := \{\mathcal{K} \in \mathcal{K}_1 \mid \mathcal{K}_M \Vdash_M \mathcal{F}\}$ and $\mathcal{K}_2 := \{\mathcal{K} \in \mathcal{K}_2 \mid \mathcal{K}_M \Vdash_M \mathcal{G}\}$
$\pi(\forall x \mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K}) := \forall x\, \pi(\mathcal{F}, \text{wp}, \text{D-vars} + x, \mathcal{L})$
where $\mathcal{L} := \{\mathcal{K}[x/a] \mid \mathcal{K} \in \mathcal{K} \text{ and } a \in \mathbb{D}\}$.
$\pi(\Box \mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K}) := \forall u\, \pi(\mathcal{F}, [\text{wp} \,.\, u], \text{D-vars}, \mathcal{L})$
u is added as a new W-variable symbol to $\mathbb{V}_W$ in Σ_P. (see def. 4.1.1)
$\mathcal{L} := \{(\phi(\mathfrak{I}), \mathfrak{d}, \mathfrak{w}[u/\phi]) \mid (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathcal{K}, \phi \in \mathfrak{I}_{\rightarrow} \text{ with } \phi(\mathfrak{I}) \neq \perp\}$.
$\pi(\exists x \mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K}) := (\pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{L}))[x \leftarrow f(\text{wp}, \text{D-vars})]$
f is added as a new D-valued function symbol of type $W \rightarrow (D^{|\text{D-vars}|} \rightarrow D)$ to $\mathbb{F}_D$ in Σ_P.

Among all possible interpretations for f select one that satisfies the following condition and add it to the signature interpretations in $\mathbf{F}_P$:

For every $\mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}$

Among the $a \in \mathbb{D}$ with $\mathcal{K}_M[x/a] \Vdash_M \mathcal{F}$ there is an $\mathfrak{a}$ with $\mathcal{K}_P(f(\text{wp. D-vars})) = \mathfrak{a}$.

$\mathcal{L} := \{\mathcal{K}[x/a] \mid a \in \mathbb{D}, \mathcal{K} \in \mathfrak{K}, \mathcal{K}_M[x/a] \Vdash_M \mathcal{F}\}$.

$\pi(\Diamond \mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K}) := \pi(\mathcal{F}, [\text{wp} \,.\, g(\text{D-vars})], \text{D-vars}, \mathcal{L})$

g is added as a new W-valued function symbol of type $D^{|\text{D-vars}|} \to (W \to W)$ to $\mathbb{F}_W$ in Σ_P.

Among all possible interpretations for g select one that satisfies the following condition and add it to the $\mathfrak{I}_W$ component in $\mathbf{F}_P$:

For every $\mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}$

Among the $\mathfrak{I}' \in \mathfrak{I}$ with $\mathfrak{R}(\mathfrak{I}, \mathfrak{I}')$ and $(\mathfrak{I}', \mathfrak{d}, \mathfrak{w})_M \Vdash_M \mathcal{F}$ there is an $\mathfrak{I}'$ with

$\mathcal{K}_P(g(\text{D-vars}))(\mathfrak{I}) = \mathfrak{I}'$.

$\mathcal{L} := \{(\mathfrak{I}', \mathfrak{d}, \mathfrak{w}) \mid \mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}\ \mathfrak{R}(\mathfrak{I}, \mathfrak{I}'),\ (\mathfrak{I}', \mathfrak{d}, \mathfrak{w})_M \Vdash_M \mathcal{F},\ \mathcal{K}_P(g(\text{D-vars}))(\mathfrak{I}) = \mathfrak{I}'\}$

$\pi(\pm P(t_1,\ldots,t_n), \text{wp}, \text{D-vars}, \mathcal{K})$ where P is an n-place predicate symbol

$:= \pm P(\text{wp}, \pi(t_1, \text{wp}, \text{D-vars}, \mathcal{K}), \ldots, \pi(t_n, \text{wp}, \text{D-vars}, \mathcal{K}))$

$\pi(f(t_1,\ldots,t_n), \text{wp}, \text{D-vars}, \mathcal{K})$ where f is an n-place function symbol

$:= f(\text{wp}, \pi(t_1, \text{wp}, \text{D-vars}, \mathcal{K}), \ldots, \pi(t_n, \text{wp}, \text{D-vars}, \mathcal{K}))$

$\pi(x, \text{wp}, \text{D-vars}) := x$ where x is a D-variable symbol. ■

The lemma below states that the variable assignment and the actual world in the $\mathfrak{K}$-argument of the augmented translation function π are well defined, i.e. when the initial M-frame is an M-model for the formula to be translated, then the M-interpretations that can be obtained from the elements of $\mathfrak{K}$ satisfy the corresponding subformula that is currently being translated.

Lemma 4.2.2: When $\mathbf{F}_M$ is an M-frame satisfying the M-formula $\mathcal{H}$ in the world $\mathfrak{I}_0$ then for each recursive call $\pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K})$ during the translation $\Pi(\mathcal{H}, \mathbf{F}_M, \mathfrak{I}_0)$ the following invariant holds as long as $\mathcal{F}$ is a formula: For every $\mathcal{K} \in \mathfrak{K}$ $\mathcal{K}_M \Vdash_M \mathcal{F}$.

Proof: By induction on the recursion depth.

Base Case: Recursion depth = 0: This is just the initial call $\pi(\mathcal{H}, [], (), \{(\mathfrak{I}_0, \emptyset, \emptyset)\})$

Since $\mathbf{F}_M$ is assumed to satisfy $\mathcal{H}$ in $\mathfrak{I}_0$, $(\mathfrak{I}_0, \emptyset, \emptyset)_M \Vdash_M \mathcal{H}$ holds by definition.

Induction Step: Let the recursion depth be greater than 0.

Let $\pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K})$ be the actual call to π.

The induction hypothesis states: For every $\mathcal{K} \in \mathfrak{K}$ $\mathcal{K}_M \Vdash_M \mathcal{F}$.

In order to show that the statement holds for the next recursion step we must perform a case analysis according to the structure of $\mathcal{F}$ and analyze the corresponding translation rule.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \wedge \mathcal{F}_2, \text{wp}, \text{D-vars}, \mathcal{K}) := \pi(\mathcal{F}_1, \text{wp}, \text{D-vars}, \mathcal{K}) \wedge \pi(\mathcal{F}_2, \text{wp}, \text{D-vars}, \mathcal{K})$.

The statement follwos immediately from the induction hypothesis and def. 2.2.5.

Case $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \vee \mathcal{F}_2, \text{wp}, \text{D-vars}, \mathcal{K}) := \pi(\mathcal{F}_1, \text{wp}, \text{D-vars}, \mathcal{K}_1) \vee \pi(\mathcal{F}_2, \text{wp}, \text{D-vars}, \mathcal{K}_2)$ where

$\mathcal{K}_1 := \{\mathcal{K} \in \mathfrak{K} \mid \mathcal{K}_M \Vdash_M \mathcal{F}_1\}$ and $\mathcal{K}_2 := \{\mathcal{K} \in \mathfrak{K} \mid \mathcal{K}_M \Vdash_M \mathcal{F}_2\}$.

Thus, the condition is explicitly enforced in the definition of π.

Case $\mathcal{F} = \forall x \mathcal{G}$. The translation rule for this case is:

$\pi(\forall x \mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}) = \forall x\, \pi(\mathcal{G}, \text{wp}, \text{D-vars} + x, \mathcal{L})$ where

$\mathcal{L} := \{\mathcal{K}[x/a] \mid \mathcal{K} \in \mathfrak{K} \text{ and } a \in \mathbb{D}\}$

The induction hypothesis implies: For every $\mathcal{K} \in \mathfrak{K}$ $\mathcal{K}_M[x/a] \Vdash_M \mathcal{G}$, (def. 2.2.5)

i.e. the condition holds for $\mathcal{L}$.

Case $\mathcal{F} = \Box\mathcal{G}$. The translation rule for this case is:

$\pi(\Box\mathcal{G}, wp, D\text{-vars}, \mathfrak{K}) := \forall u\, \pi(\mathcal{G}, [wp\ .\ u], D\text{-vars}, \mathcal{L})$ where

$\mathcal{L} = \{(\phi(\mathfrak{I}), \mathfrak{d}, \mathfrak{w}[u/\phi]) \mid (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}, \phi \in \mathfrak{I}_{\to}$ with $\phi(\mathfrak{I}) \neq \perp\}$.

Let $\mathcal{L} = (\phi(\mathfrak{I}), \mathfrak{d}, \mathfrak{w}[u/\phi]) \in \mathcal{L}$.

Since $\phi(\mathfrak{I}) \neq \perp$ holds and since ϕ is a world-access function, $\mathfrak{R}(\mathfrak{I}, \phi(\mathfrak{I}))$ holds.

The induction hypothesis therefore implies: $\mathcal{L}_M \Vdash_M \mathcal{G}$. (def. 2.2.5)

Case $\mathcal{F} = \exists x\, \mathcal{G}$. The translation rule for this case is:

$\pi(\exists x\mathcal{G}, wp, D\text{-vars}, \mathfrak{K}) := (\pi(\mathcal{G}, wp, D\text{-vars}, \mathcal{L}))[x \leftarrow f(wp, D\text{-vars})]$ where

$\mathcal{L} := \{\mathcal{K}[x/a] \mid a \in \mathbb{D}, \mathcal{K} \in \mathfrak{K}, \mathcal{K}_M[x/a] \Vdash_M \mathcal{G}\}$,

i.e. with "$\mathcal{K}_M[x/a] \Vdash_M \mathcal{G}$" the condition is explicitly enforced in the definition of π.

Case $\mathcal{F} = \Diamond\mathcal{G}$. The translation rule for this case is:

$\pi(\Diamond\mathcal{G}, wp, D\text{-vars}, \mathfrak{K}) := \pi(\mathcal{G}, [wp\ .\ g(D\text{-vars})], D\text{-vars}, \mathcal{L})$

$\mathcal{L} := \{(\mathfrak{I}', \mathfrak{d}, \mathfrak{w}) \mid \mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}\ \mathfrak{R}(\mathfrak{I}, \mathfrak{I}'),\ (\mathfrak{I}', \mathfrak{d}, \mathfrak{w})_M \Vdash_M \mathcal{G},\ \mathcal{K}_P(g(D\text{-vars}))(\mathfrak{I}) = \mathfrak{I}'\}$,

i.e. with "$(\mathfrak{I}', \mathfrak{d}, \mathfrak{w})_M \Vdash_M \mathcal{G}$" the condition is explicitly enforced in the definition of π. ■

The next lemma ensures that for each element in the $\mathfrak{K}$-argument of the augmented translation function π there is a coincidence between the actual world as defined by the nesting of the modal operators and the interpretation of the constructed world-path when applied to the initial world.

Lemma 4.2.3 When $\mathfrak{F}_M$ is an M-frame satisfying the M-formula $\mathcal{H}$ in the world $\mathfrak{I}_0$ then for each recursive call $\pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{K})$ during the translation $\Pi(\mathcal{H}, \mathfrak{F}_M, \mathfrak{I}_0)$ the following invariant holds as long as $\mathcal{F}$ is a formula:

For every $\mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}$ $\mathcal{K}_P(wp)(\mathfrak{I}_0) = \mathfrak{I}$.

Proof: By induction on the recursion depth.

Base Case: Recursion depth = 0: This is just the initial call $\pi(\mathcal{H}, [], (), \{(\mathfrak{I}_0, \emptyset, \emptyset)\})$.

Therefore $\mathcal{K}_P([])(\mathfrak{I}_0) = \mathfrak{I}_0$ holds by definition.

Induction Step: Let the recursion depth be greater than 0.

Let $\pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{K})$ be the recursive call to π.

The induction hypothesis states:

For every $\mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}$ $\mathcal{K}_P(wp)(\mathfrak{I}_0) = \mathfrak{I}$.

In order to show that the statement holds for the next recursion step we must perform a case analysis according to the structure of $\mathcal{F}$ and analyze the corresponding translation rule.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \wedge \mathcal{F}_2, wp, D\text{-vars}, \mathfrak{K}) := \pi(\mathcal{F}_1, wp, D\text{-vars}, \mathfrak{K}) \wedge \pi(\mathcal{F}_2, wp, D\text{-vars}, \mathfrak{K})$.

The statement follows immediately from the induction hypothesis.

Case $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \vee \mathcal{F}_2, wp, D\text{-vars}, \mathfrak{K}) := \pi(\mathcal{F}_1, wp, D\text{-vars}, \mathfrak{K}_1) \vee \pi(\mathcal{F}_2, wp, D\text{-vars}, \mathfrak{K}_2)$ where

$\mathfrak{K}_1 := \{\mathcal{K} \in \mathfrak{K} \mid \mathcal{K}_M \Vdash_M \mathcal{F}_1\}$ and $\mathfrak{K}_2 := \{\mathcal{K} \in \mathfrak{K} \mid \mathcal{K}_M \Vdash_M \mathcal{F}_2\}$.

The statement follows immediately from the induction hypothesis.

Case $\mathcal{F} = \forall x\mathcal{G}$. The translation rule for this case is:

$\pi(\forall x\mathcal{G}, wp, D\text{-vars}, \mathfrak{K}) = \forall x\, \pi(\mathcal{G}, wp, D\text{-vars} + x, \mathcal{L})$ where

$\mathcal{L} := \{\mathcal{K}[x/a] \mid \mathcal{K} \in \mathfrak{K}$ and $a \in \mathbb{D}\}$

Since formulae are standardized apart, x does not occur in wp and the induction hypothesis can

immediately be applied.

Case $\mathcal{F} = \Box\mathcal{G}$. The translation rule for this case is:

$\pi(\Box\mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}) := \forall u\ \pi(\mathcal{G}, [\text{wp} \,.\, u], \text{D-vars}, \mathcal{L})$ where

$\mathcal{L} = \{(\Im_0, \{(\phi(\Im), \mathfrak{d}, \mathfrak{w}[u/\phi]) \mid (\Im, \mathfrak{d}, \mathfrak{w}) \in \mathcal{K}, \phi \in \Im_{\rightarrow} \text{ with } \phi(\Im) \neq \bot\}$

Let $\mathcal{L} = (\phi(\Im), \mathfrak{d}, \mathfrak{w}[u/\phi]) \in \mathcal{L}$.

$\mathcal{L}_P([\text{wp} \,.\, u])\,(\Im_0)$

$= (\mathcal{L}_P(\text{wp}) \circ \mathfrak{w}[u/\phi](u))\,(\Im_0)$ (def. 3.2.3)

$= \phi(\Im)$. (induction hypothesis)

Case $\mathcal{F} = \exists x\ \mathcal{G}$. The translation rule for this case is:

$\pi(\exists x\mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}) := (\pi(\mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{L}))[x \leftarrow f(\text{wp}, \text{D-vars})]$ where

$\mathcal{L} := \{\mathcal{K}[x/a] \mid a \in \mathbb{D}, \mathcal{K} \in \mathcal{K}, \mathcal{K}_M[x/a] \Vdash_M \mathcal{G}\}$.

Since x does not occur in wp, the induction hypothesis can immediately be applied.

Case $\mathcal{F} = \Diamond\mathcal{G}$. The translation rule for this case is:

$\pi(\Diamond\mathcal{G}, \text{wp}, \text{D-vars}, \mathcal{K}) := \pi(\mathcal{G}, [\text{wp} \,.\, g(\text{D-vars})], \text{D-vars}, \mathcal{L})$

$\mathcal{L} := \{(\Im', \mathfrak{d}, \mathfrak{w}) \mid \mathcal{K} = (\Im, \mathfrak{d}, \mathfrak{w}) \in \mathcal{K}, \mathfrak{R}(\Im, \Im'), (\Im', \mathfrak{d}, \mathfrak{w})_M \Vdash_M \mathcal{G}, \mathcal{K}_P(g(\text{D-vars}))(\Im) = \Im'\}$

Let $\mathcal{L} = (\Im', \mathfrak{d}, \mathfrak{w}) \in \mathcal{L}$.

$\mathcal{L}_P([\text{wp} \,.\, g(\text{D-vars})])(\Im_0)$

$= (\mathcal{L}_P(\text{wp}) \circ \mathcal{L}_P(g(\text{D-vars})))\,(\Im_0)$ (def. 3.2.3)

$= \mathcal{L}_P(g(\text{D-vars}))\,(\Im)$ (induction hypothesis)

$= \Im'$. ■

Now we are ready to show the main part of the soundness proof for the translation: At each level of the translation operation, the generated P-interpretations satisfy the translated subformula.

Lemma 4.2.4 When $\mathfrak{F}_M$ is an M-frame satisfying the M-formula $\mathcal{H}$ in the world $\Im_0$ then for each recursive call $\mathcal{F}_P := \pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathcal{K})$ during the translation $\Pi(\mathcal{H}, \mathfrak{F}_M, \Im_0)$ the following invariant holds as long as $\mathcal{F}$ is a formula: For every $\mathcal{K} = (\Im, \mathfrak{d}, \mathfrak{w}) \in \mathcal{K}$ $\mathcal{K}_P \Vdash_P \mathcal{F}_P$.

Proof: By induction on the structure of M-formulae.

First Base Case: $\mathcal{F} = P(t_1,\ldots,t_n)$ is an atom.

Let $\mathcal{K} = (\Im, \mathfrak{d}, \mathfrak{w}) \in \mathcal{K}$

According to the translation rule for the atomic case we must show:

$\mathcal{K}_P \Vdash_P P(\text{wp}, \pi(t_1, \text{wp}, \text{D-vars}, \mathcal{K}),\ldots, \pi(t_n, \text{wp}, \text{D-vars}, \mathcal{K}))$.

Neither wp nor $\mathcal{K}$ changes during the recursive descent into the terms,

therefore, with lemma 4.2.3, $\mathcal{K}_P(\text{wp})(\Im_0) = \Im$ holds for the translation of the terms. (❋)

First of all we must prove by induction on the structure of M-logic terms:

$\mathcal{K}_P(\pi(t, \text{wp}, \text{D-vars}, \mathcal{K})) = \mathcal{K}_M(t)$ for $t \in \{t_1,\ldots,t_n\}$. (❖)

Base case: t is a D-variable: $\mathcal{K}_P(\pi(t, \text{wp}, \text{D-vars}, \mathcal{K})) = \mathcal{K}_P(t) = \mathfrak{d}(t) = \mathcal{K}_M(t)$. (def. 2.2.4, 3.2.3)

Induction step: $t = f(s_1,\ldots,s_m)$:

$\mathcal{K}_P(\pi(t, \text{wp}, \text{D-vars}, \mathcal{K}))$

$= \mathcal{K}_P(f(\text{wp}, \pi(s_1, \text{wp}, \text{D-vars}, \mathcal{K}),\ldots,\pi(s_m, \text{wp}, \text{D-vars}, \mathcal{K}))$ (translation rule)

$= \mathcal{K}_P(\text{wp})(\Im_0)(f)\,(\mathcal{K}_P(\pi(s_1, \text{wp}, \text{D-vars}, \mathcal{K})),\ldots, \mathcal{K}_P(\pi(s_m, \text{wp}, \text{D-vars}, \mathcal{K}))$ (def 3.2.3)

$= \Im(f)\,(\mathcal{K}_M(s_1),\ldots, \mathcal{K}_M(s_m))$ (induction hypothesis and ❋)

$= \mathcal{K}_M(t)$.

Now we can conclude:

$\mathcal{K}_M \Vdash_M P(t_1,\ldots,t_n)$ (lemma 4.2.2)

$\Rightarrow \mathfrak{I}(P)\ (\mathcal{K}_M(t_1),\ldots,\ \mathcal{K}_M(t_n))$ (def. 2.2.5)

$\Rightarrow \mathcal{K}_P(wp)(\mathfrak{I}_0)(P)\ (\mathcal{K}_P(\pi(t_1,$ wp, D-vars, $\mathfrak{K}),\ldots,\ \mathcal{K}_P(\pi(t_n,$ wp, D-vars, $\mathfrak{K}))$ (❄ and ❖)

$\Rightarrow \mathcal{K}_P \Vdash_P P(wp, \pi(t_1,$ wp, D-vars, $\mathfrak{K}),\ldots,\ \pi(t_n,$ wp, D-vars, $\mathfrak{K}) = \mathcal{F}_P$. (lemma 4.2.3, def. 3.2.4)

Second Base Case: $\mathcal{F} = \neg P(t_1,\ldots,t_n)$ is a negated atom.

Let $\mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}$

According to the translation rule for the atomic case we must show:

$\mathcal{K}_P \Vdash_P \neg P(wp, \pi(t_1,$ wp, D-vars, $\mathfrak{K}),\ldots,\ \pi(t_n,$ wp, D-vars, $\mathfrak{K})$.

With the same arguments as in the first base case we can show:

$\mathcal{K}_P\ (\pi(t,$ wp, D-vars, $\mathfrak{K})) = \mathcal{K}_M(t)$ for $t \in \{t_1,\ldots,t_n\}$.

Now we can conclude:

$\mathcal{K}_M \Vdash_M \neg P(t_1,\ldots,t_n)$ (lemma 4.2.2)

$\Rightarrow$ not $\mathfrak{I}(P)\ (\mathcal{K}_M(t_1),\ldots,\ \mathcal{K}_M(t_n))$ (def. 2.2.5)

$\Rightarrow$ not $\mathcal{K}_P(wp)(\mathfrak{I}_0)(P)\ (\mathcal{K}_P(\pi(t_1,$ wp, D-vars, $\mathfrak{K}),\ldots,\ \mathcal{K}_P(\pi(t_n,$ wp, D-vars, $\mathfrak{K}))$ (❄ and ❖)

$\Rightarrow \mathcal{K}_P \Vdash_P \neg P(wp, \pi(t_1,$ wp, D-vars, $\mathfrak{K}),\ldots,\ \pi(t_n,$ wp, D-vars, $\mathfrak{K}) = \mathcal{F}_P$ (lemma 4.2.3, def. 3.2.4)

Induction Step: Let $\mathcal{F}$ be an M-formula, but no literal.

Let $\mathcal{K} = (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}$ With lemma 4.2.2 we know $\mathcal{K}_M \Vdash_M \mathcal{F}$.

In order to show $\mathcal{K}_P \Vdash_P \mathcal{F}_P$ we must perform a case analysis according to the structure of $\mathcal{F}$.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \wedge \mathcal{F}_2,$ wp, D-vars, $\mathfrak{K}) := \pi(\mathcal{F}_1,$ wp, D-vars, $\mathfrak{K}) \wedge \pi(\mathcal{F}_2,$ wp, D-vars, $\mathfrak{K})$.

According to the induction hypothesis, we know

$\mathcal{K}_P \Vdash_P \pi(\mathcal{F}_1,$ wp, D-vars, $\mathfrak{K})$ and $\mathcal{K}_P \Vdash_P \pi(\mathcal{F}_2,$ wp, D-vars, $\mathfrak{K})$.

Thus $\mathcal{K}_P \Vdash_P \pi(\mathcal{F}_1,$ wp, D-vars, $\mathfrak{K}) \wedge \pi(\mathcal{F}_2,$ wp, D-vars, $\mathfrak{K}) = \mathcal{F}_P$. (def. 3.2.4)

Case $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \vee \mathcal{F}_2,$ wp, D-vars, $\mathfrak{K}) := \pi(\mathcal{F}_1,$ wp, D-vars, $\mathfrak{K}_1) \vee \pi(\mathcal{F}_2,$ wp, D-vars, $\mathfrak{K}_2)$ where

$\mathfrak{K}_1 := \{\mathcal{K} \in \mathfrak{K} \mid \mathcal{K}_M \Vdash_M \mathcal{F}_1\}$ and $\mathfrak{K}_2 := \{\mathcal{K} \in \mathfrak{K} \mid \mathcal{K}_M \Vdash_M \mathcal{F}_2\}$.

Because of $\mathcal{K}_M \Vdash_M \mathcal{F}$ either $\mathcal{K}_M \Vdash_M \mathcal{F}_1$ or $\mathcal{K}_M \Vdash_M \mathcal{F}_2$,

According to the induction hypothesis, we know

$\mathcal{K}_P \Vdash_P \pi(\mathcal{F}_1,$ wp, D-vars, $\mathfrak{K})$ or $\mathcal{K}_P \Vdash_P \pi(\mathcal{F}_2,$ wp, D-vars, $\mathfrak{K})$

and therefore we conclude $\mathcal{K}_P \Vdash_P \mathcal{F}_P$.

Case $\mathcal{F} = \forall x \mathcal{G}$. The translation rule for this case is:

$\pi(\forall x \mathcal{G},$ wp, D-vars, $\mathfrak{K}) = \forall x\ \pi(\mathcal{G},$ wp, D-vars + x, $\mathcal{L})$ where $\mathcal{L} := \{\mathcal{K}[x/a] \mid \mathcal{K} \in \mathfrak{K}$ and $a \in \mathbb{D}\}$.

According to the induction hypothesis, we know for every $a \in \mathbb{D}$: $\mathcal{K}[x/a]_P \Vdash_P \pi(\mathcal{G},$ wp, D-vars, $\mathfrak{K})$

and therefore we conclude $\mathcal{K}_P \Vdash_P \mathcal{F}_P$.

Case $\mathcal{F} = \Box \mathcal{G}$. The translation rule for this case is:

$\pi(\Box \mathcal{G},$ wp, D-vars, $\mathfrak{K}) := \forall u\ \pi(\mathcal{G},$ [wp . u], D-vars, $\mathcal{L})$ where

$\mathcal{L} = \{(\phi(\mathfrak{I}), \mathfrak{d}, \mathfrak{w}[u/\phi]) \mid (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{k}, \phi \in \mathfrak{I}_{\rightarrow}$ with $\phi(\mathfrak{I}) \neq \perp\}$.

$\mathcal{K}_P(prefix^*(u, \forall u\ \pi(\mathcal{G},$ [wp . u], D-vars, $\mathcal{L}))) (\mathfrak{I}_0)$

$= \mathcal{K}_P(prefix^*(u,$ [wp . u])) $(\mathfrak{I}_0)$ (lemma 4.1.3)

$= \mathcal{K}_P(wp)\ (\mathfrak{I}_0)$

$= \mathfrak{I} \neq \perp$ (lemma 4.2.3)

$\Rightarrow$ only the or case in def. 3.2.4 need be considered.

Let $\phi \in \mathfrak{I}_{\rightarrow}$ with $\phi(\mathfrak{I}) \neq \perp$.

$\Rightarrow \mathcal{K}[u/\phi]_P \Vdash_P \pi(\mathcal{G}, [wp\ .\ u], D\text{-vars}, \mathcal{L})$ (induction hypothesis)

$\Rightarrow \mathcal{K}_P \Vdash_P \mathcal{F}_P$. (def. 3.2.4)

Case $\mathcal{F} = \exists x\ \mathcal{G}$. The translation rule for this case is:

$\pi(\exists x \mathcal{G}, wp, D\text{-vars}, \mathcal{K}) := (\pi(\mathcal{G}, wp, D\text{-vars}, \mathcal{L}))[x \leftarrow f(wp, D\text{-vars})]$ where

$\mathcal{L} := \{\mathcal{K}[x/a] \mid a \in \mathbb{D}, \mathcal{K} \in \mathfrak{K}, \mathcal{K}_M[x/a] \Vdash_M \mathcal{G}\}$.

Since $\mathcal{K}_M \Vdash_M \mathcal{F}$ there is at least one $a \in \mathbb{D}$ with $\mathcal{K}_M[x/a] \Vdash_M \mathcal{G}$.

Therefore for the particular $\bar{a}$ with $\mathcal{K}_P(f(wp, D\text{-vars})) = \bar{a}$

$\mathcal{K}[x/\bar{a}]_P \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars}, \mathcal{K})$ holds according to the induction hypothesis.

Furthermore, since $\mathcal{K}[x/\bar{a}]_P(f(wp, D\text{-vars})) = \bar{a}$ ($x \notin f(wp, D\text{-vars})$

$\mathcal{K}[x/\bar{a}]_P \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars}, \mathcal{K})[x \leftarrow f(wp, D\text{-vars})] = \pi(\exists x \mathcal{G}, wp, D\text{-vars}, \mathcal{K})$

and since $x \notin \pi(\exists x \mathcal{G}, wp, D\text{-vars}, \mathcal{K})$, $\mathcal{K}_P \Vdash_P \mathcal{F}_P$.

Case $\mathcal{F} = \Diamond \mathcal{G}$. The translation rule for this case is:

$\pi(\Diamond \mathcal{G}, wp, D\text{-vars}, \mathcal{K}) := \pi(\mathcal{G}, [wp\ .\ g(D\text{-vars})], D\text{-vars}, \mathcal{L})$

$\mathcal{L} := \{(\mathfrak{I}', \mathfrak{d}, \mathfrak{w}) \mid (\mathfrak{I}, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{K}\ \mathfrak{R}(\mathfrak{I}, \mathfrak{I}'),\ (\mathfrak{I}', \mathfrak{d}, \mathfrak{w})_M \Vdash_M \mathcal{G},\ \mathcal{K}_P(g(D\text{-vars}))\ (\mathfrak{I}) = \mathfrak{I}'\}$.

The induction hypothesis is immediately applicable stating

$\mathcal{K}_P \Vdash_P \pi(\mathcal{G}, [wp\ .\ g(D\text{-vars})], D\text{-vars}, \mathcal{K}) = \mathcal{F}_P$. ■

Theorem 4.2.5 (Soundness of the Translation Algorithm)

If $\mathcal{H}$ is a satisfiable M-formula then $\Pi(\mathcal{H})$ is a satisfiable P-formula.

This is now a trivial consequence of the previous lemma. ■

Remark

The proof shows that the translation from M-logic to P-logic does not change the models of the formulae, only the way they are described. In M-logic we have the accessibility relation as a relation in the usual way whereas in P-logic the same relation is described indirectly by the domain-range relation of the world-access functions.

4.3 Completeness of the Translation

In order to demonstrate the completeness of the translation function Π we show that whenever the translated formula has a P-model then the original formula has an M-model. This proof is easier than the soundness proof because we can discard the additional information contained in the P-frame and show that the M-frame, which is the kernel of the P-frame, is a model for the original formula. To this end we show that the P-interpretations for the subformulae of the translated formula can be turned into M-interpretations for the corresponding subformulae of the original formula. Therefore we again redefine the translation functions Π and π. This time Π has as additional parameters the P-frame and a P-interpretation and π has the P-interpretations for the corresponding subformulae as an additional argument.

Definition 4.3.1 (The augmented translation functions.)

Let $\mathbf{F}_P := ((\mathbb{D}, \mathfrak{I}, \Re), \mathfrak{I}_W)$ be a P-frame and $\mathfrak{I}_P$ a P-interpretation with initial world $\mathfrak{I}_0$.

The toplevel call is now : $\Pi(\mathcal{F}, \mathbf{F}_P, \mathfrak{I}_P) := \pi(\mathcal{F}, [], (), \{\mathfrak{I}_P\})$

The recursive calls to π are:

$\pi(\mathcal{F} \wedge \mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P) \wedge \pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P)$.

$\pi(\mathcal{F} \vee \mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_{P1}) \vee \pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_{P2})$

where $\mathfrak{I}_{P1} := \{\mathfrak{I}_P \in \mathfrak{I}_P \mid \mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_1, wp, D\text{-vars})\}$

and $\mathfrak{I}_{P2} := \{\mathfrak{I}_P \in \mathfrak{I}_P \mid \mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_2, wp, D\text{-vars})\}$.

$\pi(\forall x \mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P) := \forall x\, \pi(\mathcal{F}, wp, D\text{-vars} + x, \mathfrak{F}_P)$

where $\mathfrak{F}_P := \{\mathfrak{I}_P[x/a] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ and } a \in \mathbb{D}\}$.

$\pi(\Box \mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P) := \forall u\, \pi(\mathcal{F}, [wp\,.\,u], D\text{-vars}, \mathfrak{F}_P)$

u is added as a new W-variable symbol to $\mathbb{V}_W$ in Σ_P.

$\mathfrak{F}_P := \{\mathfrak{I}_P[u/\phi] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ and } \phi \in \mathfrak{I}_{\rightarrow} \text{ with } (\mathfrak{I}_P(wp) \circ \phi)(\mathfrak{I}_0) \neq \bot\}$.

$\pi(\exists x \mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P) := (\pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{F}_P))[x \leftarrow f(wp, D\text{-vars})]$

f is added as a new D-valued function symbol of type $W \rightarrow (D^{|D\text{-vars}|} \rightarrow D)$ to $\mathbb{F}_D$ in Σ_P.

$\mathfrak{F}_P := \{\mathfrak{I}_P[x/\mathfrak{I}_P(f(wp, D\text{-vars}))] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ with } \mathfrak{I}_P(f(wp, D\text{-vars})) \neq \bot\}$.

$\pi(\Diamond \mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}, [wp\,.\,g(D\text{-vars})], D\text{-vars}, \mathfrak{I}_P)$

g is added as a new W-valued function symbol of type $D^{|D\text{-vars}|} \rightarrow (W \rightarrow W)$ to $\mathbb{F}_W$ in Σ_P.

$\pi(\pm P(t_1, \ldots, t_n), wp, D\text{-vars}, \mathfrak{I}_P)$ where P is an n-place predicate symbol

$:= \pm P(wp, \pi(t_1, wp, D\text{-vars}, \mathfrak{I}_P), \ldots, \pi(t_n, wp, D\text{-vars}, \mathfrak{I}_P))$

$\pi(f(t_1, \ldots, t_n), wp, D\text{-vars}, \mathfrak{I}_P)$ where f is an n-place function symbol

$:= f(wp, \pi(t_1, wp, D\text{-vars}, \mathfrak{I}_P), \ldots, \pi(t_n, wp, D\text{-vars}, \mathfrak{I}_P))$

$\pi(x, wp, D\text{-vars}) := x$ where x is a D-variable symbol. ∎

The next lemma states that the augmented translation functions are well defined, i.e. when the input P-frame is a P-model for the translated formula then the generated P-interpretations satisfy the corresponding translated subformulae.

Lemma 4.3.2 When $\mathfrak{F}_P$ is a P-frame satisfying the P-formula $\mathcal{H}$ in the world $\mathfrak{I}_0$ then
for each recursive call $\mathcal{F}_P := \pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathfrak{I}_P)$ during the translation $\Pi(\mathcal{H}, \mathfrak{F}_P, \mathfrak{I}_P)$ the following invariant holds as long as $\mathcal{F}$ is a formula: For every $\mathfrak{I}_P \in \mathfrak{I}_P$: $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$.

Proof: By induction on the recursion depth.

Base Case: Recursion depth = 0: This is just the initial call $\pi(\mathcal{F}, [], (), \{\mathfrak{I}_P\})$

Since $\mathfrak{I}_P$ is assumed to satisfy $\mathcal{H}$, the statement holds by definition.

Induction Step: Let the recursion depth be greater than 0.

Let $\mathcal{F}_P := \pi(\mathcal{F}, \text{wp}, \text{D-vars}, \mathfrak{I}_P)$ be the recursive call to π.

The induction hypothesis states: For every $\mathfrak{I}_P \in \mathfrak{I}_P$: $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$.

In order to show that the statement holds for the next recursion step we perform a case analysis according to the structure of $\mathcal{F}$ and analyze the corresponding translation rule.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \wedge \mathcal{F}_2, \text{wp}, \text{D-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}_1, \text{wp}, \text{D-vars}, \mathfrak{I}_P) \wedge \pi(\mathcal{F}_2, \text{wp}, \text{D-vars}, \mathfrak{I}_P)$.

The induction hypothesis implies:

For every $\mathfrak{I}_P \in \mathfrak{I}_P$: $\mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_1, \text{wp}, \text{D-vars}, \mathfrak{I}_P)$ and $\mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_2, \text{wp}, \text{D-vars}, \mathfrak{I}_P)$ (def. 3.2.4)

This is just what we need.

Case $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \vee \mathcal{F}_2, \text{wp}, \text{D-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}_1, \text{wp}, \text{D-vars}, \mathfrak{I}_{P1}) \vee \pi(\mathcal{F}_2, \text{wp}, \text{D-vars}, \mathfrak{I}_{P2})$

where $\mathfrak{I}_{P1} := \{\mathfrak{I}_P \in \mathfrak{I}_P \mid \mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_1, \text{wp}, \text{D-vars})\}$

and $\mathfrak{I}_{P2} := \{\mathfrak{I}_P \in \mathfrak{I}_P \mid \mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_2, \text{wp}, \text{D-vars})\}$.

Thus with "$\mathfrak{I}_P \Vdash_P \pi(\mathcal{F}\ldots$" the condition is explicitly enforced in the definition of π.

Case $\mathcal{F} = \forall x \mathcal{G}$. The translation rule for this case is:

$\pi(\forall x \mathcal{G}, \text{wp}, \text{D-vars}, \mathfrak{I}_P) := \forall x\, \pi(\mathcal{G}, \text{wp}, \text{D-vars} + x, \mathfrak{F}_P)$

where $\mathfrak{F}_P := \{\mathfrak{I}_P[x/a] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ and } a \in \mathbb{D}\}$

The induction hypothesis implies:

For every $\mathfrak{I}_P \in \mathfrak{I}_P$, $\mathfrak{I}_P[x/a] \Vdash_P \pi(\mathcal{G}, \text{wp}, \text{D-vars} + x, \mathfrak{F}_P)$, (def. 3.2.4)

i.e. the condition holds for $\mathfrak{F}_P$.

Case $\mathcal{F} = \Box\mathcal{G}$. The translation rule for this case is:

$\pi(\Box\mathcal{G}, \text{wp}, \text{D-vars}, \mathfrak{I}_P) := \forall u\, \pi(\mathcal{G}, [\text{wp} \,.\, u], \text{D-vars}, \mathfrak{F}_P)$

where $\mathfrak{F}_P := \{\mathfrak{I}_P[u/\phi] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ and } \phi \in \mathfrak{I}_{\rightarrow} \text{ with } (\mathfrak{I}_P(\text{wp}) \circ \phi)(\mathfrak{I}_0) \neq \perp\}$.

Let $\mathfrak{I}_P[u/\phi] \in \mathfrak{F}_P$.

$\mathfrak{I}_P(\text{prefix}^*(u, \pi(\mathcal{G}, [\text{wp} \,.\, u], \text{D-vars}, \mathfrak{F}_P)))\ (\mathfrak{I}_0)$

$= \mathfrak{I}_P(\text{prefix}^*(u, [\text{wp} \,.\, u]))\ (\mathfrak{I}_0)$ (lemma 4.1.3)

$= \mathfrak{I}_P(\text{wp})\ (\mathfrak{I}_0)$

$\neq \perp$ (lemma 4.1.4)

$\Rightarrow \mathfrak{I}_P[u/\phi] \Vdash_P \pi(\mathcal{G}, [\text{wp} \,.\, u], \text{D-vars}, \mathfrak{F}_P)$ (induction hypothesis and def. 3.2.4, or-case)

Case $\mathcal{F} = \exists x\, \mathcal{G}$. The translation rule for this case is:

$\pi(\exists x \mathcal{G}, \text{wp}, \text{D-vars}, \mathfrak{I}_P) := (\pi(\mathcal{G}, \text{wp}, \text{D-vars}, \mathfrak{F}_P))[x \leftarrow f(\text{wp}, \text{D-vars})]$

$\mathfrak{F}_P := \{\mathfrak{I}_P[x/\mathfrak{I}_P(f(\text{wp}, \text{D-vars}))] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ with } \mathfrak{I}_P(f(\text{wp}, \text{D-vars})) \neq \perp\}$.

Let $\mathfrak{I}_P[x/\mathfrak{I}_P(f(\text{wp}, \text{D-vars}))] \in \mathfrak{F}_P$.

Since $\mathfrak{I}_P(f(\text{wp}, \text{D-vars})]) = \mathfrak{I}_P[x/f(\text{wp}, \text{D-vars}))](x)$ by induction on the structure of P-terms it can be shown that $\mathfrak{I}_P$ evaluates the terms occurring in $\pi(\mathcal{G}, \text{wp}, \text{D-vars}, \mathfrak{F}_P)[x \leftarrow f(\text{wp}, \text{D-vars})]$ to the

same values as $\mathfrak{I}_P[x/\mathfrak{I}_P(f(wp, D\text{-vars}))]$ does with the corresponding terms in $\pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{F}_P)$.
Since $\mathfrak{I}_P \Vdash_P (\pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{F}_P))[x \leftarrow f(wp, D\text{-vars})]$ (induction hypothesis)
we can conclude $\mathfrak{I}_P[x/\mathfrak{I}_P(f(wp, D\text{-vars}))] \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{F}_P)$.

Case $\mathcal{F} = \Diamond\, \mathcal{G}$. The translation rule for this case is:

$\pi(\Diamond\mathcal{G}, wp, D\text{-vars}, \mathfrak{F}_P) := \pi(\mathcal{G}, [wp\ .\ g(D\text{-vars})], D\text{-vars}, \mathfrak{F}_P)$

Thus, the induction hypothesis is immediately applicable. ■

Now we are ready to prove the main part of the completeness theorem, namely that the P-interpretations for the subformulae of the translated formula can be turned into M-interpretations for the corresponding subformulae of the original formula.

Lemma 4.3.3 When $\mathfrak{F}_P := (\mathfrak{F}_M, \mathfrak{I}_W)$ is a P-frame satisfying the P-formula $\mathcal{H}$ in the world $\mathfrak{I}_0$ then for each recursive call $\pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P)$ during the translation $\Pi(\mathcal{H}, \mathfrak{F}_P)$ the following invariant holds as long as $\mathcal{F}$ is a formula:

For every $\mathfrak{I}_P := (\mathfrak{F}_P, \mathfrak{I}_0, \mathfrak{d}, \mathfrak{w}) \in \mathfrak{I}_P$: $\mathfrak{I}_M := (\mathfrak{F}_M, \mathfrak{I}_P(wp)(\mathfrak{I}_0), \mathfrak{d}) \Vdash_M \mathcal{F}$.

($\mathfrak{I}_M$ denotes the M-interpretation in the world $\mathfrak{I}_P(wp)(\mathfrak{I}_0)$ that corresponds to the P-interpretation $\mathfrak{I}_P$.)

Proof: Let $\mathcal{F}_P := \pi(\mathcal{F}, wp, D\text{-vars}, \mathfrak{I}_P)$ be the actual call to π, let $\mathfrak{I}_P \in \mathfrak{I}_P$ and
let $\mathfrak{I}_M$ denote the corresponding M-interpretation in the world $\mathfrak{I} := \mathfrak{I}_P(wp)(\mathfrak{I}_0) \neq \perp$. (lemma 4.1.4)
With lemma 4.3.2 we know $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$.
We perform an induction on the structure of M-formulae.

First Base Case: $\mathcal{F} = P(t_1,\ldots,t_n)$ is an atom.

Since the wp-argument of π remains unchanged in the recursive descent of π into terms, and since $\mathfrak{I}_P(wp)(\mathfrak{I}_0)$ is the actual world of $\mathfrak{I}_M$ we can show by induction on the structure of M-logic terms:
$\mathfrak{I}_M(t_i) = \mathfrak{I}_P(\pi(t_i, wp, D\text{-vars}, \mathfrak{I}_P))$ for i = 1,...,n.
$\mathfrak{I}_P \Vdash_P \mathcal{F}_P$ implies $\mathfrak{I}_P(t_i) \neq \perp$ for i = 1,...,n and
$\mathfrak{I}_P(wp)(\mathfrak{I}_0)(P)\ (\mathfrak{I}_M(t_1),\ldots,\mathfrak{I}_M(t_n)) = \mathfrak{I}(P)\ (\mathfrak{I}_M(t_1),\ldots,\mathfrak{I}_M(t_n))$ holds
Thus, $\mathfrak{I}_M \Vdash_M \mathcal{F}$.

Second Base Case: $\mathcal{F} = \neg P(t_1,\ldots,t_n)$ is an atom.

The proof is analogous to the first base case.

Induction Step: Let $\mathcal{F}$ be an M-formula which is no M-literal.

We perform a case analysis according to the structure of $\mathcal{F}$.

Case $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \wedge \mathcal{F}_2, wp, D\text{-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}_1, wp, D\text{-vars}, \mathfrak{I}_P) \wedge \pi(\mathcal{F}_2, wp, D\text{-vars}, \mathfrak{I}_P)$.

The statement follows immediately from the induction hypothesis and def. 2.2.5.

Case $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. The translation rule is:

$\pi(\mathcal{F}_1 \vee \mathcal{F}_2, wp, D\text{-vars}, \mathfrak{I}_P) := \pi(\mathcal{F}_1, wp, D\text{-vars}, \mathfrak{I}_{P1}) \vee \pi(\mathcal{F}_2, wp, D\text{-vars}, \mathfrak{I}_{P2})$
where $\mathfrak{I}_{P1} := \{\mathfrak{I}_P \in \mathfrak{I}_P \mid \mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_1, wp, D\text{-vars})\}$
and $\mathfrak{I}_{P2} := \{\mathfrak{I}_P \in \mathfrak{I}_P \mid \mathfrak{I}_P \Vdash_P \pi(\mathcal{F}_2, wp, D\text{-vars})\}$.
Since $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$, either $\mathfrak{I}_P \in \mathfrak{I}_{P1}$ or $\mathfrak{I}_P \in \mathfrak{I}_{P2}$.
We can apply the induction hypothesis finding either $\mathfrak{I}_M \Vdash_M \mathcal{F}_1$ or $\mathfrak{I}_M \Vdash_M \mathcal{F}_2$, thus $\mathfrak{I}_M \Vdash_M \mathcal{F}$.

Case $\mathcal{F} = \forall x \mathcal{G}$. The translation rule for this case is:

$\pi(\forall x \mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P) := \forall x\ \pi(\mathcal{G}, wp, D\text{-vars} + x, \mathfrak{F}_P)$
where $\mathfrak{F}_P := \{\mathfrak{I}_P[x/a] \mid \mathfrak{I}_P \in \mathfrak{I}_P \text{ and } a \in \mathbb{D}\}$.
Since $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$, for every $a \in \mathbb{D}$: $\mathfrak{I}_P[x/a] \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars} + x, \mathfrak{F}_P)$.
We can apply the induction hypothesis finding $\mathfrak{I}_M[x/a] \Vdash_M \mathcal{G}$, thus $\mathfrak{I}_M \Vdash_M \mathcal{F}$.

Case $\mathcal{F} = \Box\mathcal{G}$. The translation rule for this case is:

$\pi(\Box\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P) := \forall u\ \pi(\mathcal{G}, [wp\ .\ u], D\text{-vars}, \mathfrak{I}_P)$

where $\mathfrak{I}_P := \{\mathfrak{I}_P[u/\phi] \mid \mathfrak{I}_P \in \mathfrak{I}_P$ and $\phi \in \mathfrak{I}_{\rightarrow}$ with $(\mathfrak{I}_P(wp)\circ\phi)(\mathfrak{I}_0) \neq \perp\}$.

Since $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$ and since $\mathfrak{I} \neq \perp$, for every $\mathfrak{I}'$ with $\Re(\mathfrak{I},\mathfrak{I}')$ there is an $\phi \in \mathfrak{I}_{\rightarrow}$ with $\phi(\mathfrak{I}) = \mathfrak{I}'$.

We can apply the induction hypothesis finding $(\mathfrak{F}_M, \mathfrak{I}', \mathfrak{A}) \Vdash_M \mathcal{G}$, thus $\mathfrak{I}_M \Vdash_M \mathcal{F}$.

Case $\mathcal{F} = \exists x\ \mathcal{G}$. The translation rule for this case is:

$\pi(\exists x\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P) := (\pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P))[x \leftarrow f(wp, D\text{-vars})]$

$\mathfrak{I}_P := \{\mathfrak{I}_P[x/\mathfrak{I}_P(f(wp, D\text{-vars}))] \mid \mathfrak{I}_P \in \mathfrak{I}_P$ with $\mathfrak{I}_P(f(wp, D\text{-vars})) \neq \perp\}$.

Since $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$ and $\mathfrak{I}_P(f(wp,D\text{-vars})) = \mathfrak{I}_P[x/\mathfrak{I}_P(f(wp, D\text{-vars}))](x)$ we have

$\mathfrak{I}_P[x/\mathfrak{I}_P(f(wp, D\text{-vars}))] \Vdash_P \pi(\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P)$.

We can apply the induction hypothesis finding $\mathfrak{I}_M[x/\mathfrak{I}_P(f(wp, D\text{-vars}))] \Vdash_M \mathcal{G}$, thus $\mathfrak{I}_M \Vdash_M \mathcal{F}$.

Case $\mathcal{F} = \Diamond\mathcal{G}$. The translation rule for this case is:

$\pi(\Diamond\mathcal{G}, wp, D\text{-vars}, \mathfrak{I}_P) := \pi(\mathcal{G}, [wp\ .\ g(D\text{-vars})], D\text{-vars}, \mathfrak{I}_P)$.

Since $\mathfrak{I}_P \Vdash_P \mathcal{F}_P$, according to lemma 4.1.4, $\mathfrak{I}' := \mathfrak{I}_P(g(D\text{-vars}))(\mathfrak{I}) \neq \perp$.

According to def. 3.2.1, $\Re(\mathfrak{I}, \mathfrak{I}')$ holds.

We can apply the induction hypothesis finding $(\mathfrak{F}_M, \mathfrak{I}', \mathfrak{A}) \Vdash_M \mathcal{G}$, thus $\mathfrak{I}_M \Vdash_M \mathcal{F}$. ■

Theorem 4.3.3 (Completeness of the Translation Algorithm)

If $\Pi(\mathcal{H})$ is a satisfiable P-formula then $\mathcal{H}$ is a satisfiable M-formula.

This is now an obvious consequence of the previous lemma. ■

Combining theorem 4.3.3 and theorem 4.2.5 we obtain the main result of this section:

Corollary 4.3.4 An M-formula $\mathcal{F}$ is unsatisfiable if and only if $\Pi(\mathcal{F})$ is unsatisfiable. ■

Chapter Five

Tools for P-Logic

5.1. Conjunctive Normal Form

A formula in conjunctive normal form is a conjunction of a disjunction of literals, where all variables are taken to be universally quantified. Each disjunction is usually called a clause and is written as a set. Since the P-logic syntax contains the logical connectives $\wedge$ and $\vee$ and the universal quantifier, but no existential quantifier and no modal operators, a transformation of an arbitrary formula to a set of clauses is simpler than in first-order predicate logic. Therefore only the critical aspects are dealt with in the following.

Transformation of P-Formulae to Clauses:

In order to generate the conjunctive normal form of a P-formula the distributivity laws for $\wedge$ and $\vee$, laws for moving universal quantifiers over conjunctions and in certain cases over disjunctions and a law for renaming universally quantified variables are necessary. Each transformation rule must be verified by proving that a P-interpretation satisfies the original formula if and only if it satisfies the transformed formula. As this is fairly straightforward, we only demonstrate it for the rules that move quantifiers for W-variables and consider only those cases, which actually occur.

Lemma 5.1.1 (Moving Quantifiers over Conjunctions)
Let $\forall u\ (\mathcal{F} \wedge \mathcal{G})$ be an M-adjusted P-formula (i.e. the prefix of u is unique) where the W-variable u occurs both in $\mathcal{F}$ and $\mathcal{G}$. Let $\Im_P$ be a P-interpretation for $\forall u\ (\mathcal{F} \wedge \mathcal{G})$.
Then $\Im_P \Vdash_P \forall u\ (\mathcal{F} \wedge \mathcal{G})$ iff $\Im_P \Vdash_P \forall u\ \mathcal{F} \wedge \forall u\ \mathcal{G}$.
Proof: Let $\Im_0$ be the initial world of $\Im_P$.

$\Im_P \Vdash_P \forall u\ (\mathcal{F} \wedge \mathcal{G})$

iff for wp = prefix*(u, $\mathcal{F} \wedge \mathcal{G}$) (def. 3.2.4)

either $\Im_P(wp)(\Im_0) = \perp$ and $\Im_P \Vdash_P \mathcal{F} \wedge \mathcal{G}$

or $\Im_P(wp)(\Im_0) \neq \perp$ and for every $\phi \in \Im_{\rightarrow}$ with $(\Im_P(wp) \circ \phi)(\Im_0) \neq \perp$: $\Im_P[u/\phi] \Vdash_P \mathcal{F} \wedge \mathcal{G}$

iff for wp = prefix*(u, $\mathcal{F} \wedge \mathcal{G}$) = prefix*(u, $\mathcal{F}$) = prefix*(u, $\mathcal{G}$)

either $\Im_P(wp)(\Im_0) = \perp$ and $\Im_P \Vdash_P \mathcal{F}$ and $\Im_P \Vdash_P \mathcal{G}$

or $\Im_P(wp)(\Im_0) \neq \perp$ and for every $\phi \in \Im_{\rightarrow}$ with $(\Im_P(wp) \circ \phi)(\Im_0) \neq \perp$:
$\Im_P[u/\phi] \Vdash_P \mathcal{F}$ and $\Im_P[u/\phi] \Vdash_P \mathcal{G}$

iff for wp = prefix*(u, $\mathcal{F}$)

either $\Im_P(wp)(\Im_0) = \perp$ and $\Im_P \Vdash_P \mathcal{F}$

or $\Im_P(wp)(\Im_0) \neq \perp$ and for every $\phi \in \Im_{\rightarrow}$ with $(\Im_P(wp) \circ \phi)(\Im_0) \neq \perp$: $\Im_P[u/\phi] \Vdash_P \mathcal{F}$

and for wp = prefix*(u, $\mathcal{G}$)

either $\Im_P(wp)(\Im_0) = \perp$ and $\Im_P \Vdash_P \mathcal{G}$

or $\Im_P(wp)(\Im_0) \neq \perp$ and for every $\phi \in \Im_{\rightarrow}$ with $(\Im_P(wp) \circ \phi)(\Im_0) \neq \perp$: $\Im_P[u/\phi] \Vdash_P \mathcal{G}$

iff $\Im_P \Vdash_P \forall u\ \mathcal{F}$ and $\Im_P \Vdash_P \forall u\ \mathcal{G}$

iff $\Im_P \Vdash_P \forall u\ \mathcal{F} \wedge \forall u\ \mathcal{G}$. ■

Lemma 5.1.2 (Moving Quantifiers over Disjunctions)

Let $\mathcal{F} \vee \forall u\ \mathcal{G}$ be an M-adjusted P-formula where the W-variable u does not occur in $\mathcal{F}$.

Let $\Im_p$ be a P-interpretation for $\mathcal{F} \vee \forall u\ \mathcal{G}$. Then $\Im_p \Vdash_p \mathcal{F} \vee \forall u\ \mathcal{G}$ iff $\Im_p \Vdash_p \forall u\ (\mathcal{F} \vee \mathcal{G})$.

Proof: Let wp := prefix*(u, $\mathcal{G}$) = prefix*(u, $\mathcal{F} \vee \forall u\ \mathcal{G}$). Let $\Im_0$ be the initial world of $\Im_p$.

"$\Rightarrow$" Let $\Im_p \Vdash_p \mathcal{F} \vee \forall u\ \mathcal{G}$.

Case 1: $\Im_p \Vdash_p \mathcal{F}$, thus $\Im_p \Vdash_p \mathcal{F} \vee \mathcal{G}$. (def. 3.2.4)

If $\Im_p(wp)(\Im_0) = \bot$ then $\Im_p \Vdash_p \mathcal{F} \vee \mathcal{G}$ and therefore $\Im_p \Vdash_p \forall u\ (\mathcal{F} \vee \mathcal{G})$.

If $\Im_p(wp)(\Im_0) \neq \bot$ then $\Im_p[u/\phi] \Vdash_p \mathcal{F}$ for the corresponding world-access functions, thus $\Im_p[u/\phi] \Vdash_p \mathcal{F} \vee \mathcal{G}$ and therefore $\Im_p \Vdash_p \forall u\ (\mathcal{F} \vee \mathcal{G})$.

Case 2: $\Im_p \Vdash_p \forall u\ \mathcal{G}$

If $\Im_p(wp)(\Im_0) = \bot$ then $\Im_p \Vdash_p \mathcal{G}$, thus $\Im_p \Vdash_p \mathcal{F} \vee \mathcal{G}$ and therefore $\Im_p \Vdash_p \forall u\ (\mathcal{F} \vee \mathcal{G})$.

If $\Im_p(wp)(\Im_0) \neq \bot$ then $\Im_p[u/\phi] \Vdash_p \mathcal{F}$ for the corresponding world-access functions, thus $\Im_p[u/\phi] \Vdash_p \mathcal{F} \vee \mathcal{G}$ and therefore $\Im_p \Vdash_p \forall u\ (\mathcal{F} \vee \mathcal{G})$.

"$\Leftarrow$" Let $\Im_p \Vdash_p \forall u\ (\mathcal{F} \vee \mathcal{G})$.

Case 1: $\Im_p(wp)(\Im_0) = \bot$.

$\Rightarrow \Im_p \Vdash_p \mathcal{F} \vee \mathcal{G}$

$\Rightarrow \Im_p \Vdash_p \mathcal{F}$ or $\Im_p \Vdash_p \mathcal{G}$

$\Rightarrow \Im_p \Vdash_p \mathcal{F}$ or $\Im_p \Vdash_p \forall u\ \mathcal{G}$.

$\Rightarrow \Im_p \Vdash_p \mathcal{F} \vee \forall u\ \mathcal{G}$.

Case 2: $\Im_p(wp)(\Im_0) \neq \bot$.

$\Rightarrow$ For the corresponding world-access functions ϕ: $\Im_p[u/\phi] \Vdash_p \mathcal{F} \vee \mathcal{G}$. (def. 3.2.4)

$\Rightarrow$ For the corresponding world-access functions ϕ: $\Im_p[u/\phi] \Vdash_p \mathcal{F}$ or $\Im_p[u/\phi] \Vdash_p \mathcal{G}$.

$\Rightarrow \Im_p \Vdash_p \mathcal{F}$ or for the corresponding world-access functions ϕ: $\Im_p[u/\phi] \Vdash_p \mathcal{G}$.

$\Rightarrow \Im_p \Vdash_p \mathcal{F} \vee \forall u\ \mathcal{G}$. ■

Example for a transformation to conjunctive normal form.

M-formula: $\Box\exists x\ Px \vee \Diamond (Qx \wedge \Box Rx)$

$\uparrow\ \uparrow \quad \uparrow \quad \uparrow$

w a g v

$\rightarrow$ P-formula: $\forall w\ P([w], a[w]) \vee (Q([wg], a[w]) \wedge \forall v\ R([wgv], a[w]))$

$\rightarrow$ Clauses: $\forall w\ \ P([w], a[w]) \vee Q([wg], a[w])$

$\forall w'\ P([w'], a[w']) \vee R([w'gv], a[w'])$. ■

Theorem 5.1.3 (Soundness and Completeness of the Transformation into Clauses)

a) An M-adjusted P-formula $\mathcal{F}$ is satisfiable iff the corresponding conjunctive normal form is satisfiable.

b) The conjunctive normal form is M-adjusted.

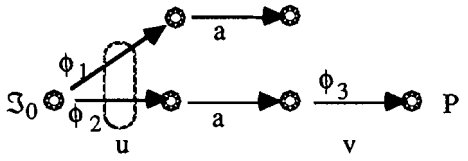
Proof: a) This is a consequence of the two previous lemmata and the corresponding lemmata for the variable renaming rule and the distributivity laws (which are obvious.)

b) This is a consequence of three facts:

1. The transformation into conjunctive normal form does not change the termstructure.
2. It is not necessary to change the ordering of quantifiers.
3. Clauses are completely variable disjoint. ■

The Semantics of Clauses Without Quantifier Prefix

The semantics of clauses without a quantifier prefix in first-order predicate logic is: A clause C with free variables $\{x_1,\ldots,x_n\}$ is true in an interpretation $\Im$ iff $\Im[x_1/c_1,\ldots,x_n/c_n] \Vdash C$ for every combination of variable assignments x_i/c_i. The choice of a particular variable assignment is independent of the other assignments. This nice property which allows to eliminate the quantifier prefix, holds also in P-logic when the accessibility relation is serial. Unfortunately it does not hold in non-serial interpretations as the following example demonstrates:
Consider the M-formula $\Box\Diamond\Box P$ and the corresponding P-logic version $\forall u,v\ P[uav]$ which is satisfied by the following interpretation:



Only the combination $\Im_P[u/\phi_2, v/\phi_3]$ satisfies the literal, whereas $\Im_P[u/\phi_1, v/\phi_3]([uav])(\Im_0) = \bot$. In fact this combination would not be considered during the recursive descent of $\Vdash_P$ because as soon as $\Im_P[u/\phi_1]$ has been generated for the quantifier $\forall u$, there is no further world accessible from $\Im_P[u/\phi_1]([ua])(\Im_0)$. Thus $\Im_P[u/\phi_1] \Vdash_P \forall v\ P[uav]$ regardless of the structure of the literal.

In order to eliminate the quantifier prefix in the non-serial case as well and to "flatten" the definition of the satisfiability relation $\Vdash_P$ for clauses, we must consider only those P-interpretations $\Im_P' := \Im_P[x_1/c_1,\ldots,x_n/c_n]$ which are "continuing", i.e. which have the property that for every W-variable x_i with $wp = prefix^*(x_i, C)$: $\Im_P'(wp) \neq \bot \Rightarrow \Im_P'([wp.x_i]) \neq \bot$. The first P-interpretation in the above example has this property, the second one has not. This motivates the following definition:

Definition 5.1.4 (Continuing P-Interpretations)
A P-interpretation $\Im_P$ with initial world $\Im_0$ is called *$\mathcal{F}$-continuing* for an M-adjusted P-formula $\mathcal{F}$ iff for every W-variable u in $\mathcal{F}$ with $wp = prefix^*(u, \mathcal{F})$: $\Im_P(wp)(\Im_0) \neq \bot$ implies $\Im_P([wp.u])(\Im_0) \neq \bot$. ■

Since P-interpretations are total functions on terms when the accessibility relation is serial, all P-interpretations are continuing in this case.

Now a correspondence between a P-model $\Im_P$ for a fully quantified clause $\forall v_1,\ldots,v_n\ C'$ and C'-continuing P-models $\Im_P[v_1/c_1,\ldots,v_n/c_n]$ for C' can be shown. It will be used frequently in the soundness proof for the resolution rule in chapter 7.

Theorem 5.1.5 (The Semantics of Clauses)

Let $C = \forall v_1 \ldots v_n\ C'$ be a fully quantified clause and let $\Im_P$ be a P-interpretation for C.

$\Im_P \Vdash_P C$ iff for every C'-continuing P-interpretation $\Im_P' := \Im_P[v_1/c_1,\ldots,v_n/c_n]$: $\Im_P' \Vdash_P C'$.

Proof: Let $\Im_0$ be the initial world of $\Im_P$.

"$\Rightarrow$" Let $\Im_P \Vdash_P C$ and let $\Im_{Pn} := \Im_P[v_1/c_1,\ldots,v_n/c_n]$ be a C'-continuing P-interpretation.

In order to show $\Im_{Pn} \Vdash_P C'$ we follow per induction the recursive descent of $\Vdash_P$. Thus we reconstruct $\Im_{Pn}$ in such a way that at each level i for $\Im_{Pi} := \Im_P[v_1/c_1,\ldots,v_i/c_i]$:

$\Im_{Pi} \Vdash_P \forall v_{i+1} \ldots v_n\ C'$ is ensured and therefore finally $\Im_P[v_1/c_1,\ldots,v_n/c_n] \Vdash_P C'$ holds.

Base Case: $\Im_P \Vdash_P C$ is the precondition.

Induction Step: Let $\Im_{Pi} := \Im_P[v_1/c_1,\ldots,v_i/c_i]$ and $\Im_{Pi} \Vdash_P \forall v_{i+1},\ldots,v_n\ C'$ (*) (induction hypothesis)

Case 1: v_{i+1} is a D-variable.

Because of (*) we know that $\Im_{Pi}[v_{i+1}/c_{i+1}] =: \Im_{Pi+1} = \Im_P[v_1/c_1,\ldots,v_{i+1}/c_{i+1}] \Vdash_P \forall v_{i+2},\ldots,v_n\ C'$

Case 2: v_{i+1} is a W-variable with wp = prefix*(v_{i+1}, C')

Case 2.1: $\Im_{Pi}(wp)(\Im_0) = \perp$.

$\Rightarrow \Im_{Pi} \Vdash_P \forall v_{i+2},\ldots,v_n\ C'$ and the satisfiability does not depend on the variable assignment of v_{i+1}.

$\Rightarrow \Im_{Pi}[v_{i+1}/c_{i+1}] =: \Im_{Pi+1} \Vdash_P \forall v_{i+2},\ldots,v_n\ C'$

Case 2.2: $\Im_{Pi}(wp)(\Im_0) \neq \perp$.

Since $\Im_{Pi}(wp) = \Im_{Pn}(wp)$ and $\Im_{Pn}([w.v_{i+1}])(\Im_0) \neq \perp$ ($\Im_{Pn}$ is C'-continuing) there is a world-access function c_{i+1} with $(\Im_{Pi}(wp) \circ c_{i+1})(\Im_0) \neq \perp$. Therefore and because of (*) we know again $\Im_{Pi}[v_{i+1}/c_{i+1}] =: \Im_{Pi+1} \Vdash_P \forall v_{i+2},\ldots,v_n\ C'$.

"$\Leftarrow$" Assume for every C'-continuing P-interpretation $\Im_{Pn} := \Im_P[v_1/c_1,\ldots,v_n/c_n]$: $\Im_{Pn} \Vdash_P C'$ (*)

In a first step we follow the recursive descent of $\Vdash_P$ when applied to the quantifier prefix of C and show at each level with $\Im_{Pi}=\Im_P[v_1/c_1,\ldots,v_i/c_i]$:

For every W-variable $v_j \in \{v_1,\ldots,v_i\}$ and wp := prefix*(v_j,C'):

if $\Im_{Pi}(wp)(\Im_0) \neq \perp$ then $\Im_{Pi}([wp.v_j])(\Im_0) \neq \perp$.

This is an immediate consequence of the or case in the definition of $\Vdash_P$ (def. 3.2.4), therefore the proof is omitted.

The result of this induction is that the literal level of the recursion is reached with C'-continuing P-interpretations $\Im_{Pn} := \Im_P[v_1/c_1,\ldots,v_n/c_n]$ and the assumption (*) states $\Im_{Pn} \Vdash_P C'$.

This is the base case of a second induction, this time "bottom up" proving that at each level either

$\Im_{Pi} := \Im_P[v_1/c_1,\ldots,v_i/c_i] \Vdash_P \forall v_{i+2} \ldots v_n C'$ when $\Im_{Pi}(\text{prefix*}(v_{i+1}, C'))(\Im_0) = \perp$, or

for every world access function c_{i+1} with $(\Im_{Pi}(wp) \circ c_{i+1})(\Im_0) \neq \perp$: $\Im_{Pi}[v_{i+1}/c_{i+1}] \Vdash_P \forall v_{i+2} \ldots v_n\ C'$

and therefore $\Im_{Pi} \Vdash_P \forall v_{i+1} \ldots v_n\ C'$.

This second induction gives the desired result: $\Im_P \Vdash_P C$. ■

5.2 Substitutions

One of our important notions is that of a substitution as a mapping from terms to terms. Substitutions must be constructed with respect to two requirements:

1. The application of a substitution to a term must produce again a wellformed term.

In our case this means that D-variables must be mapped to D-terms and W-variables must be mapped either to W-terms or to world-paths. In the latter case the world-path that replaces the W-variable u must be spliced into the world-path containing u such that the result is again a world-path.

2. The instance of a true formula should again be true.

In our case this means that a W-variable, which is interpreted in P-logic as a world-access function, can only be replaced by a world-path that can also be interpreted as a world-access function, i.e. a function that maps worlds to accessible worlds. The syntactic restrictions to substitution components for W-variables which guarantee this property depend on the properties of the accessibility relation $\mathfrak{R}$: In case $\mathfrak{R}$ is transitive, a W-variable can be replaced by every non empty world-path because in transitive relations every world which can be accessed via several other worlds can also be accessed in one step. In all other cases a W-variable can be replaced by one W-term. In case $\mathfrak{R}$ is reflexive, also the empty world-path - which is interpreted as the identity mapping - is admissible.

To fix these conditions we introduce the notion of an $\mathfrak{R}$-admissible world-path.

Definition 5.2.1 ($\mathfrak{R}$-Admissible World-Paths)

Depending on the properties of the accessibility relation $\mathfrak{R}$, a world-path wp is called $\mathfrak{R}$-admissible if it satisfies the following properties:

In case $\mathfrak{R}$ is transitive:	the length of wp may be greater than 1.
In case $\mathfrak{R}$ is reflexive:	wp may be empty.
In the other cases wp consists of exactly one W-term.	
In case $\mathfrak{R}$ is symmetric:	wp may consist of an "inverse" W-term $[g^{-1}(s_1,\ldots,s_k)]$ where g^{-1} is the associated inverse symbol for some symbol g. (c.f. def. 3.1.1). ■

Examples and counterexamples for $\mathfrak{R}$-admissible world-paths

properties of $\mathfrak{R}$	$\mathfrak{R}$-admissible	non-$\mathfrak{R}$-admissible
no special properties	[a]	[], [a b], $[a^{-1}]$
reflexive	[], [a]	[a b], $[a^{-1}]$
symmetric	[a] $[a^{-1}]$	[], [a b]
reflexive and symmetric	[], [a], $[a^{-1}]$	[a b]
transitive	[a], [a b]	[], $[a^{-1}]$
transitive and reflexive	[], [a], [a b]	$[a^{-1}]$

Lemma 5.2.2 Given a P-interpretation with an accessibility relation $\mathfrak{R}$, an $\mathfrak{R}$-admissible world-path is interpreted as a world-access function.

Proof: Obvious. ■

Definition 5.2.3: **(Substitutions)**

- A *substitution* σ is a sort preserving mapping from D-variables to D-terms and W-variables to $\mathfrak{R}$-admissible world-paths and can be represented as a finite set $\{x_1 \mapsto t_1,\ldots, x_n \mapsto t_n\}$ of variable term pairs. To emphasize that a particular substitution actually has the property to map W-variables only to $\mathfrak{R}$-admissible world-paths, we sometimes call it an *$\mathfrak{R}$-admissible substitution.*
- Substitutions can be turned into mappings from terms to terms, literals to literals and clauses to clauses using the inductive definition of terms and such that the following homomorphic equation for all function and predicate symbols F and terms w and t_i hold:

  $\sigma(F(wp,t_1,\ldots,t_n)) = F(\sigma(wp), \sigma(t_1),\ldots,\sigma(t_n))$.

  We sometimes omit parentheses and write σt instead of $\sigma(t)$.
- The composition $\sigma\circ\tau$ of two substitutions σ and τ is the usual functional composition.
  We shall omit the $\circ$-sign and write $\sigma\tau$ instead of $\sigma\circ\tau$.
  The set of substitutions with the composition $\circ$ is a monoid with identity $\emptyset$.
  It is easy to verify that the composition of two substitutions maps W-variables to $\mathfrak{R}$-admissible world-paths.
- $\sigma_{|V}$ is the restriction of a substitution σ to a set V of variables, i.e.

  $\sigma_{|V}(x) = \sigma x$ if $x \in V$

  $\sigma_{|V}(x) = x$ if $x \notin V$.
- The *domain* of a substitution σ is the set of variables which are moved by σ, i.e.

  $DOM(\sigma) := \{x \mid \sigma x \neq x\}$.
- The *codomain* of a substitution σ is $COD(\sigma) := \{\sigma x \mid x \in DOM(\sigma)\}$
- The *variables introduced by* σ are: $VCOD(\sigma) := Vars(COD(\sigma))$.
- A *ground substitution* is a substitution with $VCOD(\sigma) = \emptyset$.
- Let t be a term, atom, literal or clause and let σ be a substitution.
  σt is called an *instance* of t. σt is called a *ground instance* of t if it contains no variables.
- There is an ordering relation $\leq_{\mathfrak{R}}[V]$ on substitutions:
  For two substitutions σ and τ and a set V of variables:
  $\sigma \leq_{\mathfrak{R}} \tau\ [V]$ iff there exists an $\mathfrak{R}$-admissible substitution ξ such that $\forall x \in V$: $\sigma(x) = \xi\tau(x)$.
- A substitution σ is *idempotent* iff $\sigma\sigma = \sigma$. ■

An application of a substitution $\sigma = \{x_1 \mapsto t_1,\ldots, x_n \mapsto t_n\}$ to a term t has the intuitive meaning that all occurrences of D-variables x_i are simultaneously replaced by t_i whereas the application of a substitution component $x_j \mapsto [g_1\ldots g_n]$ for a W-variable x_j splices the partial world-path $[g_1\ldots g_n]$ into each world-path at the place of an occurrence of x_j, i.e. for example $\{x_i \mapsto [g_1\ldots g_n]\}$ $[\ldots a\ x_i b\ \ldots] = [\ldots a\ g_1\ldots g_n\ b\ldots]$. Furthermore, in symmetric interpretations the rewrite rule $[g(a_1,\ldots, a_n)g^{-1}(a_1,\ldots, a_n)] \rightarrow []$ will implicitly be applied whenever it is possible.
(We shall see that this is sufficient to avoid the occurrence of inverse functions in the resolvents.)

Lemma 5.2.4 (Some useful properties of idempotent substitutions.)

a) A substitution σ is idempotent iff $DOM(\sigma) \cap VCOD(\sigma) = \emptyset$.

b) For an idempotent substitution σ and a term t: $DOM(\sigma) \cap Vars(\sigma t) = \emptyset$.

c) For two idempotent substitutions σ, θ: $DOM(\sigma) \cap VCOD(\theta) = \emptyset \Rightarrow \sigma\theta$ is idempotent.

The proofs are similar to the proofs for standard substitutions in predicate logic which can for instance be found in [Herold 83] or [Herold 87]. ■

Lemma 5.2.5 Substitutions can be turned into semantic variable assignments:

a) For a P-interpretation $\Im_P$ with initial world $\Im_0$ and a substitution $\sigma = \{x_1 \mapsto t_1,\ldots,x_n \mapsto t_n\}$ with $\Im_P(t_i) \neq \perp$ for i = 1,...,n, the object $\Im_P[\sigma] := \Im_P[x_1/\Im_P(t_1),\ldots, x_n/\Im_P(t_n)]$ is again a P-interpretation.

b) Given a substitution σ such that $\Im_P[\sigma]$ is a P-interpretation, for every term t: $\Im_P[\sigma](t) = \Im_P(\sigma t)$.

Proof: a) This follows immediately from the fact that substitutions map W-variables to $\Re$-admissible world-paths which are interpreted as world-access functions (lemma 5.2.2).

b) We perform an induction on the structure of terms and world-paths.

The two base cases t = [] and t is a variable is trivial.

Induction step:

Case 1: $t = f(wp,s_1,\ldots,s_n)$ where f is a D-valued function symbol.

$$\begin{aligned}
\Im_P[\sigma](f(wp,s_1,\ldots,s_n)) &= \Im_P[\sigma](wp)(\Im_0)(f)(\Im_P[\sigma](s_1),\ldots,\Im_P[\sigma](s_1)) && \text{(def. 3.2.3)}\\
&= \Im_P(\sigma wp)(\Im_0)(f)\ (\Im_P(\sigma s_1),\ldots,\Im_P(\sigma s_1)) && \text{(ind. hypothesis)}\\
&= \Im_P(\sigma t).
\end{aligned}$$

Case 2: $t = g(s_1,\ldots,s_n)$ where g is a W-valued function symbol.

$$\begin{aligned}
\Im_P[\sigma](g(s_1,\ldots,s_n)) &= \Im_W(g)(\Im_P[\sigma](s_1),\ldots,\Im_P[\sigma](s_1)) && \text{(def. 3.2.3)}\\
&= \Im_W(g)\ (\Im_P(\sigma s_1),\ldots,\Im_P(\sigma s_1)) && \text{(ind. hypothesis)}\\
&= \Im_P(\sigma t).
\end{aligned}$$

Case 3: t = [p .s]

$$\begin{aligned}
\Im_P[\sigma]([p\ .s]) &= \Im_P[\sigma](p)\circ\Im_P[\sigma](s)\\
&= \Im_P(\sigma p)\circ\Im_P(\sigma s) && \text{(ind. hypothesis)}\\
&= \Im_P(\sigma t). && \blacksquare
\end{aligned}$$

5.3 Prefix-Stability - An Invariant on the Structure of Terms.

In lemma 4.1.2 we noticed that terms of a translated M-formula have the property that all occurrences of a W-variable have identical subterms (M-adjusted termstructure). A term like f([w], g[vw]) for example can not be the result of a translation. This property ensures that the unification algorithms for transitive models are finitary, and should therefore be preserved for resolvents also. This property is now formally defined for terms and substitutions and some consequences are shown

Definition 5.3.1 (Prefix-Stable Terms and Prefix-Preserving Substitutions)
Let s be a term, a world-path, a list or a set of those. Let x be a W-variable and let σ be a substitution.

- *prefix-stable* (s) :$\Leftrightarrow \forall u \in$ W-Vars(s): |prefix(u, s)| = 1.
- If s is prefix-stable then *prefix-preserving* (σ, s) :$\Leftrightarrow$ prefix-stable (σs). ■

Examples for prefix stable terms and prefix-preserving substitutions.
prefix(w, f([w v], g[v w])) = {[w], [v w]}. This term is not prefix-stable.
prefix(w, f([w], g[w v])) = {[w]}. This term is prefix-stable.
prefix-preserving({w ↦ |k(a|v|)}, [w])
not prefix-preserving({w ↦ [k(a[v|)}, f([w], k(a|c v|)),
i.e. the a substitution may be prefix-preserving for a prefix-stable term s and not prefix-preserving for another prefix-stable term t. ■

Lemma 5.3.2 All subterms of prefix-stable terms are also prefix-stable.
The proof is obvious. ■

The next lemma gives the conditions under which a substitution that is prefix-preserving for a set of terms (for instance the unified terms in a resolution step) is also prefix preserving for a larger set of terms (for instance all terms occurring in the resolvent.)

Lemma 5.3.3 (Lifting of the Prefix-Preserving Predicate)
Let s and t be terms, world-paths, lists or a sets of those and let σ be a substitution, then
prefix-stable(s) $\wedge$ prefix-preserving(σ, s) $\wedge$ (Vars(σ) \ Vars(s)) $\cap$ Vars(t) = $\emptyset$ $\wedge$ prefix-stable((s,t)) $\Rightarrow$
prefix-preserving(σ, (s,t))

Proof: Let a) prefix-stable(s) and b) prefix-preserving(σ,s) and
c) (Vars(σ) \ Vars(s)) $\cap$ Vars(t) = $\emptyset$ and d) prefix-stable((s,t)).
Let u $\in$ W-Vars(σ(s,t)).

Case 1:	u $\notin$ Vars(σ)	
	$\Rightarrow$ u $\in$ Vars((s,t))	
	$\Rightarrow$ prefix(u, (s,t)) is unique	(d)
	$\Rightarrow$ prefix(u, σ(s,t)) is unique.	(since u $\notin$ Vars(σ))
Case 2:	u $\in$ Vars(σ)	
	$\Rightarrow$ u $\in$ COD(σ)	(since u $\in$ W-Vars(σ(s,t))
	$\Rightarrow \exists\, y_1 \ldots y_k \in$ Vars(s,t): u $\in \sigma y_i$	
	$\Rightarrow \{y_1 \ldots y_k\} \subseteq$ Vars(s)	(c)
	$\Rightarrow \forall y \in \{y_1 \ldots y_k\}$: y is a W-variable implies prefix(y, s) = prefix(y, (s,t)) is unique	(d)

$\Rightarrow$ Each occurrence of u in $\sigma(s,t)$ that is caused by a replacement of some $y \in \{y_1 \ldots y_k\}$ has a unique prefix. (b)

In addition the occurrences of u in $\sigma(s,t)$ which stem from (s,t) have to be considered:

Case 2.1: $u \notin Vars(s)$

$\Rightarrow u \notin Vars(t)$ (c)

$\Rightarrow$ Each occurrence of u in $\sigma(s,t)$ is caused by a replacement of some $y \in y_1 \ldots y_k$ and has therefore a unique prefix.

Case 2.2: $u \in Vars(s)$

$\Rightarrow$ prefix(u, (s, t)) is unique. (d)

prefix-preserving(σ, s) implies that each occurrence of u in σs has the same prefix regardless whether it stems from s or from a replacement by σ.

$\Rightarrow$ prefix(u, σs) = prefix(u, $\sigma(s,t)$).

$\Rightarrow$ prefix(u, $\sigma(s,t)$) is unique.

$\Rightarrow$ prefix-preserving(σ, (s, t)). ■

It is instructive to see what will happen if the disjointness condition $(Vars(\sigma) \setminus Vars(s)) \cap Vars(t) = \emptyset$ is not satisfied: Let $\sigma := \{w \mapsto v\}$, s := [w] and t := [a v]. Clearly prefix-stable(s), prefix-preserving(σ, s) and prefix-stable((s, t)), but σs = [v] and σt = [a v] and therefore prefix-preserving(σ, (s,t)) is not true. The disjointness condition therefore forces a unification algorithm to build a unifier either with W-variables from the terms to be unified or with completely new W-variables in the codomain. (This very natural condition is well known from theory unification algorithms for associativity for instance.)

An important property of prefix-stable terms is that they have no multiple occurrences of a W-variable on the toplevel of a world-path (toplevel linearity). Furthermore a W-variable in a prefix-stable term cannot occur more than once in its own prefix (prefix linearity). Nevertheless it is possible, that a W-variable occurs a second time behind its first occurrence in a prefix-stable world-path, however only at a deeper level of nesting, as for example in [w g(a[w v])].

Definition 5.3.4 (Toplevel and Prefix Linearity)

Let wp be a world-path and let s be a term.

- wp is called *toplevel linear* iff no variable occurs more than once at the toplevel of wp.
  s is called toplevel linear iff each world-path in s is toplevel linear.
- wp is called *prefix linear* iff each toplevel variable u in wp =: $[s_1 \ldots s_{k-1}\ u \ldots]$ does not occur in $[s_1 \ldots s_{k-1}]$.
  s is called prefix linear iff each world-path in s is prefix linear. ■

Lemma 5.3.5 (Prefix-Stable Terms are Toplevel and Prefix Linear)

Each prefix-stable finite term s is toplevel linear and prefix linear.

Proof: a) Toplevel linearity: If there was a world-path $[s_1 \ldots s_{k-1}\ u\ s_{k+1} \ldots u \ldots]$ in s with two occurrences of a variable u, then u would have two different prefixes, which contradicts the prefix-stability of s.

b) Prefix linearity: If there was a world-path $[s_1 \ldots f(g([\ldots u \ldots])) \ldots s_k\ u \ldots]$ in s where the variable u occurs in its prefix $[s_1 \ldots f(g([\ldots u \ldots])) \ldots s_k]$, this second occurrence must have the same prefix, i.e. again another occurrence of u in its prefix, etc., otherwise s would not be prefix-stable. This is not possible in finite terms. ■

Chapter Six

Modal Unification

6.1 Introduction

Unification is the basic operation in resolution based deduction systems. In this chapter we define the unification algorithms for atoms, terms and world-paths in P-logic. Since there is no equational theory for the function symbols, they can be treated like free function symbols in first-order logic and unified in the usual way. For the unification of world-paths, however, we need additional algorithms. As we shall see in chapter 8, only $\Re$-admissible substitutions are allowed as unifiers, consequently for the unification of world-paths we need a specialized algorithm for the different accessibility relation types. However, the algorithms do not depend on the seriality or non-seriality of the accessibility relation. Only reflexivity, symmetry, transitivity and their combinations have to be considered. Before going into technical details, let us discuss briefly the characteristics of each unification algorithm for world-paths.

Unification where the Accessibility Relation has no Special Properties (Modal Logics K and D)

$\Re$-admissible substitutions are allowed to substitute a partial world-path with *exactly* one W-term for a W-variable. Two world-paths like [v a] and [b w] are therefore unifiable with a unifier $\{v \mapsto b, w \mapsto a\}$, whereas the two world-paths [v a] and [b u w] would require a non-$\Re$-admissible substitution $\{v \mapsto [b\,u], w \mapsto a\}$. They are not unifiable.

In general two world-paths are unifiable when they have equal length and the W-terms are pairwise unifiable with compatible unifiers. Thus, the world-paths can be treated like ordinary terms and except that the argument lists may be of different length, there is no difference to the unification of first-order terms. *There is at most one most general unifier*, which is unique up to variable renaming, for every unification problem, i.e. the unification is of type *unitary*. When an algorithm in the style of Martelli & Montanari's algorithm for free first order terms is used, the complexity of the unification is therefore linear [Martelli&Montanari 82].

Unification where the Accessibility Relation is Reflexive. (Modal Logic T)

The substitution component $w \mapsto []$ represents the assignment of the identity mapping to a W-variable. It is $\Re$-admissible, because in reflexive interpretations a world is accessible from itself. Therefore $\Re$-admissible substitutions are allowed to substitute a partial world-path with *at most* one W-term for a W-variable. The substitution components $w \mapsto []$ allow to remove a variable completely from a world-path such that for example the world-paths [v a] and [b u w] are unifiable with the two indepen-

dent unifiers $\{v \mapsto b, u \mapsto [], w \mapsto a\}$ and $\{v \mapsto b, u \mapsto a, w \mapsto []\}$. Hence, this kind of unification is akin to unification of first order terms with an identity element.

The unification algorithm enumerates all possibilities to remove W-variables w by the substitution component $w \mapsto []$ and then unifies the W-terms in the reduced world-paths pairwise. Since there are only finitely many variables to be removed, there are *at most finitely many most general unifiers* for each unification problem, i.e. the unification is of type *finitary*. The number of unifiers computed in this way is at most 2^n where n is the number of W-variables in the terms to be unified. (This is the size of the powerset of W-variables.) Therefore the complexity of the unification is exponential and there is no chance to do it better.

Unification where the Accessibility Relation is Symmetric. (Modal Logic DB)

In symmetric interpretations, each W-valued function symbol has an associated inverse function symbol. A substitution component $w \mapsto a^{-1}$ for example is therefore suitable for collapsing the partial world-path [a w] into $[a\, a^{-1}] = []$. $\Re$-admissible substitutions are allowed to substitute a partial world-path with *exactly* one W-term or an "inverse" W-term for a W-variable. The "inverse" v^{-1} of a W-variable is also allowed, because the interpretation of a W-variable is also a function whose inverse exists in symmetric interpretations. For example the two world-paths [v w] and [] are unifiable with a unifier $\{w \mapsto v^{-1}\}$.

The unification algorithm must consider all possibilities to collapse a W-variable w and its predecessor t in the world-path by the substitution component $w \mapsto [t^{-1}]$ to the empty path [] and to unify the W-terms in the reduced world-paths pairwise. Since there are only finitely many variables to be collapsed, there are *at most finitely many most general unifiers* for each unification problem, i.e. the unification is again of type *finitary*. The number of unifiers is at most $2^{n/2}$ where n is the number of W-variables in the terms to be unified. Therefore the complexity of the unification is also exponential.

This is the first case where the prefix-stability of terms can be exploited: When a W-variable w and its predecessor t in a world-path have been collapsed with the substitution component $w \mapsto t^{-1}$, we know that in all other terms in the clause set, t is the predecessor of w. The application of $w \mapsto t^{-1}$ to an arbitrary term containing w in the clause set will therefore collapse [t w] to []. No inverse W-term will ever occur in an instantiated term, thus we need not investigate the unification of such terms.

Unification where the Accessibility Relation is Reflexive and Symmetric. (Modal Logic B)

The two basic techniques for reflexivity and symmetry can now be joined: The unification algorithm enumerates all possibilities for the removal of W-variables w by the substitution component $w \mapsto []$ and for collapsing a W-variable w and its predecessor t in the world-path by the substitution component $w \mapsto t^{-1}$ and then unifies the W-terms in the reduced world-paths pairwise. Since there are only finitely many variables to be collapsed or to be removed, there are *at most finitely many most general unifiers* for each unification problem.

For example the two world-paths [a u v] and [w] are unifiable with the two independent unifiers $\{u \mapsto a^{-1}, v \mapsto w\}$ and $\{v \mapsto u^{-1}, w \mapsto a\}$.
The number of unifiers may again be exponential in the number of W-variables.

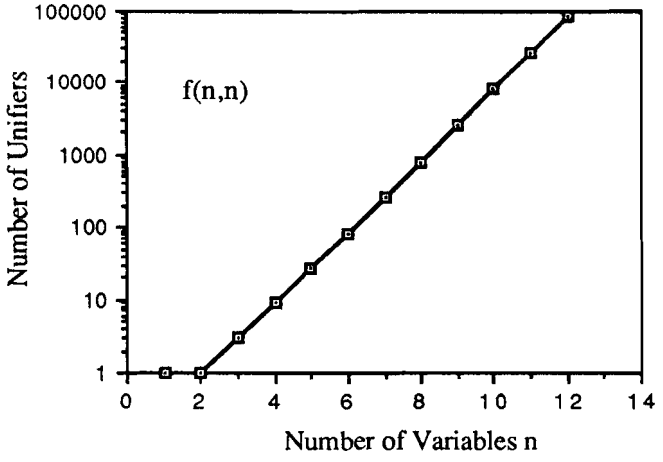
Unification where the Accessibility Relation is Transitive. (Modal Logics K4 and D4)

$\mathfrak{R}$-admissible substitutions may now substitute arbitrary partial world-paths for a W-variable. For example a unifier for the two world-paths [v c d] and [a b w d] is $\{v \mapsto [a\ b], w \mapsto c\}$, but the substitution $\{v \mapsto [a\ b\ w'], w \mapsto [w'\ c]\}$ with a new W-variable w' is also a unifier. Thus, we introduce a variable splitting technique similar to the splitting technique in the unification algorithm for associative functions. This is in general infinitary, but fortunately it turns out that the toplevel linearity of world-paths is sufficient to *keep this unification finitary*. The number of unifiers, however, can be extremely large. When for example two world-paths $[v_1 \ldots v_n] = [u_1 \ldots u_m]$ consisting of variables only are to be unified, all possibilities to assign v_1 and w_1 to leading parts of the opposite world-path must be enumerated and the corresponding tails of the world-paths must be unified recursively. The number f(n,m) of unifiers - without variable splitting - can then be calculated with the following recursive formula:

$f(n, 0) = f(0,m) = 0, \quad f(n,1) = f(1,m) = 1$

$f(n,m) = f(n-1,m-1) + \ldots + f(n-1,1) + f(n-2, m-1) + \ldots + f(1, m-1).$

The graph of the function f(n,n), i.e. the number of unifiers for world-paths of equal length is drawn below:



It shows a clear exponential behaviour which is worse than 2^n.

Unification where the Accessibility Relation is Reflexive and Transitive. (Modal Logic S4)

The algorithms for the reflexive case and for the transitive case can be joined without any further problems. The algorithm for the transitive case must be augmented by a step that removes W-variables w with a substitution component $w \mapsto []$. There are still at most *finitely many most general unifiers* for each unification problem. The number of unifiers can again be exponential with the number of W-variables.

Unification where the Accessibility Relation is an Equivalence Relation (Modal Logic S5)

World-paths for S5 interpretations have a special normal form (of modal degree one) consisting of at most one W-term, i.e. they look like [] or [t]. Two world-paths [] and [t] can only be unified when t is a variable, the unifier is $t \mapsto []$. Two world-paths [s] and [t] can be unified when s and t are unifiable. Therefore there is *at most one most general unifier* for each unification problem.

The following definition formally introduces the usual notions of complete and minimal sets of unifiers. Since the accessibility relation plays the same role as an equational theory in equational based unification we adopt the notions of unification under equational theories (c.f. [Schmidt-Schauss 87]).

Definition 6.1.1 (Complete and Minimal Sets of $\mathfrak{R}$-Unifiers)
Let $\Gamma := \{"s_i = t_i" \mid i = 1,\ldots,n\}$ be a system of equations for prefix-stable D-terms, W-terms, atoms or world-paths.
Let $\mathfrak{R}$ be an accessibility relation.

a) A substitution σ *$\mathfrak{R}$-unifies* Γ iff σ is $\mathfrak{R}$-admissible and for every equation "$s_i = t_i$" in Γ we have $\sigma s_i = \sigma t_i$. In this case we say σ is an $\mathfrak{R}$-unifier for Γ, or simply σ unifies Γ.
The set of all $\mathfrak{R}$-unifiers for Γ is denoted by $U_{\mathfrak{R}}(\Gamma)$.

b) A *complete* set $cU_{\mathfrak{R}}(\Gamma)$ of unifiers for Γ is a set satisfying
   i) $cU_{\mathfrak{R}}(\Gamma) \subseteq U_{\mathfrak{R}}(\Gamma)$ (correctness)
   ii) $\forall \sigma \in U_{\mathfrak{R}}(\Gamma)\ \exists \tau \in cU_{\mathfrak{R}}(\Gamma)\colon\ \tau \leq_{\mathfrak{R}} \sigma\ [Vars(\Gamma)]$ (completeness)

c) A complete set is called *minimal* or a set of *most general unifiers* (mgus), iff additionally
   iii) $\forall \sigma,\tau \in cU_{\mathfrak{R}}(\Gamma)\colon\ \tau \leq_{\mathfrak{R}} \sigma\ [Vars(\Gamma)] \Rightarrow \tau = \sigma$ (minimally)
   Minimal sets are also denoted as $\mu U_{\mathfrak{R}}(\Gamma)$. ■

6.2 Unification as Transformations on Systems of Equations

We consider the process of unification as a sequence of - in general nondeterministic - transformations on systems of equations that starts with the terms or atoms "p = q" to be unified and terminates in the positive case with a system "$x_i = t_i$" in solved form. The nondeterministic choice of the transformation rules generates a tree like search space where the nodes are the actual state of the equation system. Each successful transformation chain computes a unifier for p and q. This follows the ideas in [Herband 30], [Martelli&Montanari 82] and others. We shall divide a system of equations into an *unsolved ordered* part Γ, an *ordered* set, that initially contains the single equation {"p = q"} to be solved, and into an initially empty *solved* part σ with components of the form "x = t" such that σ represents an idempotent substitution.

The transformation starts by checking the trivial cases, i.e. whether or not the initial system "p = q" is already in the form "x = t".

Each transformation replaces a system Γ, σ by a modified system Γ', σ' as follows:

- Pick the *left most* equation $s = t \in \Gamma$ (depth first, left to right selection, Γ is ordered).
  Remove $s = t$ from Γ.
- Select from the set of admissible transformation rules a rule $\mathcal{T}$ which is applicable to $s = t$ or $t = s$.
  If no rule is applicable then terminate this branch in the search space with failure.
- Apply the rule $\mathcal{T}$ to $s = t$ (or $t = s$ respectively).
  Let $s_1 = t_1 \,\&\, \ldots \,\&\, s_n = t_n$ be the result of the transformation.
- For $i = n \ldots 1$:
  If s_i equals t_i then ignore this component (*tautology rule*).
  If s_i and t_i are both non-variable terms then push $s_i = t_i$ at the front of Γ.
  otherwise let w.l.o.g s_i be a variable.
  If $s_i \in t_i$ (occurs check) or if s_i is a W-variable and t_i is a non-$\mathfrak{R}$-admissible world-path
  then terminate this branch in the search space with failure,
  otherwise replace all occurrences of s_i in Γ and σ by t_i (*application rule*) and
  insert $s_i = t_i$ into σ.

It is noted that we imposed a Prolog like depth first, left to right linear selection strategy and an immediate application of the computed substitutions on the control structure of the transformation process. This ensures that an equation is completely solved once it is selected before the next one is attacked. This strategy simplifies the termination proof of the splitting rule (see below) considerably.

The following transformation rules are needed to build unification systems for P-logic:
(The letters written outlined denote - possibly empty - strings of W-terms.)

Definition 6.2.1 **(Transformation Rules)**
The transformation system *P-Unify* consists of the following rules:

$f(s_1,\ldots,s_n) = f(t_1,\ldots t_n)$	$\rightarrow s_1 = t_1 \,\&\ldots\, \& s_n = t_n$	(Decomposition)
$[\mathbf{s}\ \mathbf{s}] = [\mathbf{t}\ \mathbf{t}]$	$\rightarrow \mathbf{s} = \mathbf{t} \,\&\, \mathbf{s} = \mathbf{t}$	(Separation)
$[\mathbf{s}\ w\ \mathbf{s}'] = \mathbf{t}$	$\rightarrow w = [] \,\&\, [\mathbf{s}\ \mathbf{s}'] = \mathbf{t}$	(Identity)
$[\mathbf{s}\ s\ w\ \mathbf{s}'] = \mathbf{t}$	$\rightarrow w = s^{-1} \,\&\, [\mathbf{s}\ \mathbf{s}'] = \mathbf{t}$	(Inverse)
$[w\ \mathbf{s}] = [\mathbf{t}\ \mathbf{t}']$	$\rightarrow w = \mathbf{t} \,\&\, \mathbf{s} = \mathbf{t}'$	(Path-Separation)
$[w\ s\ \mathbf{s}] = [\mathbf{t}\ t\ v\ \mathbf{t}']$	$\rightarrow v = [v_1\ v_2] \,\&\, w = [\mathbf{t}\ t\ v_1] \,\&\, [\mathbf{s}\ \mathbf{s}] = [v_2\ \mathbf{t}']$	(Splitting)

if s and t exist. v_1 and v_2 are new variables. ■

Since the application rule explicitly checks the equations for $\mathfrak{R}$-admissibility, it is in principle not necessary to define separate rule systems for each type of the accessibility relation. In order to obtain more efficient algorithms, however, it seems a good idea to group the unification rules according to the type of the accessibility relation such that those rules are eliminated which produce non-$\mathfrak{R}$-admissible substitution components. We do this in the following way:

Definition 6.2.2 (Logic Dependent Unification Rule Systems)

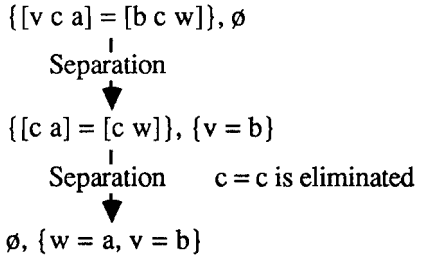
Name of the system	applicable to modal logics	properties of the accessibility relation	transformation rules
K-D	K, D	no special properties	Decomposition, Separation
T	T	reflexivity	System K-D, Identity
DB	DB	symmetry	System K-D, Inverse
B	B	symmetry and reflexivity	System T, Inverse
K4-D4	K4, D4	transitivity	System K-D, Path-Separation, Splitting
S4	S4	reflexivity and transitivity	System K4-D4, Identity
S5	S5	equivalence relation	Decomposition

■

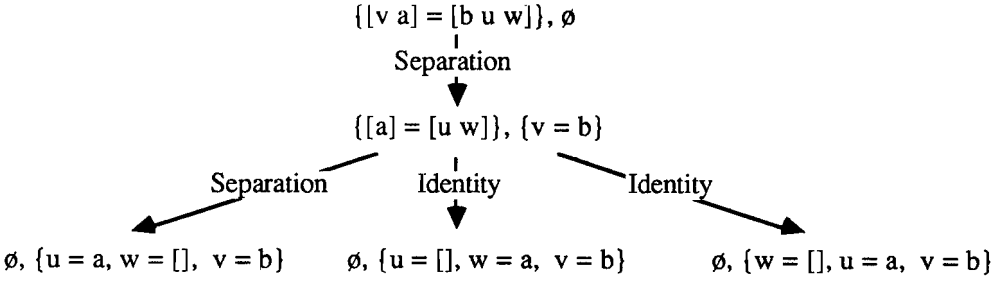
Examples 6.2.3 (for some applications of the unification rules).

In the following examples we show only the most important transformation branches.

1. Suppose the accessibility relation has no special properties. We apply the rule system K-D.
   We unify [v c a] with [b c w]:



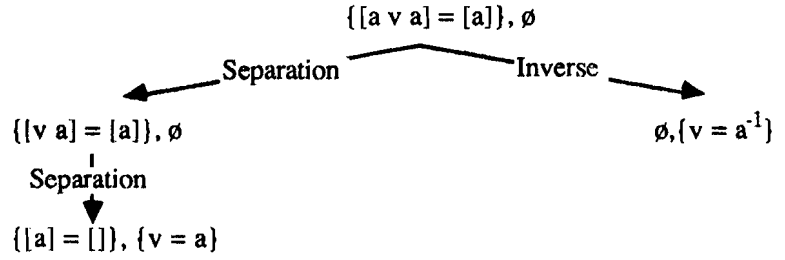
2. Suppose the accessibility relation is reflexive. We apply the rule system T.
   We unify |v a| with |b u w|:



(The third solution is redundant, i.e. the algorithm is *not minimal.*)

3. Suppose the accessibility relation is symmetric. We apply the rule system DB

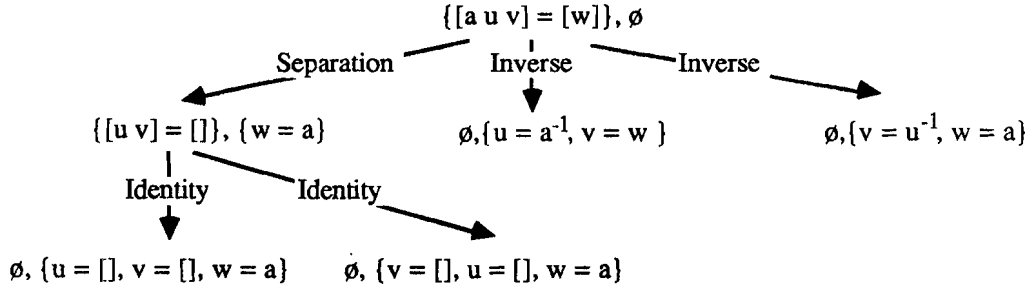
We unify [a v a] with [a]:



Failure since no further rule is applicable

4. Suppose the accessibility relation is symmetric and reflexive. We apply the rule system B.

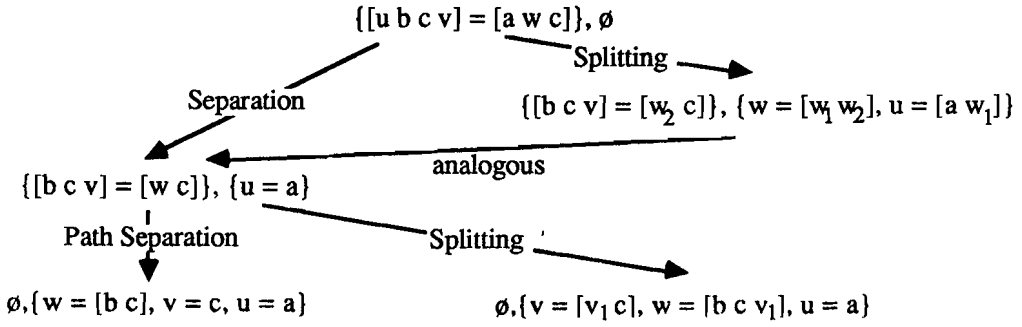
We unify [a u v] with [w]:



The algorithm computes again superfluous unifiers. The two most general unifiers are $\{u \mapsto a^{-1}, v \mapsto w\}$ and $\{v \mapsto u^{-1}, w \mapsto a\}$.

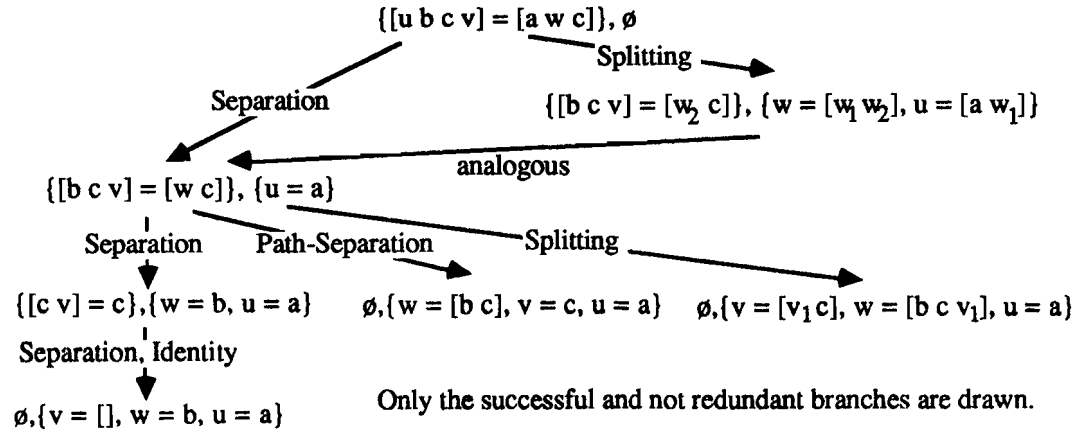
5. Suppose the accessibility relation is transitive. We apply the rule system K4-D4.

We unify [u b c v] with [a w c]:



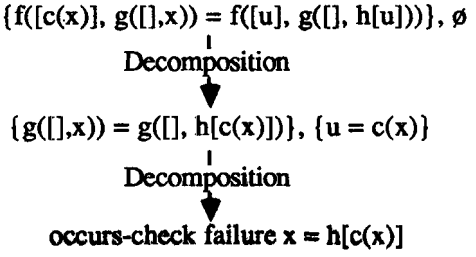
Only the successful and not redundant branches are drawn.

6. Suppose the accessibility relation is reflexive and transitive. We apply the rule system S4.
We unify [u b c v] with [a w c]:



Only the successful and not redundant branches are drawn.

7. Suppose the accessibility relation is an equivalence relation.
The world-paths are in modal degree one normal form.We apply the rule system S5.
We unify f([c(x)], g([], x)) with f([u] g([], h[u])):



■

Remark

The correspondences between the properties of the accessibility relation and the type of the unification algorithm was first recognized by L.Wallen [Wallen 87]. Exploiting the prefix stability of terms we were however able to optimize the corresponding algorithms for semigroups, monoids etc.

In the next three sections we prove the soundness, termination and completeness of the unification rule system P-Unify as defined in 6.2.1. Since the control algorithm for the transformation explicitly checks the equations for $\mathfrak{R}$-admissibility, it is not necessary to consider the rule systems for each type of the accessibility relation separately.

6.3 Soundness of the Unification Procedure

First we show that the unification rules compute idempotent, $\mathfrak{R}$-admissible, prefix-preserving unifiers for the terms to be unified, i.e. their soundness.

Lemma 6.3.1 The unification rules P-Unify compute idempotent and $\mathfrak{R}$-admissible substitutions.
Proof: Idempotence and $\mathfrak{R}$-admissibility is explicitly ensured in the control algorithm for the transformation which is the only place where new equations are inserted into the solved part. The idempotence follows from the occurs-check and the fact that the application of an equation $x = t$ to the other equations removes x completely from the equation system before $x = t$ is inserted into the solved part. ■

Lemma 6.3.2 If the unification rules P-Unify are applied to terms p and q, the resulting substitutions actually unify p and q.
Proof: In order to prove that the substitutions actually unify the terms p and q we must show that no transformation $\Gamma,\sigma \rightarrow \Gamma',\sigma'$ of the equations increases the set of solutions i.e. $U_{\mathfrak{R}}(\Gamma\cup\sigma) \supseteq U_{\mathfrak{R}}(\Gamma'\cup\sigma')$. With other words we must analyze each transformation rule in def. 6.2.1 and show that a unifier for $\Gamma'\cup\sigma'$ is also a unifier for $\Gamma\cup\sigma$. Since this is trivial, we omit these proofs. By induction on the length of a transformation chain we can then conclude that the unifier corresponding to the solved equation system unifies the original terms p and q. ■

Lemma 6.3.3 If the unification rules P-Unify are applied to terms p and q, the resulting substitutions are prefix-preserving for p and q.
Proof: The idea for the proof is as follows: At first we modify the transformation system slightly:
After selecting the initial equation "p = q" from the initial equation system Γ_0, "p = q" is not removed from Γ such that it is automatically instantiated when a new equation "x = t" is computed. When the procedure terminates, "p = q" is then instantiated with the computed unifier σ. Obviously the initial system Γ_0 is prefix-stable. Now we show for each transformation $\Gamma \rightarrow \Gamma'$ that the transformed system Γ' is again prefix-stable. With induction on the length of the computation path, we obtain that the final system which now contains $\sigma p = \sigma q$ is prefix-stable, i.e. σ is prefix-preserving for p and q.

Unfortunately the separation rules remove leading parts of the world-path. Therefore the intermediate equation systems are not prefix-stable, just because the unified leading parts of the world-paths are removed. For the purposes of this proof we therefore modify the transformation system again by adding to each equation "s = t" the removed leading part $\mathfrak{p}$, i.e. we manipulate tuples $\mathfrak{p}$, "s = t". $\mathfrak{p}$ has no influence on the algorithm, but it is used to prove the invariant that the intermediate equation systems Γ are prefix-stable. $\mathfrak{p}$ is automatically instantiated when a new equation "x = a" is computed. The depth

first, left to right selection strategy for the rule application ensures that $\mathbb{p}$ is always the common unified part of the world paths $[\mathbb{p}\ s]$ and $[\mathbb{p}\ t]$. The prefixes of the variables both in s and t are therefore computed with respect to $\mathbb{p}$.

With the modified transformation system we can now prove that each transformation $\Gamma \rightarrow \Gamma'$ leaves the prefix-stability invariant:

Decomposition: $f(s_1,\ldots,s_n) = f(t_1,\ldots t_n) \rightarrow s_1 = t_1 \ \&\ldots\ \& \ s_n = t_n$

If none of the s_i and t_i is a variable, Γ' is obviously prefix-stable.

If there are components $x = s$ among the $s_i = t_i$ we can apply lemma 5.3.3 to $\{x \mapsto s\}$, yielding prefix-preserving $(\{x \mapsto s\}, (\{x, s\}, \Gamma))$.

With induction on the number of these variable-term pairs we obtain that Γ' is prefix-stable.

Separation: $\mathbb{p}, [s\ s] = [t\ t] \rightarrow s = t\ \&\ [\mathbb{p}\ s], s = t.$

The left to right selection strategy of the rule application ensures that before $[\mathbb{p}\ s]$, $s = t$ is selected as the new equation to be transformed, s and t are unified. Therefore it does not matter if $[\mathbb{p}\ s]$ or $[\mathbb{p}\ t]$ is taken for the new prefix of $s = t$.

If neither s nor t is a variable then Γ' is obviously prefix-stable.

W.l.o.g let t be a variable and $t \notin s$.

Then we have prefix-stable($[\mathbb{p}\ s]$, Γ) and

since $t \notin s$ and $t \notin \mathbb{p}$ = prefix(t, Γ): prefix-preserving($\{t \mapsto s\}$, $([\mathbb{p}\ s], [\mathbb{p}\ t])$).

Furthermore $Vars(\{t \mapsto s\}) \setminus Vars([\mathbb{p}\ s], [\mathbb{p}\ t])) = \emptyset$.

Thus, we can again apply lemma 5.3.3 yielding that Γ' is prefix-stable.

Identity: $\mathbb{p}, [s\ w\ s'] = t \rightarrow w = [] \ \&\ \mathbb{p}, [s\ s'] = t$.

w vanishes simultaneously from all world-paths in Γ, therefore Γ' is again prefix-stable.

Inverse: $\mathbb{p}, [s\ s\ w\ s'] = t \rightarrow w = s^{-1}\ \&\ \mathbb{p}, [s\ s'] = t.$

[sw] vanishes simultaneously from all world-paths in Γ, therefore Γ is again prefix-stable.

Path-Separation: $\mathbb{p}, [w\ s] = [t\ t'] \rightarrow w = t\ \&\ [\mathbb{p}\ w], s = t'$.

With the same arguments as for the separation rule we obtain that Γ' is again prefix-stable.

Splitting: $\mathbb{p}, [w\ s\ s] = [t\ t\ v\ t'] \rightarrow v = [v_1\ v_2]\ \&\ w = [t\ t\ v_1]\ \&\ [\mathbb{p}\ w], [s\ s] = [v_2\ t']$
if both s and t exist. v_1 and v_2 are new variables.

We can assume $w \notin [t\ t]$ (otherwise there is an occurs check clash.)

Let $\lambda := \{v \mapsto [v_1\ v_2], w \mapsto [t\ t\ v_1]\}$ or $\lambda := \{v \mapsto [v_1\ s\ s], w \mapsto [t\ t\ v_1]\}$ when $t' = []$.

We have prefix-stable($[\mathbb{p}\ w]$, $[\mathbb{p}\ t\ t\ v]$).

Since $v \notin [\mathbb{p}\ t\ t]$ = prefix(v, Γ) and $w \notin \mathbb{p}$ =prefix(v, Γ), thus $w \notin [\mathbb{p}\ t\ t]$.

Therefore $Vars(\lambda) \setminus Vars([\mathbb{p}\ w], [\mathbb{p}\ t\ t\ v]) \subseteq \{v_1\ v_2\}$ and $\{v_1\ v_2\} \cap Vars(\Gamma) = \emptyset$ (v_1,v_2 are new).

Finally we can apply lemma 5.3.3 once again yielding that Γ' is prefix-stable. ■

Corollary 6.3.4 When the transformation system is applied to prefix-stable terms p and q, the variables w and v in the splitting rule

$[w\ s\ s] = [t\ t\ v\ t'] \rightarrow v = [v_1\ v_2]\ \&\ w = [t\ t\ v_1]\ \&\ [s\ s] = [v_2\ t']$

are always different. Furthermore if $w \notin [t\ t]$, w and v do not occur at toplevel of $[s\ s]$ and t'.

Proof: Applying the same construction as in the previous lemma, we obtain

prefix-stable($([\mathbb{p}\ w\ s\ s]$, $[\mathbb{p}\ t\ t\ v\ t']$).

Since v and w have different prefixes, they cannot be equal. Because of the toplevel linearity of the two world-paths (lemma 5.3.5), w cannot occur in $[s\ s]$ and v cannot occur in t'. If w would occur at the toplevel of t', v would occur in the prefix of w, i.e. in its own prefix $[\mathbb{p}\ t]$ which is impossible (lemma

5.3.5). If v would occur at the toplevel of [s s], w would occur in the prefix of v, i.e. because $w \notin [t\,t]$ w would occur in p which is again impossible. ■

Collecting the results we can finally state:

Theorem 6.3.5 (Soundness of the Unification)
Applied to prefix-stable terms or atoms p and q, the unification rules P-Unify compute idempotent, prefix-preserving and $\mathfrak{R}$-admissible unifiers for p and q. ■

6.4 Termination of the Unification Procedure

In order to prove the termination of the unification process we define a well founded complexity measure $\mu(\Gamma)$ for the *unsolved* part of the equation system and show for each node Γ,σ in the search space: If μ is the current measure for Γ then in each branch below Γ,σ there is a finite number of transformations after which the measure is smaller than μ.

For an equation system Γ let $\mu(\Gamma) =: (V, S)$ where

$V = Vars(\Gamma)$ and

$$S = \text{number of symbol occurrences in } \Gamma + \sum_{wp=\text{world-path in } \Gamma} |wp|$$

i.e. the length of each world-path is added to the number of symbol occurrences in Γ.
We order μ lexicographically. Since both components are always non-negative, this is clearly a well founded ordering on Γ.

Theorem 6.4.1 The unification rules P-Unify terminate, if applied to prefix-stable terms or atoms.
Proof: We prove for each node Γ,σ in the search tree: If μ is the current measure at Γ then in each branch below Γ,σ there is a finite number of transformations after which the measure is smaller than μ.
Let Γ,σ be the current set of unsolved equations and let $\mu(\Gamma) =: (V, S)$ be the current measure
We examine each possible transformation.
(Remember that equations x=t are immediately applied to all other equations, i.e. x vanishes completely.)

Decomposition: $f(s_1,\ldots,s_n) = f(t_1,\ldots t_n) \rightarrow s_1 = t_1 \,\&\ldots\, \& s_n = t_n$
If one of the s_i or t_i is a variable, V decreases immediately, otherwise S decreases by 2 because the two occurrences of f disappear.

Separation: $[s\,s] = [t\,t] \rightarrow s = t \,\&\, s = t$
In case either s or t is a variable, V decreases immediately, otherwise S decreases by 2 because the two new world-paths are shorter.

Identity: $[s\,w\,s'] = t \rightarrow w = [] \,\&\, [s\,s'] = t$. V decreases at least by 1.

Inverse: $[s\,s\,w\,s'] = t \rightarrow w = s^{-1} \,\&\, [s\,s'] = t$. V decreases at least by 1.

Path-Separation: $[w\,s] = [t\,t']$ $\rightarrow w = t \,\&\, s = t'$. V decreases at least by 1.

Splitting: $[w\ s\ \mathbb{s}] = [\mathbb{t}\ t\ v\ t'] \rightarrow v = [v_1\ v_2]\ \&\ w = [\mathbb{t}\ t\ v_1]\ \&\ [s\ \mathbb{s}] = [v_2\ \mathbb{t}']$

if both s and t exist. v_1 and v_2 are new variables.

According to corollary 6.3.4, w is different to v, furthermore neither w nor v occurs at toplevel of $\mathbb{s}$ and $\mathbb{t}$, therefore the substitution of v and w does not change the length of $\mathbb{s}$ and $\mathbb{t}$. (✪)

Case 1: $\mathbb{t}'$ is empty. Since solved variables are immediately applied to Γ, v and w vanish whereas v_1 is inserted, i.e. the total number of variables deceases by 1.

Case 2: $\mathbb{t}'$ is not empty. We exploit the linear selection strategy for the rule application which states that $[s\ \mathbb{s}] = [v_2\ \mathbb{t}']$ is to be selected next. In this case each of the transformation rules except the splitting rule causes at least one variable to vanish form Γ in the next step, i.e. in these branches the number of variables decreases by 1 after one additional transformation.

The splitting rule can be applied only finitely often to $[s\ \mathbb{s}] = [v_2\ \mathbb{t}']$ because of (✪) each splitting shrinks these world-paths. Therefore the two world-paths will eventually be small enough that case 1 applies and the number of variables decreases by 1.

Since the lexicographic ordering on μ is well founded, no infinite transformation chain is possible. ■

6.5 Completeness of the Unification Procedure

A unification algorithm is said to be complete if it computes a complete set $cU_{\mathfrak{R}}$("p = q") of unifiers for the two given terms p and q. To prove this property for a unification algorithm, according to def. 6.1.1 we must show that for every $\mathfrak{R}$-admissible unifier λ for p and q there is a unifier $\tau \in cU_{\mathfrak{R}}$("p = q") with $\lambda \leq_{\mathfrak{R}} \tau$ [Vars (p, q)].

Theorem 6.5.1 (Completeness of the Unification Rules P-Unify)
Let p and q be two terms or atoms and let λ_0 be an $\mathfrak{R}$-admissible unifier for p and q, i.e. $\lambda_0 p = \lambda_0 q$. Then the rule system P-Unify computes a unifier τ which is more general than λ_0,
i.e. $\lambda_0 \leq_{\mathfrak{R}} \tau$ [Vars (p, q)].

Proof: Let V := Vars (p, q). The idea for the proof is as follows: Starting with the initial node Γ_0,σ_0 = {"p = q"}, ∅, and λ_0 as the initial unifier for Γ_0,σ_0, we extend λ_0 by substitution components for the generated new variables (see the splitting rule) and show for each node Γ,σ in the search tree: if the current extended version λ of λ_0 unifies Γ,σ then there is at least one successor node Γ',σ' = Γ,σ(rule application) which is unified by an extended substitution λ'. Since λ_0 unifies the initial equation system Γ_0,σ_0, by induction on the depth of the search tree we can then conclude that there is a successful search path such that the last version λ_ω of λ_0 unifies the leaf node ∅,τ, i.e. since τ is $\mathfrak{R}$-admissible (lemma 6.3.1), $\lambda_\omega \leq_{\mathfrak{R}} \tau$ [V]. Since $\lambda_{\omega|V} = \lambda_0$ therefore $\lambda_0 \leq_{\mathfrak{R}} \tau$ [V] holds.

Now let Γ,σ be the current node in the search tree and let λ be the current extended version of λ_0. Furthermore let "s = t" $\in \Gamma$ be the equation to be transformed next. With the induction hypothesis we can assume that λ unifies Γ,σ and in particular $\lambda s = \lambda t$. The control algorithm of the transformation system ensures that neither s nor t is a variable. Therefore the following cases remain to be examined:

Case 1: "s = t" =: "$f(s_1 \ldots, s_n) = f(t_1, \ldots, t_n)$".

$\lambda s = \lambda t$ implies $\lambda s_1 = \lambda t_1, \ldots, \lambda s_n = \lambda t_n$. Therefore λ unifies $\Gamma',\sigma' := \Gamma,\sigma$(decomposition).

Case 2: "s = t" =: "$[s_1 \ldots, s_n] = [t_1, \ldots, t_m]$" are world-paths.

Case 2.1 W.l.o.g λ collapses a leading part of t, i.e. $\lambda[t_1, \ldots, t_k] = []$.

(The case that λ collapses a leading part of s is symmetric.)

$\Rightarrow$ Some of the t_i in $[t_1,\ldots,t_k]$ must be variables and either $\lambda t_i = []$ or $\lambda t_i = t_{i-1}^{-1}$.

$\Rightarrow$ λ unifies $\Gamma',\sigma' := \Gamma,\sigma$(identity or inverse).

Case 2.2: λ collapses no leading part of s or t.

Case 2.2.1: $\lambda s_1 = \lambda t_1$.

$\Rightarrow \lambda[s_2\ldots s_n] = \lambda[t_2\ldots t_m]$

$\Rightarrow$ λ unifies $\Gamma',\sigma' := \Gamma,\sigma$(separation).

Case 2.2.2: $\lambda s_1 \neq \lambda t_1$.

Case 2.2.2.1: $\lambda s_1 = \lambda[t_1\ldots t_k]$ for some $k > 1$.

$\Rightarrow \lambda[s_2\ldots s_n] = \lambda[t_k\ldots t_m]$

$\Rightarrow$ λ unifies $\Gamma',\sigma' := \Gamma,\sigma$(path-separation).

Case 2.2.2.2: $\lambda t_1 = \lambda[s_1\ldots s_k]$ for some $k > 1$. This case is symmetric to the previous one.

Case 2.2.2.3: $\lambda s_1 \neq \lambda[t_1\ldots t_k]$ and $\lambda t_1 \neq \lambda[s_1\ldots s_k]$ for every $k > 1$.

$\Rightarrow$ either s_1 or t_1 is a variable and both s_2 and t_2 must exist.

Case 2.2.2.3.1: Both s_1 and t_1 are variables.

Let $\lambda s_1 =: [a_1\ldots a_g]$ and $\lambda t_1 =: [a_1\ldots a_h]$ where w.l.o.g $h > g$ (since $\lambda s_1 \neq \lambda t_1$, $h = g$ is not possible)
Since $\lambda t_1 \neq \lambda[s_1\ldots s_k]$ for every k, there must be another variable s_i, $i > 1$ such that the situation looks as follows:

$$
\begin{array}{lll}
s & = [s_1 & s_2\ldots s_{i-1} \quad s_i \quad s_{i+1}\ldots] \\
 & \quad | & \quad | \quad | (v_1) \quad | (v_2) \quad | \\
\lambda s & = [a_1\ldots a_g & a_{g+1}\ldots a_j\, a_{j+1}\ldots a_h\, b_1\ldots b_l\ \lambda s_{i+1}\ldots] \quad \text{i.e. } \lambda s_i = [a_{j+1}\ldots a_h\, b_1\ldots b_l] \\
\hline
\lambda t & = [a_1\ldots a_g & a_{g+1}\ldots\ldots\ldots\ldots a_h\, b_1\ldots b_l\, c_1\ldots\,] \\
t & = [t_1 & \qquad\qquad\qquad\qquad t_2\ldots\,]
\end{array}
$$

Therefore the splitting rule

"$[w\ s\ \mathbf{s}] = [t\ t\ v\ t'] \rightarrow v = [v_1\ v_2]\ \&\ w = [t\ t\ v_1]\ \&\ [s\ \mathbf{s}] = [v_2\ t']$
if s and t exist. v_1 and v_2 are new variables."

is applicable for splitting s_i.

We split s_i yielding "$s_i = [v_1\ v_2]$" & "$t_1 = [s_1\ldots s_{i-1} v_1]$" & "$[t_2\ldots t_m] = [v_2\ s_{i+1}\ldots s_n]$".

We define $\lambda' := \{v_1 \mapsto [a_{j+1}\ldots a_h], v_2 \mapsto [b_1\ldots b_l]\} \circ \lambda$ and have

$\lambda' s_i = [a_{j+1}\ldots a_h\, b_1\ldots b_l] = \lambda'[v_1\ v_2]$ and

$\lambda' t_1 = [a_1\ldots a_g\, a_{g+1}\ldots a_h] = \lambda'[s_1\ldots s_{i-1} v_1]$ and

$\lambda'[t_2\ldots t_m] = [b_1\ldots b_l\, c_1\ldots] = [\lambda' v_2\ \lambda' s_{i+1}\ldots\ \lambda' s_n] = \lambda'[v_2\ s_{i+1}\ldots s_n]$.

Thus, λ' unifies $\Gamma'\sigma' = \Gamma\sigma$(splitting).

(That this case can be handled by splitting a variable other than s_1 and t_1 is one of the main reasons for the termination of the unification process.)

Case 2.2.2.3.2: W.l.o.g t_1 is a variable and s_1 is not a variable.

Let $\lambda t_1 =: [a_1\ldots a_h]$. Since $\lambda t_1 \neq \lambda[s_1\ldots s_k]$ for every k, there must be another variable s_i, $i > 1$ such that the situation looks as follows:

$$
\begin{array}{lll}
s & = [s_1\ldots s_{i-1} & s_i \quad s_{i+1}\ldots] \\
 & \quad | & \quad | (v_1) \quad | (v_2) \quad | \\
\lambda s & = [a_1\ldots a_g & a_{g+1}\ldots a_h\, b_1\ldots b_l\ \lambda s_{i+1}\ldots] \quad \text{i.e. } \lambda s_i = [a_{g+1}\ldots a_h\, b_1\ldots b_l] \\
\hline
\lambda t & = [a_1\ldots\ldots\ldots\ldots a_h\, b_1\ldots b_l\, c_1\ldots\,] \\
t & = [t_1 & \qquad\qquad t_2\ldots\,]
\end{array}
$$

Therefore the splitting rule

"$[w\ s\ s] = [t\ t\ v\ t'] \rightarrow v = [v_1\ v_2]$ & $w = [t\ t\ v_1]$ & $[s\ s] = [v_2\ t']$
if s and t exist. v_1 and v_2 are new variables."

is again applicable for splitting s_i.

We split s_i yielding "$s_i = [v_1\ v_2]$" & "$t_1 = [s_1 \ldots s_{i-1} v_1]$" & "$[t_2 \ldots t_m] = [v_2\ s_{i+1} \ldots s_n]$".

We define $\lambda' := \{v_1 \mapsto [a_{g+1} \ldots a_h], v_2 \mapsto [b_1 \ldots b_l]\} \circ \lambda$ and have again

$\lambda' s_i = [a_{g+1} \ldots a_h\ b_1 \ldots b_l] = \lambda' [v_1\ v_2]$ and

$\lambda' t_1 = [a_1 \ldots a_g\ a_{g+1} \ldots a_h] = \lambda' [s_1 \ldots s_{i-1} v_1]$ and

$\lambda' [t_2 \ldots t_m] = [b_1 \ldots b_l\ c_1 \ldots] = [\lambda' v_2\ \lambda' s_{i+1} \ldots\ \lambda' s_n] = \lambda' [v_2\ s_{i+1} \ldots s_n]$.

Thus, λ' unifies $\Gamma'\sigma' = \Gamma\sigma$(splitting). ■

Conclusion We have presented sound, terminating and complete unification algorithms for P-logic terms. Except for the relatively simple cases where the accessibility relation has no special properties or is an equivalence relation, the algorithm is not minimal (see the examples 6.2.3), i.e. it may compute superfluous unifiers. Since the number of unifiers is still finite, a minimal algorithm may be obtained by eliminating the redundant unifiers in an appropriate postprocessing step. However, because in general there may be exponentially many unifiers, there is a need for further investigation into the unification of P-logic anyhow in order to find more restrictions during the generation of redundant unifiers.

Chapter Seven

Modal Resolution

Two different versions of the resolution rule are necessary. Resolution for interpretations with a serial accessibility relation is just like ordinary resolution. The only difference is that the unification may produce more than one, but at most finitely many unifiers. When the accessibility relation is not serial a more complex theory resolution operation is necessary. The two versions are defined in this chapter and their soundness is shown. For the completeness proofs more technical machinery is necessary which will be provided in the next chapters.

7.1 Resolution for Serial Interpretations

There is no significant difference to the resolution rule for predicate logic. For simplicity we incorporate the factoring rule into the resolution rule.

Definition 7.1.1 (The Resolution Rule for Serial Interpretations)
Let $C = Ptl_1 \vee \ldots \vee Ptl_n \vee C'$ and
$D = \neg Psl_1 \vee \ldots \vee \neg Psl_m \vee D'$
be two clauses with no variables in common, the *parent clauses*, and let σ be a prefix-preserving and $\mathfrak{R}$-admissible unifier for the termlists $tl_1,\ldots,tl_n$ and $sl_1,\ldots,sl_m$ of the *resolution literals* $\{Ptl_1,\ldots,Ptl_n\}$ and $\{\neg Psl_1,\ldots,\neg Psl_m\}$ i.e. $\sigma tl_1 = \ldots = \sigma tl_n = \sigma sl_1 = \ldots = \sigma sl_m$.
Then the clause $\sigma C' \vee \sigma D'$ is called a resolvent of the parent clauses C and D. ■

The soundness proof is a special case of the soundness proof for the resolution rule for non-serial interpretations which will be given below.

Examples for resolution operations:

$C = P[va] \vee Q[v]$ Let the accessibility relation be reflexive
$D = \neg P[buw] \vee S[bu]$ $\sigma = \{v \mapsto b, u \mapsto a, w \mapsto []\}$

Resolvent: $Q[b] \vee S[ba]$

$C = P[va] \vee Q[v]$ Let the accessibility relation be transitive
$D = \neg P[buw] \vee S[bu]$ $\sigma = \{v \mapsto [bu], w \mapsto a\}$

Resolvent: $Q[bu] \vee S[bu]$. ■

7.2 Resolution for Non-Serial Interpretations.

The execution of a resolution operation usually consists of two steps:

Step 1: The parent clauses must be instantiated with the unifier.

Step 2: The contradictory literals must be identified and the remaining literals in the instantiated parent clauses must be collected into the resolvent.

Both steps are not without problems when the accessibility relation is not serial. We shall discuss the problems with some examples and then give a solution.

7.2.1 Conditioned Instantiation of Clauses

Consider the simple clause C = P[u]. This clause is satisfiable with a P-interpretation consisting of the initial world only, but the instance $\{u \mapsto a\}P[u] = P[a]$ with a non-variable 'a' is *not* satisfiable in this interpretation. Therefore a straightforward instantiation rule is <u>not</u> sound. The reason is that certain P-interpretations may satisfy a quantified formula just by making the quantification empty. Of course this P-interpretation can no longer satisfy the instantiated formula where the variable in question has been replaced by a non-variable term. Thus, we can only create a conditioned instance of the clause where the condition for the instantiated W-variable u expresses somehow "if there is a world accessible from the world denoted by prefix(u, C) then ..." or after rewriting the implication as a disjunction: "either there is no world accessible from the world denoted by prefix(u, C) or ...". In order to express such conditions as literals we need a special predicate 'End' which takes one world-path p as argument and expresses "The world denoted by p is the last one". The correct instance of the example above is then $\{u \mapsto a\}P[u] = End([]) \vee P[a]$ with the informal meaning: Either there is no world beyond the initial world or there is one and P holds in this world.

Definition 7.2.1 (The 'End' Predicate)
A special predicate symbol 'End' is defined which is distinguished from all other symbols. 'End' takes one world-path (or W-term) as argument. Its semantics is:
For a P-interpretation $\mathfrak{I}_P$ with initial world $\mathfrak{I}_0$:

$\mathfrak{I}_P \Vdash_P End(p)$ iff $\mathfrak{I}_P(p)(\mathfrak{I}_0) \neq \perp$ and there is no world accessible from $\mathfrak{I}_P(p)(\mathfrak{I}_0)$. ■

Definition 7.2.2 (Conditioned Instantiation of Clauses)
Let C = be a clause and let σ be a prefix-preserving substitution.

$$\sigma\downarrow C := \bigcup\{\{End(p), End([pa_1]), \ldots, End([pa_1 \ldots a_n])\} \mid$$
$$p = \sigma(prefix^*(u, C)) \text{ where } u \in DOM(\sigma) \text{ and } \sigma u = [a_1 \ldots a_n a_{n+1}]\} \quad (a)$$
$$\cup \bigcup\{\{End([]), \ldots, End([\ldots b_m])\} \mid [\ldots b_m b_{m+1}] \text{ is a subterm in } COD(\sigma)\} \quad (b)$$
$$\cup \sigma C. \quad (c)$$ ■

Examples for conditioned instantiation of clauses:

$\{v \mapsto [cv_1d], w \mapsto [ew_1w_2], x \mapsto y\}\downarrow P[a(x)vbw] =$

$End([a(y)]) \vee End([a(y)c]) \vee End([a(y)cv_1]) \vee$

$End([a(y)cv_1db]) \vee End([a(y)cv_1dbe]) \vee End([a(y)cv_1dbew_1])$

$\vee P[a(y)cv_1dbew_1w_2]$

$\{x \mapsto f[ab]\}\downarrow Q([]\ x) = End([]) \vee End([a]) \vee Q([]\ f[ab])$

(This looks horribly inefficient, but the number of End-literals can be considerably reduced using the End-reduction rule which is defined below. Furthermore most of the remaining End-literals disappear when the resolvent is generated.)

■

Theorem 7.2.3 (Conditioned Instantiation is Sound)

Let $C = \forall u_1,\ldots,u_k\ C'$ be a fully quantified clause, let $\Im_P$ be a P-model for C with initial world $\Im_0$, let σ be an $\Re$-admissible, idempotent and prefix-preserving substitution and let $D := \forall v_1,\ldots,v_n\ \sigma\downarrow C'$ be the fully quantified conditioned instance of C. Then $\Im_P \Vdash_P D$.

Proof: Since σ is prefix-preserving, D is again an M-adjusted clause.

W.l.o.g let $DOM(\sigma) := \{x_1,\ldots,x_l\} \subseteq \{u_1,\ldots,u_k\}$.

The correspondences between the different variable sets is as follows:

$Vars(C) = \{u_1,\ldots \ldots \ldots, u_k\}$

$DOM(\sigma) = \{x_1,\ldots,x_l\} \quad \subseteq Vars(C)$

$Vars(D) = \{v_1,\ldots,v_n\} \cap DOM(\sigma) = \emptyset$ (since σ is idempotent)

Thus, $Vars(C) \setminus DOM(\sigma) = Vars(C) \cap Vars(D)$.

In order to apply theorem 5.1.5, let $\Im_P' := \Im_P[v_1/c_1 \ldots v_n/c_n]$ be a $\sigma\downarrow C'$-continuing P-interpretation.

We must show that $\Im_P'$ satisfies $\sigma\downarrow C'$.

Case 1: $\Im_P'$ satisfies one of the generated End-literals.

Obviously $\Im_P'$ satisfies $\sigma\downarrow C'$.

Case 2: $\Im_P'$ satisfies not a single of the generated End-literals.

Since none of the literals "$\bigcup\{\{End([]),\ldots, End([\ldots b_m])\} | [\ldots b_m b_{m+1}]$ is a subterm in $COD(\sigma)\}$" (def. 7.2.2,b) is true in $\Im_P'$, obviously $\Im_P'(\sigma x_i) \neq \bot$ for $i = 1,\ldots,l$.

Therefore let $\Im_P'' := \Im_P'[\sigma]$ (see lemma 5.2.5).

Lemma 5.2.5 states in particular

for every D-term t in C': $\Im_P''(t) \neq \bot$ implies $\Im_P''(t) = \Im_P'(\sigma t)$ and

for every world-path p in C': $\Im_P''(p)(\Im_0) \neq \bot$ implies $\Im_P''(p)(\Im_0) = \Im_P'(\sigma p)(\Im_0)$. (✻)

The next thing to be show is that $\Im_P''$ is C'-continuing, i.e.

for every world-path $[p.u] \in C'$: if $\Im_P''(p)(\Im_0) \neq \bot$ then $\Im_P''([p.u])(\Im_0) \neq \bot$: (def. 5.1.4)

Therefore let $[p.u\ldots] \in C'$ with $\Im_P''(p)(\Im_0) \neq \bot$. With (✻) we have $\Im_P'(\sigma p)(\Im_0) \neq \bot$.

Case a) $u \notin Dom(\sigma)$.

Since $\sigma p \in \sigma\downarrow C'$ and $\Im_P'$ is $\sigma\downarrow C'$-continuing, $\bot \neq \Im_P'([\sigma p.u])(\Im_0) = \Im_P''([p.u])(\Im_0)$.

Case b) $u \in Dom(\sigma)$.

$\Rightarrow End(\sigma p) \in \sigma\downarrow C'$ (def. 7.2.2,a)

Since $\Im_P'$ does not satisfy $End(\sigma p)$, and therefore $\Im_P''$ does not satisfy $End(p)$, there is a world accessible from $\Im_P''(p)(\Im_0)$, thus $\Im_P''([p.u])(\Im_0) \neq \bot$.

Now, since $\Im_P''$ is C'-continuing, by theorem 5.1.5, $\Im_P'' \Vdash_P C'$, i.e. $\Im_P'' \Vdash_P L$ for some literal L in C'. Furthermore, with (✻) we get $\Im_P' \Vdash_P \sigma L$, and therefore again $\Im_P' \Vdash_P \sigma\downarrow C'$.

Applying theorem 5.1.5 now, we conclude $\Im_P \Vdash_P D$. ■

Theorem 7.2.4 (The End-Reduction Rule)
A literal End(p) can be removed from a clause C if there is another literal containing a world-path [p.u...] with a W-variable u.
Proof: Clearly no C-continuing P-interpretation $\Im_P$ with initial world $\Im_0$ can satisfy End(p), for otherwise there is no accessible world form $\Im_P(p)(\Im_0)$ such that $\Im_P([p.u])(\Im_0) \neq \bot$. Hence, End(p) is false in all C-continuing P-models for C and can therefore be removed. ■

The next example demonstrates the power of the End-reduction rule:
Instantiation without the End-reduction rule:

$\{v \mapsto [cv_1d], w \mapsto [ew_1w_2], x \mapsto y\}\downarrow P[a(x)vbw] =$
$\quad End([a(y)]) \vee End([a(y)c]) \vee End([a(y)cv_1]) \vee$
$\quad End([a(y)cv_1db]) \vee End([a(y)cv_1dbe]) \vee End([a(y)cv_1dbew_1])$
$\quad \vee P[a(y)cv_1dbew_1w_2]$

Instantiation and application of the the End-reduction rule:

$\{v \mapsto [cv_1d], w \mapsto [ew_1w_2], x \mapsto y\}\downarrow P[a(x)vbw] =$
$\quad End([a(y)]) \vee End([a(y)cv_1]) \vee End([a(y)cv_1db])$
$\quad \vee P[a(y)cv_1dbew_1w_2].$

7.2.2 Complementary Literals

Usually two complementary literals, i.e. literals with opposite sign and the same predicate symbol can be used as resolution literals. They are removed from the resolvent because they are semantically contradictory. We must extend this definition by saying what is complementary to the new 'End' predicate. The semantics of the End-predicate obviously enforces that every literal L containing a world-path [p.a...], $a \neq []$ is complementary to a literal End(p).

Definition 7.2.5 (Complementary Literals)
Two literals L and K are called complementary if either:
- L = Ptl, K = ¬Ptl or tl is a termlist
- L = ±Ptl, K = End(p), [p.a...] ∈ tl for some $a \neq []$ or
- L = End(s), K = End(p), either s = [p.a...] or p = [s.a...] for some $a \neq []$

Two sets L and K of literals are called complementary if every literal in L is complementary to each literal in K. ■

Lemma 7.2.6 Two sets C and D of complementary literals are unsatisfiable by a P-interpretation.

Proof: Assume there is a P-interpretation $\Im_P$ with initial world $\Im_0$ that satisfies a literal $L \in C$ and a literal $K \in D$.

Case 1: L = End(p), i.e. $\Im_P(p)(\Im_0) \neq \perp$ and there is no world accessible from $\Im_P(p)(\Im_0)$.

Case 1.1: K = End([p.a...]).
This contradicts the fact $\Im_P([p.a])(\Im_0) = \perp$.

Case 1.2: K = End(s) and p = [s.a...],
i.e. $\Im_P(s)(\Im_0) \neq \perp$ and there is no world accessible from $\Im_P(s)(\Im_0)$.
This contradicts the fact $\Im_P(p)(\Im_0) \neq \perp$.

Case 1.3: $K = \pm Ptl, [pa...] \in tl$.
Since there is no world accessible from $\Im_P(p)(\Im_0)$, $\Im_P([p.a])(\Im_0) = \perp$
which contradicts $\Im_P \Vdash_P K$.

Case 2: L = Ptl
The case that K is an End-literal is symmetric to case 1.3. The only remaining case is $K = \neg Ptl$. Since $\Im_P$ satisfies L, $\Im_P(t) \neq \perp$ for every term t in tl. Therefore $\Im_P$ cannot satisfy Ptl and $\neg Ptl$.

In all cases we got a contradiction. Thus, $\Im_P$ cannot satisfy both sets C and D. ■

The previous lemma holds for unquantified literals. It does not imply that two quantified literals $\forall u\, P[u]$ and $\forall u \neg P[u]$ are contradictory because the P-interpretation consisting of the initial world only satisfies both of them. The reason is again that the quantification $\forall u$ is empty in this interpretation and in this case the structure of the literal is irrelevant for the truth value of the whole formula. If we therefore assume that both $\forall u\, P[u]$ and $\forall u \neg P[u]$ are true in a P-interpretation, we can deduce that there can't be worlds accessible from the initial world, i.e. End([]) must be true in this P-interpretation. End([]) is therefore implied by $\forall u\, P[u]$ and $\forall u \neg P[u]$ and must be inserted as a *residue* into a resolvent where P[u] and $\neg P[u]$ are used as resolution literals (see [Stickel 85]).

In general such *residue* literals are constructed as follows:

Definition 7.2.7 (The Residue of Literal Sets)
For a literal set C: Residue(C) := {End(p) | p = prefix*(u, C), u ∈ W-vars(C)} ■

Example for residues: Residue({Q[aubcv]}) = {End[a], End[aubc]}. ■

Now we are ready to define the general resolution rule.

Definition 7.2.8 (The General Resolution Rule)

Let $C = C_1 \vee C_2$ and

$D = D_1 \vee D_2$ be two clauses with no variables in common, the *parent clauses*, and let σ be a prefix-preserving and $\mathfrak{R}$-admissible substitution such that

$\sigma\downarrow C = \sigma C_1 \vee C_3 \vee \sigma C_2 \vee C_4$ and

$\sigma\downarrow D = \sigma D_1 \vee D_3 \vee \sigma D_2 \vee D_4$ are the conditioned ("End-reduced") instances

where the End-literals C_3, C_4, D_3 and D_4 are partitioned such that $\sigma C_1 \cup C_3$ and $\sigma D_1 \cup D_3$ are complementary. The clause

$$\text{Residue}(\sigma C_1 \cup C_3 \cup \sigma D_1 \cup D_3) \vee \sigma C_2 \vee C_4 \vee \sigma D_2 \vee D_4$$

is called a resolvent of the parent clauses C and D. ■

It is noted that the substitution σ may be a unifier for literals with the same predicate symbol and different signs, but σ may also unify the world-path p of an End-literal with the leading part q of a world-path [q.a...] occurring in the resolution literals of the second clause. The term a following q in this second world-path should be a non-variable term, for, otherwise the generated resolvent will be subsumed by one of its parent clauses. (It is an exercise for the reader to prove this.)

Examples for resolution operations with End-reduction in non-serial interpretations.

a)	C = P[avv], Q[avv]	
	D = ¬P[wbu], S[wbu]	$\sigma = \{w \mapsto a, v \mapsto b, u \mapsto v\}$
Instantiation:	$\sigma\downarrow C$ = P[abv], End[a], Q[abv]	
	$\sigma\downarrow D$ = ¬P[abv], End[], End[ab], S[abv]	
End-reduction:	$\sigma\downarrow C$ = P[abv], End[a], Q[abv]	
	$\sigma\downarrow D$ = ¬P[abv], End[], S[abv]	Residue = End[ab]
Resolvent:	End[ab], Q[abv], S[abv]	
End-reduction:	Q[abv], S[abv]	
b)	C = P([], x), Q[]	
	D = ¬P([], c[au])	$\sigma = \{x \mapsto c[au]\}$
Instantiation:	$\sigma\downarrow C$ = P([], c[au]), End[], End[a], Q[]	
	$\sigma\downarrow D$ = ¬P([], c[au])	
End-reduction:	$\sigma\downarrow C$ = P([], c[au]), End[], Q[]	
	$\sigma\downarrow D$ = ¬P([], c[au])	Residue = End[a]
Resolvent:	End[a], Q[]	
c)	C = End[vc], Q[v]	
	D = S[abcd]	$\sigma = \{v \mapsto [ab]\}$
Instantiation:	$\sigma\downarrow C$ = End[abc], End[], End[a], Q[ab]	
	$\sigma\downarrow D$ = S[abcd]	
Resolvent:	Q[ab]	■

Theorem 7.2.9 (Soundness of the General Resolution Rule).
Let C and D be two clauses with no variables in common, and let σ be a prefix-preserving and $\Re$-admissible substitution such that $\sigma\downarrow C =: C' \cup C''$ and $\sigma\downarrow D = D' \cup D''$ are the conditioned instances and $E = \text{Residue}(C' \cup D') \cup C'' \cup D''$ is a resolvent.
Then every P-model for the fully quantified clauses C and D is also a P-model for the fully quantified resolvent.

Proof: Let $\Im_P$ be a P-model for the two fully quantified parent clauses C and D. According to theorem 7.2.3, $\Im_P$ is also a P-model for the fully quantified instances $\sigma\downarrow C$ and $\sigma\downarrow D$. Since σ is prefix-preserving, the W-variables in the instantiated parent clauses and the resolvent have the same prefixes. In order to apply theorem 5.1.5, let $\Im_P' := \Im_P[v_1/c_1,\ldots,v_n/c_n]$ be an E-continuing P-interpretation, where w.l.o.g $\{v_1,\ldots,v_n\}$ are the variables occurring in the instantiated parent clauses and the resolvent.

Case 1: $\Im_P'$ is not $\sigma\downarrow C$-continuing.

I.e. there is a variable v with $[p.v] = \text{prefix}(v, \sigma\downarrow C)$ and $\Im_P'(p) \neq \bot$ and $\Im_P'([p.v]) = \bot$.

Since $\Im_P'$ is E-continuing, v cannot occur in C'' and must therefore occur in C'.

In this case, the literal End(p) is part of the residue and satisfies $\Im_P'$. Hence, $\Im_P' \Vdash_P E$.

Case 2: $\Im_P'$ is not $\sigma\downarrow D$-continuing. This case is symmetric to the previous one.

Case 3: $\Im_P'$ is $\sigma\downarrow C \cup \sigma\downarrow D$-continuing.

Since $\Im_P$ satisfies the fully quantified instances $\sigma\downarrow C$ and $\sigma\downarrow D$, by theorem 5.1.4, $\Im_P'$ satisfies $\sigma\downarrow C$ and $\sigma\downarrow D$, i.e. $\Im_P'$ satisfies a literal L in $\sigma\downarrow C$ and a literal K in $\sigma\downarrow D$. If L is in C'' then $\mathcal{K}' \Vdash_P E$ as well. Therefore let $L \in C'$. Since $\Im_P'$ cannot satisfy any of the complementary literals in D' (lemma 7.2.6), K must be an element of $D'' \subseteq E$. Hence, $\Im_P' \Vdash_P E$.

Applying theorem 5.1.5 we can now conclude that $\Im_P$ satisfies the fully quantified resolvent. ■

Chapter Eight

Term Frames

The goal of the three subsequent chapters is to prove the completeness of the resolution rules. In this context completeness means that the empty clause, i.e. 'False', can be deduced from every unsatisfiable clause set by a sequence of resolution operations. The completeness proof we are going to present follows the ideas of the completeness proof for the resolution rule in first order predicate logic (c.f. [Chang&Lee 73]). The first step is to reduce the definition of unsatisfiability, such that not all possible P-frames need be considered to find out whether a clause set is unsatisfiable, but only certain *"term frames"*, or *"T-frames"* for short, whose domain consists of terms. The second step is to represent all possible T-frames in a "semantic tree" and to show that for every unsatisfiable clause set a given semantic tree can be cut below a certain depth such that the remaining finite tree still contains enough information for determining the unsatisfiability of the clause set. This finite tree can then be used to generate the desired sequence of resolution operations terminating with the empty clause.

Before we come to the definition of term frames, however, we define an algebraic relation between P-frames:

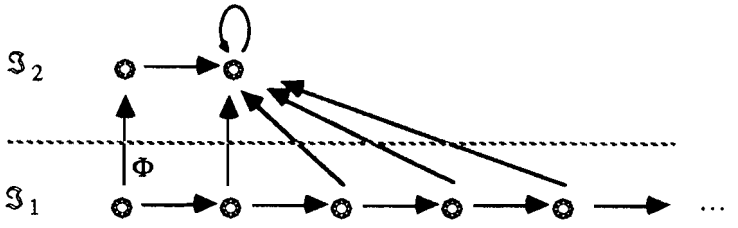
8.1 Frame Homomorphisms

Frame homomorphisms map the worlds in one frame to the worlds in another frame such that the accessibility relations are respected. A frame homomorphism will be used to describe the basic relation between an arbitrary model and a term model for a clause set.

Definition 8.1.1 (Frame Homomorphism)
Given two M-frames $\mathfrak{F}_1 = (\mathbb{D}_1, \mathfrak{I}_1, \mathfrak{R}_1)$ and $\mathfrak{F}_2 = (\mathbb{D}_2, \mathfrak{I}_2, \mathfrak{R}_2)$, over the same signature, a mapping Φ: $\mathfrak{F}_1 \rightarrow \mathfrak{F}_2$ is called a *frame homomorphism* if it maps the worlds in $\mathfrak{I}_1$ to worlds in $\mathfrak{I}_2$ with respect to the accessibility relations, i.e.

a) for every $\mathfrak{I}_1, \mathfrak{I}_2 \in \mathfrak{I}_1$: $\mathfrak{R}_1(\mathfrak{I}_1, \mathfrak{I}_2) \Rightarrow \mathfrak{R}_2(\Phi(\mathfrak{I}_1), \Phi(\mathfrak{I}_2))$ and
b) for every $\mathfrak{I}_1 \in \mathfrak{I}_1$ and $\mathfrak{I}_2' \in \mathfrak{I}_2$: $\mathfrak{R}_2(\Phi(\mathfrak{I}_1), \mathfrak{I}_2') \Rightarrow \exists\, \mathfrak{I}_2 \in \mathfrak{I}_1$: $\mathfrak{I}_2' = \Phi(\mathfrak{I}_2)$ and $\mathfrak{R}_1(\mathfrak{I}_1, \mathfrak{I}_2)$

Example
for a frame
homomorphism



The next lemma states that two P-frames correlated by a frame homomorphism evaluate terms in a similar way.

Lemma 8.1.2

Given two P-frames $\mathfrak{F}_1 = ((\mathbb{D}_1, \mathfrak{I}_1, \mathfrak{R}_1), \mathfrak{I}_{W1})$ and $\mathfrak{F}_2 = ((\mathbb{D}_2, \mathfrak{I}_2, \mathfrak{R}_2), \mathfrak{I}_{W2})$ with a frame homomorphism $\Phi: \mathfrak{F}_1 \rightarrow \mathfrak{F}_2$, for every P-interpretation $\mathfrak{I}_P = (\mathfrak{F}_1, \mathfrak{I}, \emptyset, \emptyset)$:

For every ground term t: $\mathfrak{I}_P(t) \neq \perp$ iff $\Phi(\mathfrak{I}_P)(t) \neq \perp$ and

for every ground world-path t: $\mathfrak{I}_P(t)(\mathfrak{I}) \neq \perp$ iff $\Phi(\mathfrak{I}_P)(t)(\Phi(\mathfrak{I})) \neq \perp$

where $\Phi(\mathfrak{I}_P)$ is defined to be $(\mathfrak{F}_2, \Phi(\mathfrak{I}), \emptyset, \emptyset)$.

Proof: Let $\mathfrak{I}_P = (\mathfrak{F}_1, \mathfrak{I}, \emptyset, \emptyset)$ and let t be a ground term or a ground world-path.

"$\Rightarrow$" Let $\mathfrak{I}_P(t) \neq \perp$ or $\mathfrak{I}_P(t)(\mathfrak{I}) \neq \perp$ respectively.

We perform an induction on the structure of t.

The single base case t = [] is trivial.

The induction steps are:

Case 1: t = [$\mathbb{p}$. s] is a world-path.

The induction hypothesis states $\Phi(\mathfrak{I}_P)(\mathbb{p})(\Phi(\mathfrak{I})) \neq \perp$ and $\Phi(\mathfrak{I}_P)(s) \neq \perp$.

Since $\mathfrak{I}_P([\mathbb{p} . s])(\mathfrak{I}) \neq \perp$, $\mathfrak{R}_1(\mathfrak{I}_P(\mathbb{p})(\mathfrak{I}), \mathfrak{I}_P([\mathbb{p} . s])(\mathfrak{I}))$ must hold and

since Φ is a frame homomorphism, there must be an $\mathfrak{R}_2$-accessible world from $\Phi(\mathfrak{I}_P)(\mathbb{p})(\Phi(\mathfrak{I}))$.

Finally, because world-access functions are maximally defined (def. 3.2.1),

$\Phi(\mathfrak{I}_P)(t)(\Phi(\mathfrak{I})) = \Phi(\mathfrak{I}_P)(\mathbb{p}) \circ \Phi(\mathfrak{I}_P)(s)\ (\Phi(\mathfrak{I})) \neq \perp$.

Case 2: t is a W-term or a D-term. The statement follows immediately from the induction hypothesis.

"$\Leftarrow$" Let $\Phi(\mathfrak{I}_P)(t) \neq \perp$ or $\Phi(\mathfrak{I}_P)(t)(\Phi(\mathfrak{I})) \neq \perp$ respectively.

Again we perform induction on the structure of t.

The base case t = [] is trivial.

The induction steps are

Case 1: t = [$\mathbb{p}$. s] is a world-path.

The induction hypothesis states $\mathfrak{I}_P(\mathbb{p})(\mathfrak{I}) \neq \perp$ and $\mathfrak{I}_P(s) \neq \perp$

Since $\Phi(\mathfrak{I}_P)([\mathbb{p} . s])(\Phi(\mathfrak{I})) \neq \perp$ there is a world accessible from $\Phi(\mathfrak{I}_P)(\mathbb{p})(\Phi(\mathfrak{I}))$, namely

$\mathfrak{I}_2' := \Phi(\mathfrak{I}_P)([\mathbb{p} . s])(\Phi(\mathfrak{I})) = (\Phi(\mathfrak{I}_P)(\mathbb{p}) \circ \Phi(\mathfrak{I}_P)(s))\ (\Phi(\mathfrak{I}))$

$\Rightarrow \exists\, \mathfrak{I}_2 \in \mathfrak{I}_1: \mathfrak{I}_2' = \Phi(\mathfrak{I}_2)$ and $\mathfrak{R}_1(\mathfrak{I}_P(\mathbb{p})(\mathfrak{I}), \mathfrak{I}_2)$ (def. 8.1.2)

$\Rightarrow (\mathfrak{I}_P(\mathbb{p}) \circ \mathfrak{I}_P(s))(\mathfrak{I}) \neq \perp$ (def. 3.2.1, world-access functions are maximally defined)

$\Rightarrow \mathfrak{I}_P(t)(\mathfrak{I}) \neq \perp$.

Case 2: t is a W-term or a D-term. The statement follows immediately from the induction hypothesis. ■

8.2 Construction of Term Frames

T-frames are the analogue to Herbrand interpretations in predicate logic. The name "T-frame" abbreviates "term frame" and suggests that the domain of a T-frame consists of ground D-terms over a given signatur. In the non-serial case there are not necessarily all possible ground D-terms. Function symbols are mapped to term constructor functions. However, since the worlds usually interpret a function symbol differently, there must be different term constructor functions for each world. A function symbol f is therefore mapped in a world $\mathfrak{I}$ to a term constructor function that takes terms $t_1,\ldots,t_n$ and creates a term $f(p, t_1,\ldots,t_n)$ where p is a characteristic term for the world $\mathfrak{I}$. To stay in the P-logic syntax, this p must be a world-path. Thus, we must construct the possible worlds structure in T-frames such that there is a unique correspondence between a ground world-path p and a world $\mathfrak{I}_p$. In the sequel we shall therefore always write worlds in T-frames indexed by their characteristic world-path

Definition 8.2.1 (T-Frames)

A T-frame $\mathfrak{F}_T := ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$ over a signature Σ_p is a special P-frame where

a) The signature interpretations $\mathfrak{I}_p \in \mathfrak{I}$ map the D-valued function symbols f to term constructor functions. $f': t_1,\ldots,t_n \mapsto f(p, t_1,\ldots,t_n)$.

b) $\mathbb{D}$ is the set of ground D-terms containing world-paths which have a corresponding element in $\mathfrak{I}$.

c) The accessibility relation $\mathfrak{R}$ is the corresponding reflexive, symmetric, transitive closure over a basic relation which relates worlds $\mathfrak{I}_p$ with $\mathfrak{I}_{[p.s]}$.

d) $\mathfrak{I}_W$ maps W-valued function symbols g to functions $g': (s_1,\ldots,s_n) \mapsto (\mathfrak{I}_p \mapsto \mathfrak{I}_{[p.g(s1,\ldots,sn)]})$. Since $\mathfrak{I}_W(g)(s_1,\ldots,s_n)$ is always injective, the interpretation of associated inverse function symbols in the symmetric case is straightforward (c.f. def. 3.2.2).

Lemma 8.2.2

A T-frame $\mathfrak{F}_T := ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$ establishes the following correspondence between the accessible worlds $\mathfrak{R}(\mathfrak{I}_r, \mathfrak{I}_s)$ which are indexed with the world-paths r and s and the syntactic structure of r and s:

a) In case $\mathfrak{R}$ has no special properties, except seriality, s = [r . a] where a is a single W-term.

b) In case $\mathfrak{R}$ is reflexive, s = r is in addition possible.

c) In case $\mathfrak{R}$ is symmetric, r = [s . a] where a is a single W-term is in addition possible.

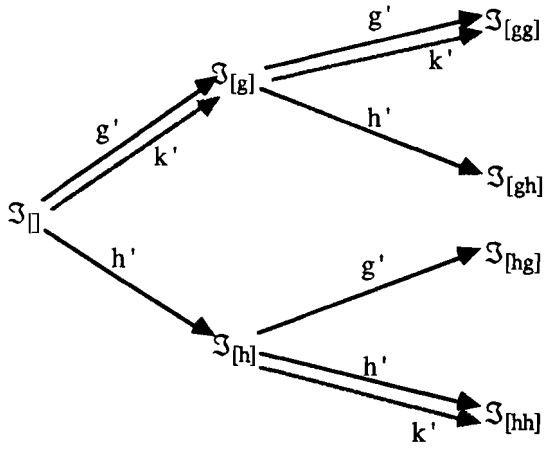
d) In case $\mathfrak{R}$ is transitive, s = [r . p] where p is a non-empty world-path.

Proof: Obvious. ∎

One of the advantages of Herbrand interpretations in predicate logic is that there is no difference between semantic variable assignments $x \mapsto a$ and syntactic ground substitutions. This is so because the domain elements are just ground terms. In the definition of the satisfiability relation $\Vdash$ (for predicate logic), the case handling the universal quantifier can therefore be expressed simply:
For an Herbrand interpretation $\mathfrak{I}_H$: $\mathfrak{I}_H \Vdash \forall x\, \mathcal{F}$ iff $\mathfrak{I}_H[x/a] \Vdash \mathcal{F}$ for every ground term a.
That means the semantics of the quantifier can be expressed using syntactic notions only.

To regain this nice property also in P-logic for term interpretations (P-interpretations built from T-frames) is the subject of the rest of this chapter. There are no problems with quantifications over D-variables because the domain elements in term interpretations are also ground terms, though not necessary all possible ground terms. But what about quantifications over W-variables? Candidates for the syntactic counterpart of the semantic world-access functions are $\mathfrak{R}$-admissible world-paths whose interpretation are actually world-access functions (lemma 5.2.2). But is this enough? Can't there be world-access functions in term interpretations which are not representable as world-paths? In fact, the following example confirms this negative conjecture.

Example for a world-access function in a term interpretation which is not representable as a world-path. Consider the following non-serial term-interpretation for a signature with the two W-valued function symbols g and h:



k' is a world-access function that is not representable as a world-path. ■

Fortunately it can be shown that these additional world-access functions are not necessary for describing the semantics of the class of M-adjusted formulae (def. 3.1.6), we are interested in. The reason is that in our case W-variables have a unique prefix which denotes in each interpretation one particular world. A quantification over a W-variable need therefore range just over the set of worlds which are accessible from this particular world (that is the original semantics of the $\Box$-operator!), and world-access functions that correspond to ground world-paths are sufficient for accessing from a given world all accessible worlds.

Definition 8.2.3 (Term-World-Access Functions)
Given a T-frame $\mathfrak{F}_T := ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$,

let $\mathfrak{I}_{T\to} := \{\mathfrak{I}_W(g_1)(s_{11},\ldots,s_{1n1}) \circ \ldots \circ \mathfrak{I}_W(g_k)(s_{k1},\ldots,s_{knk}) \mid [g_1(s_{11},\ldots,s_{1n1})\ldots g_k(s_{k1},\ldots,s_{knk})]$ is an $\mathfrak{R}$-admissible ground world-path.$\}$

be the set of "term-world-access functions". ■

According to lemma 5.2.2, $\mathfrak{I}_{T\to} \subseteq \mathfrak{I}_{\to}$ and, as the above example demonstrates, $\mathfrak{I}_{T\to} \neq \mathfrak{I}_{\to}$ in general.

Lemma 8.2.4 (Exhaustiveness of Term-World-Access Functions)
Given a T-frame $\mathfrak{F}_T := ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$ and a world $\mathfrak{I}_p \in \mathfrak{I}$, every accessible world $\mathfrak{I}_{p'}$ can be accessed by some term-world-access function.
Proof: For the basic accessibility relation this is a consequence of conditions b and c in the definition of T-frames (def. 8.2.1). When the accessibility relation is reflexive, the $\mathfrak{R}$-admissible empty world path [] which denotes the identity mapping accesses the world itself. When the accessibility relation is symmetric, world-paths built with inverse function symbols denote functions which access the "backwards lying" worlds. Finally when the accessibility relation is transitive, the statement follows by induction on the number of steps which are necessary to access $\mathfrak{I}_{p'}$ from $\mathfrak{I}_p$ in the basic non-transitive accessibility relation. ■

The next lemma states that quantification over world-access functions which correspond to world-paths are sufficient for describing the semantics of W-variables.

Lemma 8.2.5 (A restriction for Quantifications over W-variables)
Given a term interpretation $\mathfrak{I}_T$ with initial world $\mathfrak{I}_r$ and an M-adjusted formula $\forall u\, \mathcal{F}$ where u is a W-variable and p = prefix*(u, $\mathcal{F}$),
$\mathfrak{I}_T \Vdash_P \forall u\, \mathcal{F}$ iff
either $\mathfrak{I}_T(p) = \perp$ and $\mathfrak{I}_T \Vdash_P \mathcal{F}$
or $\mathfrak{I}_T(p) \neq \perp$ and for every $\phi \in \mathfrak{I}_{T\rightarrow}$ with $(\mathfrak{I}_T(p) \circ \phi)(\mathfrak{I}_r) \neq \perp$: $\mathfrak{I}_T[u/\phi] \Vdash_P \mathcal{F}$.
Proof: "⇒" This direction of the proof follows immediately from $\mathfrak{I}_{T\rightarrow} \subseteq \mathfrak{I}_{\rightarrow}$ and def. 3.2.4.
"⇐" The only thing to be proved is that in the or case of the original definition of $\mathfrak{I}_T \Vdash_P \forall u\, \mathcal{F}$ (def. 3.2.4) the quantification "for every $\phi \in \mathfrak{I}_{\rightarrow}$" can indeed be reduced to "for every $\phi \in \mathfrak{I}_{T\rightarrow}$".
Thus, assume $\mathfrak{I}_T(p) \neq \perp$ and let ϕ' be any world-access function with $(\mathfrak{I}_T(p) \circ \phi')(\mathfrak{I}_r) \neq \perp$. Using lemma 8.2.4 we know that the world $(\mathfrak{I}_T(p) \circ \phi')(\mathfrak{I}_r)$ can be accessed by some term-world-access function ϕ, i.e. $(\mathfrak{I}_T(p) \circ \phi')(\mathfrak{I}_r) = (\mathfrak{I}_T(p) \circ \phi)(\mathfrak{I}_r)$. Since all occurrences of u in $\mathcal{F}$ have the same subterm, namely p, and since we know now $\mathfrak{I}_T[u/\phi']([p.u]) = \mathfrak{I}_T[u/\phi]([p.u])$, both term interpretations $\mathfrak{I}_T[u/\phi']$ and $\mathfrak{I}_T[u/\phi]$ evaluate every term in $\mathcal{F}$ to the same value. Thus, since $\mathfrak{I}_T[u/\phi] \Vdash_P \mathcal{F}$, $\mathfrak{I}_T[u/\phi'] \Vdash_P \mathcal{F}$ must hold as well, and finally, with the definition of $\Vdash_P$ we conclude $\mathfrak{I}_T \Vdash_P \forall u\, \mathcal{F}$. ■

Now we are ready to describe the semantics of clauses in term interpretations.

Corollary 8.2.6 (Semantics of Clauses in Term Interpretations)
Let $C = \forall v_1,\ldots,v_n\, C'$ be a fully quantified clause and let $\mathfrak{I}_T$ be a term interpretation for C.
$\mathfrak{I}_T \Vdash_P C$ iff for every C'-continuing term interpretation $\mathfrak{I}_T' := \mathfrak{I}_T[v_1/\mathfrak{I}_T(c_1),\ldots,v_n/\mathfrak{I}_T(c_n)]$ where the c_i are either ground D-terms or $\mathfrak{R}$-admissible ground world-path: $\mathfrak{I}_T' \Vdash_P C'$.
Proof: The proof follows immediately from theorem 5.1.5 and the previous lemma. ■

8.3 The Existence Theorem

The final task to be done in this chapter is to prove that term frames are the right objects to represent P-models for clauses, i.e. to show that there is a term model for each satisfiable clause set. We first define for P-frames associated T-frames which evaluate ground literals to the same truth value as the P-frame and then show that each P-model for a clause set has an associated T-model.

Definition 8.3.1 (Associated T-Frame)

Given a P-frame $\mathfrak{F}_P$ and a world $\mathfrak{I}$ in $\mathfrak{F}_P$, a T-frame $\mathfrak{F}_T$ is said to be $\mathfrak{I}$-associated iff

a) The mapping $\Phi: \mathfrak{F}_T \rightarrow \mathfrak{F}_P : \Phi(\mathfrak{I}_p) = \mathfrak{I}_P(p)(\mathfrak{I})$ is a frame homomorphism, where $\mathfrak{I}_P$ is the special P-interpretation with initial world $\mathfrak{I}$.

b) For every world $\mathfrak{I}_p$ in $\mathfrak{F}_T$: $\mathfrak{I}_p$ and $\Phi(\mathfrak{I}_p)$ assign the same truth values to ground literals ■

Lemma 8.3.2 (Existence of Associated T-Frames)

For every P-frame $\mathfrak{F}_P = ((\mathbb{D}, \mathfrak{I}, \mathfrak{R}), \mathfrak{I}_W)$ and for every world $\mathfrak{I}$ in $\mathfrak{F}_P$ there is an $\mathfrak{I}$–associated T-frame $\mathfrak{F}_T = ((\mathbb{D}_T, \mathfrak{I}_T, \mathfrak{R}_T), \mathfrak{I}_{WT})$.

Proof: We define $\mathfrak{I}_T$ to be the domain of the mapping $\Phi(\mathfrak{I}_p) = \mathfrak{I}_P(p)(\mathfrak{I})$.

Thus, $\mathfrak{I}_P(p)(\mathfrak{I}) \neq \perp$ for every ground world-path occurring in $\mathbb{D}_T$.

We must show that Φ is a frame homomorphism.

Let $\mathfrak{I}_P := (\mathfrak{F}_P, \mathfrak{I}, \emptyset, \emptyset)$ be the special P-interpretation with initial world $\mathfrak{I}$.

a) Let $\mathfrak{I}_r, \mathfrak{I}_s \in \mathfrak{I}_T$ with $\mathfrak{R}_T(\mathfrak{I}_r, \mathfrak{I}_s)$ (def. 8.1.1, a)
$\mathfrak{R}(\Phi(\mathfrak{I}_r), \Phi(\mathfrak{I}_s)) = \mathfrak{R}(\mathfrak{I}_P(r)(\mathfrak{I}), \mathfrak{I}_P(s)(\mathfrak{I}))$
A case analysis according to the properties of $\mathfrak{R}$ and lemma 8.1.2 confirms the last relation $\mathfrak{R}(\mathfrak{I}_P(r)(\mathfrak{I}), \mathfrak{I}_P(s)(\mathfrak{I}))$.

b) Let $\mathfrak{I}_r \in \mathfrak{I}_T$ and $\mathfrak{I}_2' \in \mathfrak{I}$ such that $\mathfrak{R}(\Phi(\mathfrak{I}_r), \mathfrak{I}_2')$ holds.
$\Phi(\mathfrak{I}_r)$ has accessible worlds, therefore there must be a world $\mathfrak{I}_{[r.s]} \in \mathfrak{I}_T$ with
$\Phi(\mathfrak{I}_{[p.s]}) = \mathfrak{I}_P([p.s])(\mathfrak{I}_b) = \mathfrak{I}_2'$ and $\mathfrak{R}_T(\mathfrak{I}_p, \mathfrak{I}_{[p.s]})$ holds.

Thus, Φ is a frame homomorphism. ■

Theorem 8.3.3 (Existence of Term Models)

A set $\mathbf{C}$ of pairwise variable disjoint clauses is satisfiable if and only if it has a term model.

Proof: "⇒" Let $\mathfrak{F}_P$ be a P-model for $\mathbf{C}$ which satisfies $\mathbf{C}$ in the world $\mathfrak{I}$ and let $\mathfrak{F}_T$ be an $\mathfrak{I}$-associated T-frame (def. 8.3.1) where Φ is the corresponding frame homomorphism.

Let $\mathfrak{I}_P := (\mathfrak{F}_P, \mathfrak{I}, \emptyset, \emptyset)$ be the special P-interpretation with initial world $\mathfrak{I} = \Phi(\mathfrak{I}_{[]})$.

The corresponding special T-interpretation is $\mathfrak{I}_T := (\mathfrak{F}_T, \mathfrak{I}_{[]}, \emptyset, \emptyset)$.

When $v_1,\ldots,v_n$ are the variables of $\mathbf{C}$ and $\sigma := \{v_1 \mapsto c_1,\ldots,v_n \mapsto c_n\}$ is a ground substitution, let $\mathfrak{I}_T[\sigma] := \mathfrak{I}_T[v_1/\mathfrak{I}_T(c_1),\ldots,v_n/\mathfrak{I}_T(c_n)]$ be a $\mathbf{C}$-continuing term interpretation with ground D-terms and ground world-paths respectively.

Since the clauses are pairwise variable disjoint, there are no conflicts.

Furthermore, the fact that $\mathfrak{I}_T[\sigma]$ is a term interpretation guarantees $\mathfrak{I}_T(c_i) \neq \perp$ for $i = 1,\ldots,n$.

Lemma 8.1.2 then states $\mathfrak{I}_P(c_i) \neq \perp$ for $i = 1,\ldots,n$.

Therefore we can define the P-interpretation $\mathfrak{I}_P[\sigma] := \mathfrak{I}_P[v_1/\mathfrak{I}_P(c_1),\ldots,v_n/\mathfrak{I}_P(c_n)]$.

We show that $\mathfrak{I}_P[\sigma]$ is $\mathbb{C}$-continuing:

Let p := prefix*(w_j, $\mathbb{C}$) for the W-variable w_j and assume $\mathfrak{I}_P[\sigma](p) \neq \perp$.

With lemma 8.1.2 we know $\mathfrak{I}_T[\sigma](p) \neq \perp$, and since $\mathfrak{I}_T[\sigma]$ is $\mathbb{C}$-continuing, $\mathfrak{I}_T[\sigma]([p.w_i]) \neq \perp$.

Again with lemma 8.1.2 , $\mathfrak{I}_P[\sigma]([p.w_i]) \neq \perp$ must hold.

Thus, $\mathfrak{I}_P[\sigma]$ is $\mathbb{C}$-continuing.

Since $\mathfrak{I}_P$ satisfies $\mathbb{C}$, for every clause $C \in \mathbb{C}$ there is a literal L such that $\mathfrak{I}_P[\sigma] \Vdash_P L$ (theorem 5.1.5).
With lemma 5.2.5, we obtain for every term t in $\mathbb{C}$: $\mathfrak{I}_P[\sigma](t) = \mathfrak{I}_P(\sigma t)$.

Thus, $\mathfrak{I}_P \Vdash_P \sigma L$ (σL is a ground literal.)

$\Rightarrow \mathfrak{I}_T \Vdash_P \sigma L$ (def. 8.3.1, b)

$\Rightarrow \mathfrak{I}_T[\sigma] \Vdash_P L$ (lemma 5.2.5)

Since $\mathfrak{I}_T[\sigma]$ was arbitrarily chosen, corollary 8.2.6 states that $\mathfrak{I}_T$ satisfies each fully quantified clause in $\mathbb{C}$, thus $\mathfrak{F}_T$ is a P-model for $\mathbb{C}$.

" $\Leftarrow$" This direction of the proof is trivial. ■

An obvious consequence of the previous proof is:

Corollary 8.3.4

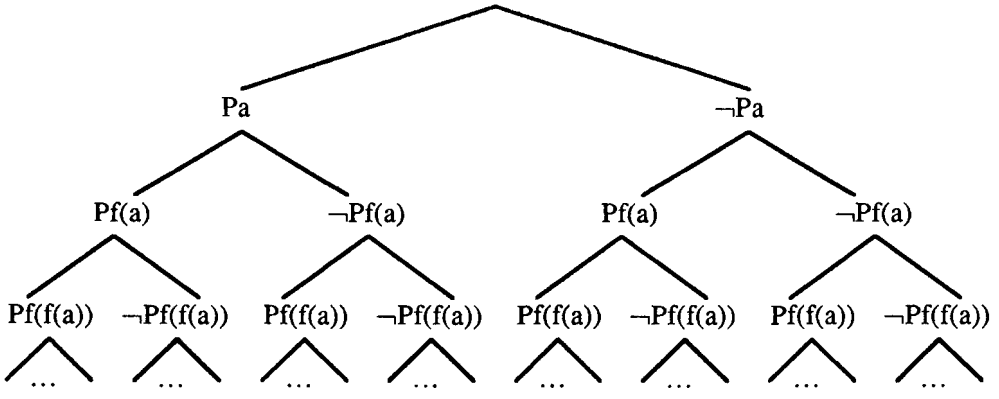
When a formula is satisfiable at all then there is always a T-model satisfying it in the world $\mathfrak{I}_{[]}$. ■

Thus, we need no longer distinguish between a term model for a formula $\mathcal{F}$, i.e. a T-frame satisfying $\mathcal{F}$ in *some* world, and the special term interpretation with initial world $\mathfrak{I}_{[]}$ that satisfies $\mathcal{F}$.

Chapter Nine

Semantic Trees

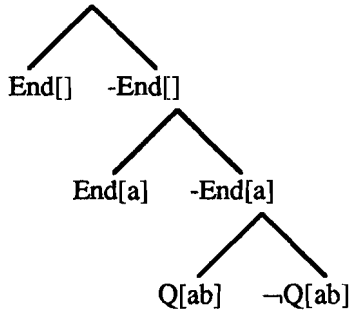
The last step of the preparation for the completeness proof for the resolution rules is to find a representation for the set of *all* term interpretations for a given clause set. When the clause set is unsatisfiable, a finite part of this datastructure should be sufficient to prove its unsatisfiability. A tree structure, called semantic trees, is used in predicate logic for this task. A semantic tree in predicate logic is a (downward) tree T with an unlabeled root node and subtrees where the nodes are labeled with ground atoms as follows: If $N_1,\ldots,N_k$ are the tip nodes of T, then take a new ground atom A and attach k copies of the partial tree consisting of the root node and exactly two descendent nodes labeled with A and $\neg$A to $N_1,\ldots,N_k$. (There are other equivalent definitions.)
As an example consider the semantic tree for a signature consisting of a constant symbol 'a', a one place function symbol f and a one place predicate symbol P. The set of ground atoms built with these symbols is {Pa, Pf(a), Pf(f(a)), ...}. A semantic tree for this signature is:



Clearly the set of labels in each branch in the tree represents an Herbrand interpretation. For instance the rightmost branch represents the Herbrand interpretation assigning False to Pa, Pf(a), Pf(f(a)) etc. The set of all branches gives an exhaustive survey over all Herbrand interpretations.

The definition of semantic trees in P-logic for serial interpretations is exactly the same as the above definition for predicate logic. For non-serial interpretations however, it is necessary to extend the definition for incorporating the information about the "end-worlds", i.e. the worlds where no further worlds are accessible. Therefore we use the special 'End'-predicate (def. 7.2.1) and its negation for representing the possibilities to terminate paths in the world structure. Remember that a literal End(p) is true in a term frame if $\Im_p$ exists, i.e. if there is a world denoted by p, and there is no world accessible from $\Im_p$. We shall further introduce a literal -End(p) (which never occurred so far). It is intended to represent the information that the world structure of the term frame corresponding to the path containing -End(p) in a semantic tree must have a world $\Im_p$ and there must be worlds accessible from $\Im_p$. That means -End(p) is not exactly the negation of End(p).

To illustrate this, consider the partial tree corresponding to the ground atom Q[ab] which looks like:



The branch ending with End[] represents the information that the world structure of the corresponding term frame consists of the initial world only. This interpretation would falsify both Q[ab] and ¬Q[ab], therefore there is no descendent node. The other branch passing -End[] represents the information that the world structure must have at least one further accessible world. Consequently the next layer in the tree represents the two possibilities that either the world structure ends with the world denoted by the path [a] or that it has further accessible worlds. Only the interpretation corresponding to the second possibility can be extended such that either Q[ab] or ¬Q[ab] holds.

The formal definition for semantic trees given below includes the serial case. For the non-serial case some auxiliary notions will be introduced which correspond to partial semantic trees for single worlds-paths and atoms. These partial semantic trees may contain inconsistencies, i.e. paths containing contradictory literals. They are removed in the final definition for semantic trees for a signature.

Definition 9.1 (Semantic Trees)

A *world-path tree* for a world-path $p = [t_1 \ldots t_n]$ is a (downward) tree consisting of an unlabeled root node, two descendent nodes labeled with End([]) and -End([]) respectively, and subtrees which are defined as follows: If N is a tip node labeled with -End($[t_1 \ldots t_k]$) where $k < n-1$ then N has two descendent nodes labeled with End($[t_1 \ldots t_{k+1}]$) and -End($[t_1 \ldots t_{k+1}]$) respectively.

A *world-path tree* for a *set* $P = \{p_1,\ldots,p_m\}$ of world-paths is a tree T constructed as follows:
Let $P' := (p_{1'},\ldots,p_{m'})$ be a permutation of P such that a world-path p in P′ precedes all world-paths in q in P′ containing p as a subterm.
Initialize T with a world-path tree for $p_{1'}$.
For $i' = 2',\ldots,m'$: Extend T by attaching a copy of a world-path tree for $p_{i'}$ at each tip node of T.

A *partial semantic tree* for a ground atom A in a *serial* interpretation is a tree consisting of a root node and two descendent nodes labeled with A and ¬A.
A *partial semantic tree* for a ground atom A in a *non-serial* interpretation is a tree consisting of a world-path tree for the set of world-paths occurring in A where the tip nodes labeled with -End(...) have exactly two descendent nodes labeled with A and ¬A.

A *semantic tree* for a P-signature Σ_p is a (downward) tree consisting of the root node and subtrees which are constructed in three steps:

Step 1: Let $\mathbb{A}$ be the set of ground atoms which can be constructed with symbols from Σ_p. Starting from the empty tree T, select a new atom A from $\mathbb{A}$ and extend T by attaching a copy of a partial world-path tree for A at each tip node of T.

Step 2: Cut each branch B of T at the first occurrence of a label which is complementary to another label at a higher node (closer to the root node) in B.

Step 3: If a node N has only one descendent node M, remove M and attach the descendent node of M below N.

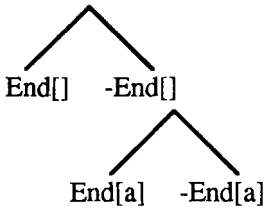
A semantic tree T is said to be *complete* for Σ_p iff each ground atom, built from symbols in Σ_p occurs in the labels of T. (A complete semantic tree can be obtained by performing step 1 in the above definition as long as possible .)

If B is a branch in T, let literals(B) be the set of all labels of the nodes in B.
If N is a tip node in T, let literals(N) be the set of all labels of the nodes in the branch terminating with N.

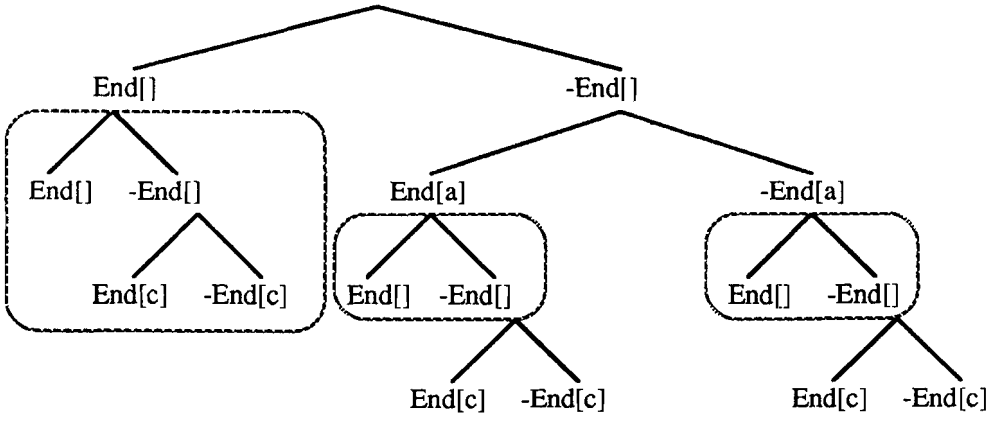
A semantic tree *for a clause set* $\mathbb{C}$ is a semantic tree for the P-signature Σ_p of $\mathbb{C}$.
In case $\mathbb{C}$ contains only End-literals, add an artificial predicate symbol to Σ_p. ■

Examples for semantic trees:

The world-path tree for p = [ab] is:

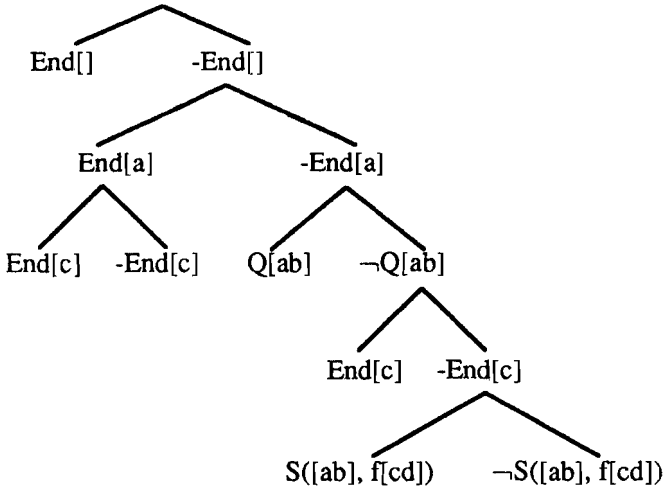


The world-path tree for the set {[ab], [cd]} is:



The marked parts of the tree will be removed at the steps 2 and 3 when this partial tree is integrated into a complete semantic tree.

A reduced partial semantic tree for the two atoms Q[ab] and S([ab], f[cd]) may look like:



■

Lemma 9.2 Each branch B in a semantic tree T for a P-signature Σ_P corresponds to a particular term frame $\mathfrak{F}_T =: \mathfrak{F}_T(B)$ for Σ_P where the possible worlds structure consists only of those worlds $\mathfrak{I}_p$ where -End(p) occurs in B and the relations that are assigned to a predicate symbol Q are defined as follows:

If $Q(p, t_1,\ldots,t_m) \in$ literals(B) then assign 'true' to $Q(t_1,\ldots,t_m)$ in $\mathfrak{I}_p$

If $\neg Q(p, t_1,\ldots,t_m) \in$ literals(B) then assign 'false' to $Q(t_1,\ldots,t_m)$ in $\mathfrak{I}_p$.

The assignment of values to atoms A where neither $A \in$ literals(B) nor $\neg A \in$ literals(B) is left open. $\mathfrak{F}_T(B)$ is said to be a *partial* frame in this case. ■

Lemma 9.3 A *complete* semantic tree T for a P-signature Σ_P corresponds to an exhaustive survey of all possible term interpretations for Σ_P.

Proof: This is mainly a consequence of the fact that all possible ground atoms for Σ_P are used to construct the tree. The world-paths contained in these ground atoms denote all possibilities to construct worlds in a term interpretation. ■

Definition 9.4 (Failure Nodes and Closed Semantic Trees)

Let T be a semantic tree for a clause set $\mathbb{C}$ and let N be one of its nodes.

- $\mathfrak{F}_N$ denotes the partial term frame corresponding to the branch in T which terminates with N.
- N *falsifies* a literal L if $\mathfrak{F}_N$ does not satisfy L and no predecessor node of N falsifies L. Actually N falsifies a literal L if its label is complementary to L.
- N is called a *failure node* iff $\mathfrak{F}_N$ falsifies some ground instance $\sigma\downarrow C$ of a clause C in $\mathbb{C}$ and no predecessor node of N has this property.
- T is called *closed* iff every branch of T terminates with a failure node. ■

Examples for failure nodes and closed semantic trees.

Both clause sets in the examples below are unsatisfiable, therefore there is a finite closed semantic tree.

Example 1:

Clauses:

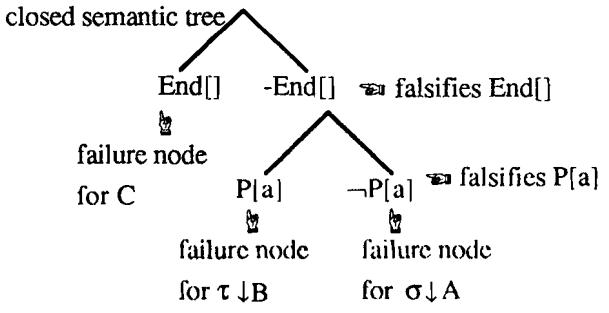
A: $\forall u \quad P[u]$ $\quad \sigma = \{u \mapsto a\}$

B: $\forall v \neg P[v]$ $\quad \tau = \{v \mapsto a\}$

C: $\quad Q[a]$

$\sigma \downarrow A = P[a], End[]$

$\tau \downarrow B = \neg P[a], End[]$



Example 2:

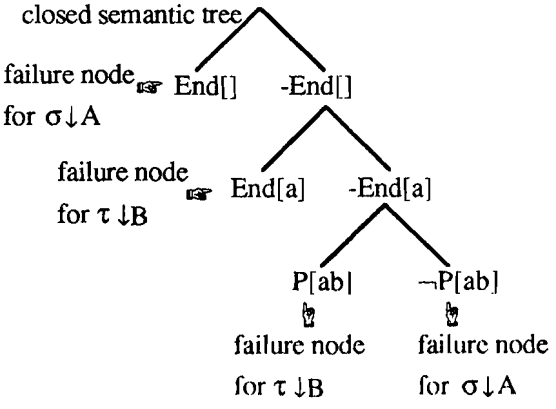
Clauses:

A: $\forall u \quad P[au]$ $\quad \sigma = \{u \mapsto b\}$

B: $\forall v \neg P[vb]$ $\quad \tau = \{v \mapsto a\}$

$\sigma \downarrow A = P[ab], End[a]$

$\tau \downarrow B = \neg P[ab], End[]$



■

The next theorem confirms that for every unsatisfiable clause set there is a closed semantic tree.

Theorem 9.5 A finite set $\mathbb{C}$ of clauses is unsatisfiable if and only if corresponding to every complete semantic tree of $\mathbb{C}$, there is a finite closed semantic tree.

Proof: "$\Rightarrow$" Suppose $\mathbb{C}$ is unsatisfiable. Let T be a complete semantic tree for $\mathbb{C}$. For each branch B of T, let $\mathfrak{F}_T(B)$ be the corresponding term frame. Since $\mathbb{C}$ is unsatisfiable, $\mathfrak{F}_T(B)$ must falsify an instance $\sigma\downarrow C$ of $\mathbb{C}$. However, since $\sigma\downarrow C$ is finite, there must exist a failure node N_B (which is a finite number of links away from the root node) on the branch B. Since every branch of T has a failure node, there is a closed semantic tree T' for $\mathbb{C}$. Furthermore, since each node has only two immediate descendent nodes, T' must be finite.

"$\Leftarrow$" Conversely, if corresponding to every complete semantic tree T for $\mathbb{C}$ there is a finite closed semantic tree, then every branch of T contains a failure node. Because the branches of a complete semantic tree correspond to an exhaustive survey of all possible term frame, this means that every term frame falsifies $\mathbb{C}$. Hence, $\mathbb{C}$ is unsatisfiable. ■

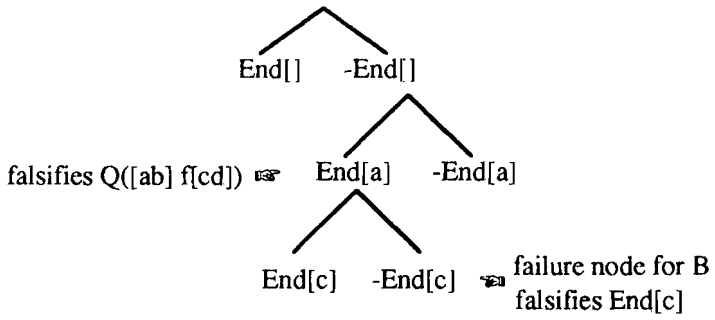
Inference Nodes

Inference nodes in semantic trees for predicate logic and P-logic with serial interpretations are simply nodes whose two immediate descendent nodes are failure nodes for two ground instances of some clauses. Given an unsatisfiable clause set and a corresponding closed semantic tree, an inference node N suggests a (ground) resolution step that is definitely a step forward towards the deduction of the empty clause. The resolution literals for this step are determined just by taking the two literals that are falsified by the two failure nodes below N. Since these two literals are instances of literals in the original non-ground clause set, there is no problem to lift this resolution step to a resolution step between two original clauses. In non-serial interpretations we have the problem that these ground literals may be End-literals that are generated by the conditioned instantiation. They are actually not available for a resolution step with the uninstantiated clauses. The example below shows a situation where a failure node falsifies an End-literal of an instance of a clause which is not contained in the clause itself.

A: $\forall$ u Q([ab] f[cu])

ground instance:

B: Q([ab] f[cd]), End[c]



The trivial solution, adding an instantiation rule, is unacceptable because instantiation rules generate too large search spaces. The other solution is to prove that there is still another resolution possibility that uses only directly instantiated literals. The above example gives a hint where another resolution step might be possible. If the node labeled -End[a] is a failure node, there must be a clause containing a literal End[a] and if this is a direct instance of some literal in the original clause set, it can be used to resolve directly with clause A. In this case the node labeled -End[] would be the inference node. Our definition of inference nodes in serial interpretations coincides therefore with the corresponding definition in predicate logic, whereas for the non-serial case it is more general.

Definition 9.6 Inference Nodes

Given a finite unsatisfiable clause set $\mathbf{C}$ and a corresponding finite closed semantic tree, a node N is called an *inference node* iff

either its two immediate descendent nodes are failure nodes and both falsify some direct instances of literals of clauses in $\mathbf{C}$

or its two immediate descendent nodes N_+ and N_- are labeled End(p) and -End(p) for some p and:
For N_+ there is a clause $\lambda\downarrow C$, falsified by some branch containing N_+ and a literal $K \in C$ containing a world-path $[p'a\ldots]$, $a \neq []$, a is no variable, $p = \lambda p'$ and λK is falsified by N_+.
N_- is a failure node and falsifies a direct instance $\lambda End(p') = End(p) \in \lambda C$ of some $C \in \mathbf{C}$. ■

Before we can prove the existence of an inference node, we need some auxiliary lemmata to throw some light on the circumstances where nodes labeled with End-literals are failure nodes. They concern only non-serial interpretations.

Lemma 9.7 Let T be a closed semantic tree for a clause set $\mathbb{C}$. Let N be a failure node labeled -End(p) and falsifying a ground instance $\lambda\downarrow C$ of a clause $C \in \mathbb{C}$. Then

either N falsifies a literal $\lambda End(p') = End(p) \in \lambda\downarrow C$

or there is a node M in the branch above N, labeled End(q) and a literal $K \in C$ containing a world-path $[q'a\ldots]$, $a \neq []$, a is no variable and $q = \lambda q'$ such that λK is falsified by M.

Proof:

Case 1: N falsifies some direct ground instance L of a literal in C.

L can in principle be a literal End(r) where r is a start sequence of p. If r is a start sequence of p, there is a node labeled -End(r) in the branch above N that falsifies End(r) (def. 9.1). That contradicts the part of the definition for falsifying nodes (def. 9.4) that requires no node in the branch above the current one to falsify the literal. Therefore r = p and $L = \lambda End(p') = End(p)$.

Case 2: N falsifies no direct ground instance of a literal in C.

Since N must falsify at least one literal in $\lambda\downarrow C$, with the same arguments as in case 1, it can be shown that N falsifies a literal $End(p) \in \lambda\downarrow C$. This time, End(p) must have come into $\lambda\downarrow C$ by conditioned instantiation (def. 7.2.2). Therefore there must be another literal $K_\lambda \in \lambda\downarrow C$ containing a world-path $[pc\ldots]$, $c \neq []$ which is a direct instance of a literal $K \in C$, i.e. $K_\lambda = \lambda K$. Since K_λ contains a world-path that has p as a start sequence, K_λ cannot be falsified by a node labeled just with the negation of K_λ because these nodes would occur *below* N. It can also not be falsified by a node labeled End(r) where r is a start sequence of p; there would be two contradictory labels in one branch. Since K_λ must be falsified by some node in the branch of N, the only chance is that there is a node M with a label End(q) and q is a start sequence of some world-path $wp = [qa\ldots]$, $a \neq []$ occurring in K_λ.

Suppose wp is an instance of some world-path $[s\ u\ldots]$ with $u\lambda = [a_1\ldots a_{n+1}]$ such that $[qa\ldots] = [\lambda s\ a_1\ldots a_i]$ for some $i \leq n$. In this case End(q) would be in $\lambda\downarrow C$ (def. 7.2.2,a) which is impossible because the branch of M could not falsify $\lambda\downarrow C$. Using the same argument it can be excluded that wp came into K_λ as a codomain term in λ (using def. 7.2.2,b). The only possibility is now that wp is an instance of a world-path $[q'a\ldots]$ with $\lambda q' = q$ and a is no variable. ■

Lemma 9.8 Let T be a closed semantic tree for a clause set $\mathbb{C}$. Let N be a failure node labeled End(p) and falsifying a ground instance $\lambda\downarrow C$ of a clause $C \in \mathbb{C}$. Then

either there is a literal $L \in C$ containing a world-path $[p'a\ldots]$, $a \neq []$, a is no variable and $p = \lambda p'$ and N falsifies λL.

or there is a node M in the branch above N, labeled End(q) and a literal $K \in C$ containing a world-path $[q'a\ldots]$, $a \neq []$, a is no variable and $q = \lambda q'$ such that λK is falsified by M.

Proof: Let L be a literal in $\lambda\downarrow C$ that is falsified by N, i.e. complementary to End(p). (def. 7.2.5)

Case 1: L = End(r) and r is a start sequence of p.

This is not possible because -End(r) is a label in the branch above N that already falsifies L.

Case 2: L contains a world-path wp such that p is a start sequence of wp, i.e. $wp = [p\ c\ldots]$, $c \neq []$

Case 2.1: L is an instance of a literal in C.

Suppose wp is an instance of some world-path $[s\ u\ldots]$ with $\lambda u = [a_1\ldots a_{n+1}]$ such that $[pc\ldots] = [\lambda s\ a_1\ldots a_i]$ for some $i \leq n$. In this case End(p) would be in $\lambda\downarrow C$ (def. 7.2.2,a) which is impossible because the branch of N could not falsify $\lambda\downarrow C$. Using the same argument it can be excluded that wp came into K_λ as a codomain term in λ (using def. 7.2.2,b). The only possibility is now that wp is an instance of a world-path $[p'c\ldots]$ with

$\lambda p' = p$ and a is no variable. This is just the either case we wanted to prove.

Case 2.2: L is not an instance of a literal in C.

L must be a literal End(wp) that came into $\lambda\downarrow C$ by conditioned instantiation. Therefore there must be another literal $K_\lambda \in \lambda\downarrow C$ containing a world-path $wp = [rb\ldots] = [pc\ldots b\ldots]$, $c \neq []$, $b \neq []$ which is a direct instance of a literal $K \in C$, i.e. $K_\lambda = \lambda K$. If K_λ is falsified by N, we have case 2.1 and we are ready. If K_λ is not falsified by N, it must be falsified by some other node M above N. Proceeding just like in case 2 of lemma 9.7 we find that there is this node M in the branch above N, labeled End(q) and K_λ is the instance of a literal $K \in C$ containing a world-path $[q'a\ldots]$, $a \neq []$, a is no variable and $q = \lambda q'$ that is falsified by M. ■

Theorem 9.9 (Existence of Inference Nodes)

In every finite closed semantic tree T for an unsatisfiable clause set there exists an inference node.

Proof: First of all we notice, that there is at least one node whose immediate descendent nodes are failure nodes, for, if it did not, then every node would have at least one nonfailure descendent. We could then find an infinite branch through T, violating the fact that T is finite.

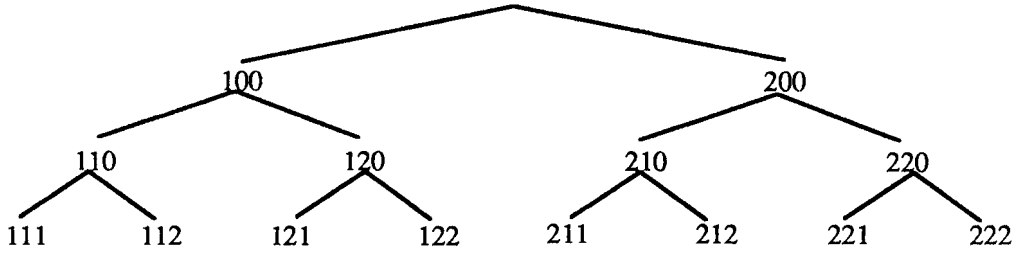
If there is a node whose two immediate descendent nodes both falsify some direct instances of literals in $\mathbb{C}$, we are ready.

Therefore, assume for every node whose two immediate descendent nodes are failure nodes, at least one of them falsifies only End-literals generated by conditioned instantiation. That means all pairs of failure nodes must be labeled with literals End(q) and -End(q) for some world-path q. (*)

Now we must prove that an inference node N exists that satisfies the or-case in definition 9.6, i.e.

its two immediate descendent nodes N_+ and N_- are labeled End(p) and -End(p) for some p and:
For N_+ there is a clause $\lambda\downarrow C$, falsified by some branch containing N_+ and a literal $K \in C$ containing a world-path $[p'a\ldots]$, $a \neq []$, a is no variable, $p = \lambda p'$ and λK is falsified by N_+.
N_- is a failure node and falsifies a direct instance $\lambda End(p') = End(p) \in \lambda C$ of some $C \in \mathbb{C}$.

In order to find this inference node, we define a procedure for searching this node and prove that it must successfully terminate. The procedure moves around the tree, but its motions are always into right neighbour branches. Because T is finite, it must therefore terminate. In order to formalize this, we must give coordinates to the nodes in the tree which increase when moving to right neighbour branches. The coordinates are as follows: If the maximal depth of the tree is n, we take numbers of length n, the first digit is for the first level, the second for the second etc. The digit for the left branch becomes the number 1, the digit for the right branch becomes 2 as the figure below illustrates.



Clearly the numbers increase when moving downwards or to right neighbor branches.

Remember now that our semantic trees have been organized such that a node with label End(p) is placed to the left of the -End(p) node.

The search procedure works as follows:
It starts with a tip node labeled either End(p) or -End(p) that falsifies no direct instance of a literal in $\mathbb{C}$.

Let N be the current tip node that falsifies no direct instance of a literal in $\mathbb{C}$.
Let $n_1 \ldots n_k$ be the coordinate of this node.
According to the lemmata 9.7 and 9.8 there exists a node M, labeled End(q) in the branch above N that falsifies some direct instances of literals in $\mathbb{C}$. The coordinates of M are $n_1 \ldots n_j 0 \ldots 0$, according to its depth in the tree. Since the nodes labeled with End(q) are in the left branches we know that $n_j = 1$.
Consider now its right sibling node M´ labeled -End(q) and with the coordinates $n_1 \ldots n_{j-1} 20 \ldots 0$.

Case 1: M´ is a failure node.

Case 1.1: M´ falsifies some direct instances of literals in $\mathbb{C}$.
In lemma 9.7. it has been proved that this instance is really End(p) and no End-literal with shorter world-path. The common predecessor node is therefore an inference node and the procedure can terminate with success.

Case 1.2: M´ falsifies no direct instance of literals in $\mathbb{C}$.
We move from node N with coordinates $n_1 \ldots n_{j-1} 1 n_{j+1} \ldots n_k$ to node M´ with the greater coordinates $n_1 \ldots n_{j-1} 20 \ldots 0$ and continue the search with M´.

Case 2: M´ is no failure node.
In this case the subtree of M´ must again contain a node whose two immediate descendents are failure nodes. According to our assumption (*) they must be labeled with End(r), -End(r) for some world-path r and at least one of them, say M" does not falsify a direct instance of a literal in $\mathbb{C}$. We continue the search with M" whose coordinates $n_1 \ldots n_{j-1} 2 l_{j+1} \ldots l_k$ are also greater than the coordinates of N.

We have moved from node to node with ever increasing coordinates. Since the tree is finite, the procedure must terminate. Moreover, the only possibility to terminate is the successful case 1.1. ■

Chapter Ten

Completeness of Modal Resolution

"Completeness of the resolution rules" means that for every finite unsatisfiable clause set $\mathbb{C}$ there is a finite sequence of resolution operations terminating with the empty clause. Since there are still some uncertainties concerning variable renamings, we define precisely how this sequence of resolution operations is to be generated.

Definition 10.1 (Resolution Refutation Procedure)
The resolution refutation procedure works as follows:
Given a finite set $\mathbb{C}$ of clauses, select as long as the empty clause is not in $\mathbb{C}$ two variable disjoint copies of clauses in $\mathbb{C}$ which are resolvable with a most general $\mathfrak{R}$-admissible unifier, generate the resolvent E, replace all variables in E by new ones and add the renamed resolvent to $\mathbb{C}$.

More precisely we consider two variable disjoint clauses C and D to be resolvable upon the *resolution literals* $C' \subseteq C$ and $D' \subseteq D$ iff

Case 1: $C' = \{Pt_1, \ldots, Pt_n\}$ and $D' = \{\neg Ps_1, \ldots, \neg Ps_k\}$ for some predicate P and termlists t_i and s_j, and there is a most general $\mathfrak{R}$-admissible unifier σ for $\{t_1, \ldots, t_n\}$, $\{s_1, \ldots, s_k\}$, i.e.
$\sigma t_1 = \ldots = \sigma t_n = \sigma s_1 = \ldots = \sigma s_k$.

Case 2: (non-serial case),
$C' = \{End(p_1), \ldots, End(p_n)\}$ and $D' = \{A \in D \mid [q\ a\ldots] \in A, a \neq [], a \text{ is no variable}\}$ and there is a most general $\mathfrak{R}$-admissible unifier σ for $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots q_k\} := \{q \mid [q\ a\ldots] \in D'\}$, i.e.
$\sigma p_1 = \ldots = \sigma p_n = \sigma q_1 = \ldots = \sigma q_k$. ■

It is noted that this definition allows "self resolution", i.e. resolution with two copies of the same clause. Although there is a strong conjecture that this is not necessary, it cannot be proved with the methods available so far. (A corresponding completeness proof for resolution without self resolution in first-order predicate logic exploits the completeness of hyperresolution [Eisinger 87].)

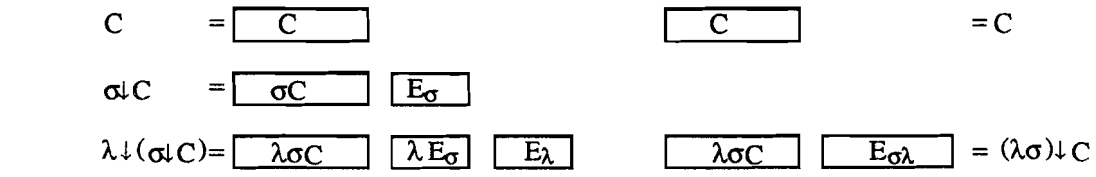
For an implementation the general resolution operation can of course be split into a factoring operation which merges two literals in the same clause and a narrow resolution operation that considers only one resolution literal per clause.

10.1 Preliminaries

There are some auxiliary lemmata necessary for the final completeness proof.

For the ordinary (not conditioned) instantiation operation the equation $\lambda(\sigma C) = (\lambda\sigma)C$ holds because the composition operation for substitutions is just defined in this way. It is not at all obvious and must therefore be proved that the corresponding equation $\lambda\downarrow(\sigma\downarrow C) = (\lambda\sigma)\downarrow C$ holds for conditioned instantiation as well.

Lemma 10.1.1 ("Associativity" of Conditioned Instantiation)
Let C be a prefix-stable clause and let σ and λ be two prefix-preserving substitutions, $\Re$-admissible with non-serial accessibility relations. (That means components $u \mapsto []$ (reflexivity) or $u \mapsto [a^{-1}]$ (symmetry) do not occur.) Then $\lambda\downarrow(\sigma\downarrow C) = (\lambda\sigma)\downarrow C$.
Proof: W.l.o.g we assume $DOM(\sigma) \subseteq Vars(C)$ and $DOM(\lambda) \subseteq Vars(\sigma\downarrow C)$.
The situation is as follows:



E_σ are the End-literals generated by the conditioned instantiation of C with σ.
E_λ are the End-literals generated by the conditioned instantiation of $\sigma\downarrow C$ with λ.
$E_{\sigma\lambda}$ are the End-literals generated by the conditioned instantiation of C with $\lambda\sigma$.
It is obvious that the $\lambda\sigma C$ parts of $\lambda\downarrow(\sigma\downarrow C)$ and $(\lambda\sigma)\downarrow C$ are identical.
Therefore $\lambda E_\sigma \cup E_\lambda = E_{\sigma\lambda}$ remains to be shown.

"$\subseteq$" Let $L =: End(p_\lambda) \in \lambda E_\sigma \cup E_\lambda$.
Case 1: $L \in \lambda E_\sigma$.
$\Rightarrow L = End(\lambda p_\sigma)$ for some world-path p_σ with $End(p_\sigma) \in E_\sigma$.
We consider the two different possibilities to generate End-literals (def. 7.2.2 a and b):
Case 1.1: $\exists K \in C$: $[p\ u\ldots] \in K$ and $\sigma u = [a_1 \ldots a_{n+1}]$ and
$p_\sigma = [\sigma p\ a_1 \ldots a_i]$ for some $i \in \{0,\ldots,n\}$. (def. 7.2.2,a)
$\Rightarrow \lambda\sigma u = [\lambda a_1 \ldots \lambda a_{n+1}]$
$\Rightarrow End[\lambda\sigma p\ \lambda a_1 \ldots \lambda a_i] = End(\lambda p_\sigma) = L \in (\lambda\sigma)\downarrow C$ (def. 7.2.2,a)
Case 1.2: $\exists K \in C$: $x \in K$, x is a variable and $p_\sigma \in \sigma x$. (def. 7.2.2,b)
$\Rightarrow \lambda p_\sigma \in \sigma\lambda x$
$\Rightarrow End(\lambda p_\sigma) = L \in (\lambda\sigma)\downarrow C$ (def. 7.2.2,b)
Case 2: $L \in E_\lambda$
Case 2.1: $\exists K_\sigma \in \sigma\downarrow C$: $[p_\sigma\ v\ldots] \in K_\sigma$ and $\lambda v = [a_1 \ldots a_{n+1}]$ and
$p_\lambda = [\lambda p_\sigma\ a_1 \ldots a_i]$ for some $i \in \{0,\ldots,n\}$. (def. 7.2.2,a)
Case 2.1.1: $v \notin Cod(\sigma)$
$\Rightarrow \exists K \in C$: $[p\ v\ldots] \in K$ and $p_\sigma = \sigma p$
$\Rightarrow End[\sigma\lambda p\ a_1 \ldots a_i] = End[\lambda p_\sigma\ a_1 \ldots a_i] = End(p_\lambda) = L \in (\lambda\sigma)\downarrow C$ (def. 7.2.2,a)
Case 2.1.2: $v \in Cod(\sigma)$
Case 2.1.2.1: $u \mapsto [q\ v\ldots] \in \sigma$ for some W-variable u. (q may be empty.)

$\Rightarrow \exists\, K \in C$: $s = [r\ u\ldots] \in K$, $\sigma s = [\sigma r\ q\ v\ \ldots]$, $[\sigma r\ q] = p_\sigma$.

$\Rightarrow End[\lambda\sigma r\ \lambda q\ a_1\ldots a_i] = End[\lambda(\sigma r\ q)\ a_1\ldots a_i] = End[p_\sigma\ a_1\ldots a_i] = End(p_\lambda) = L \in (\lambda\sigma)\downarrow C$ (def. 7.2.2,a)

Case 2.1.2.2: $x \mapsto t \in \sigma$ for some variable x and $[p_\sigma v\ldots] \in t$.

$\Rightarrow x \mapsto \lambda t \in \lambda\sigma$ and $\exists\, K \in C$: $x \in K$

$\Rightarrow End[\lambda p_\sigma\ a_1\ldots a_i] = End(p_\lambda) = L \in (\lambda\sigma)\downarrow C$ (def. 7.2.2,b)

Case 2.2: $\exists\, K_\sigma \in \sigma\downarrow C$: $x \in K_\sigma$, x is a variable and $[p_\lambda a\ldots] \in \lambda x$, $a \neq []$.

$\Rightarrow [p_\lambda a\ldots] \in COD(\lambda\sigma)$.

$\Rightarrow End(p_\lambda) \in C\downarrow(\lambda\sigma)$.

"$\supseteq$" Let $L =: End(p_{\sigma\lambda}) \in E_{\sigma\lambda}$

Case 1: $\exists\, K \in C$: $[p\ u\ldots] \in K$ and $\lambda\sigma u = [a_1\ldots a_{n+1}]$ and

$p_{\sigma\lambda} = [\lambda\sigma p\ a_1\ldots a_i]$ for some $i \in \{0,\ldots,n\}$. (case 7.2.2,a)

Let $\sigma u =: [b_1\ldots b_{r+1}]$

Case 1.1: $\exists\, j \in \{0,\ldots,r\}$ $\lambda[b_1\ldots b_j] = [a_1\ldots a_i]$

$\Rightarrow End[\sigma p\ b_1\ldots b_j] \in E_\sigma$

$\Rightarrow End[\lambda\sigma p\ a_1\ldots a_i] = End(p_{\sigma\lambda}) = L \in \lambda E_\sigma \subseteq \lambda\downarrow(\sigma\downarrow C)$.

Case 1.2: $\forall\, j \in \{0,\ldots,r\}$ $\lambda[b_1\ldots b_j] \neq [a_1\ldots a_i]$

$\Rightarrow \sigma u = [b_1\ldots b_j\ v\ q]$, $q \neq []$ and $\lambda v = [a_k\ldots a_i]$ such that $[\lambda b_1\ldots\lambda b_j\ a_k\ldots a_i] = [a_1\ldots a_i]$.

$\Rightarrow End[\sigma p\ b_1\ldots b_j v] \in E_\sigma$

$\Rightarrow End[\lambda\sigma p\ \lambda b_1\ldots\lambda b_j\ a_k\ldots a_i] = End[\lambda\sigma p\ a_1\ldots a_i] = End(p_{\sigma\lambda}) = L \in E_\lambda \subseteq \lambda\downarrow(\sigma\downarrow C)$.

Case 2: $x \mapsto t \in \lambda\sigma$ with $[p_{\sigma\lambda}\ a\ldots] \in t$, $a \neq []$ (case 7.2.2,b)

$\Rightarrow \exists\, K \in C$ with $x \in K$.

Case 2.1: $x \notin DOM(\sigma)$

$\Rightarrow x \in \sigma K \in \sigma C$ and $x \mapsto t \in \lambda$

$\Rightarrow End(p_{\sigma\lambda}) \in E_\lambda \subseteq \lambda\downarrow(\sigma\downarrow C)$. (def. 7.2.2,b)

Case 2.2: $x \in DOM(\sigma)$

$\Rightarrow x \mapsto t' \in \sigma$ and $\lambda t' = t$ and $t' \in \sigma K \in \sigma C$.

Case 2.2.1 $\exists\, y \in t'$ with $\lambda p_\sigma = p_{\sigma\lambda}$

$\Rightarrow End(p_{\sigma\lambda}) \in E_\lambda \subseteq \lambda\downarrow(\sigma\downarrow C)$.

Case 2.2.2 $\exists\, [p_\sigma a'] \in t'$ with $\lambda p_\sigma = p_{\sigma\lambda}$

$\Rightarrow End(p_\sigma) \in E_\sigma$ (def. 7.2.2,b)

$\Rightarrow End(\lambda p_\sigma) = End(p_{\sigma\lambda}) = L \in \lambda E_\sigma \subseteq \lambda\downarrow(\sigma\downarrow C)$.

Case 2.2.3 $\exists\, [p_\sigma\ v\ldots] \subset t'$: $\lambda v = [a_1\ldots a_{n+1}]$ such that $[\lambda p_\sigma\ a_1\ldots a_i] = p_{\sigma\lambda}$ for some $i \in \{1,\ldots,n\}$.

$\Rightarrow End[\lambda p_\sigma\ a_1\ldots a_i] = End(p_{\sigma\lambda}) = L \in E_\lambda \subseteq \lambda\downarrow(\sigma\downarrow C)$. ∎

The next lemma establishes a very useful correspondence between conditioned instances of the residue of a literal set and the conditioned instances of these literals themselves.

Lemma 10.1.2 Let C be a prefix-stable literal set and let λ be a ground substitution with $Vars(C) \subseteq DOM(\lambda)$ that is $\Re$-admissible with non-serial accessibility relations. Then $\lambda\downarrow Residue(C) \subseteq \lambda\downarrow C$

Proof: Let $L =: End(p_\lambda) \in \lambda\downarrow Residue(C)$.

Case 1: $L \in \lambda Residue(C)$

$\Rightarrow L = End(\lambda p)$ with $End(p) \in Residue(C)$.

$\Rightarrow q = [p\ u...] \in C$ (def. 7.2.7)

$\Rightarrow End(\lambda p) = L \in \lambda\downarrow C$.

Case 2: $L \notin \lambda Residue(C)$, i.e. L is an End-literal generated by the instantiation with λ.

Case 2.1: $\exists\ End|p\ v\ ...b| \in Residue(C)$ with $\lambda v = |a_1...a_{n+1}|$ and

$p_\lambda = |\lambda p\ a_1...a_i|$ for some $i \in \{0,...,n\}$ (b may be empty.) (def 7.2.2,a)

$\Rightarrow [p\ v\ ...bw...] =: [q\ w...] \in C$

$\Rightarrow End[\lambda p\ a_1...a_i] = End(p_\lambda) \in \lambda\downarrow C$ (def 7.2.2,a)

Case 2.2: $\exists\ x \mapsto t \in \lambda$ where x is a D-variable and $p_\lambda \in t$. (case 7.2.2,b)

$\Rightarrow x \in End(p) \in Residue(C)$ for some world-path p

$\Rightarrow x \in C$

$\Rightarrow p_\lambda \in \lambda\downarrow C$. (def. 7.2.2,b) ■

In first order predicate logic, the "lifting lemma" states that for each resolvent E´ of instances of two clauses there is a resolvent E of the clauses itself with most general unifier that is more general than E. This lemma confirms that whenever there is a resolution deduction of the empty clause with some instances of the original clauses, resolution with most general unifiers does the same job, hence no additional instantiation rule is necessary. The proof of the lifting lemma exploits that every unifier for two terms is an instance of the corresponding most general unifier. Although the original proof assumes the existence of only one most general unifier for two terms (apart from variable renaming) there is no difference in the proof when a complete set of most general unifiers is available. Since this is the case for P-logic (theorem 6.3.4), we prove the lifting lemma only for non-serial accessibility relations where all these complicated things like conditioned instantiation and residues must be considered. Actually a proof for the serial case can be obtained from the proof below by forgetting residues and End-literals.

Lemma 10.1.3 (Lifting Lemma)

Let C and D be two prefix-stable clauses and let σ' be a prefix-preserving substitution such that $\sigma'\downarrow C$ and $D' \subseteq \sigma'\downarrow D$ are ground and the two literal sets $C' \subseteq \sigma'\downarrow C$ and $D' \subseteq \sigma'\downarrow D$ are complementary in one of the following two ways:

Case 1: $C' = \{A_{\sigma'}\}$ where $A_{\sigma'}$ is a normal literal (no End-literal) and $D' = \{\neg A_{\sigma'}\}$.

Case 2: a) $C' = \{End(p_{\sigma'})\}$ and $End(p_{\sigma'})$ is a direct instance of at least one element of C and

b) $D' \subseteq \sigma' D_1 := \{K \mid [p\ a...] \in K, a \neq [], a$ is no variable, $\sigma' p = p_{\sigma'}\} \subseteq D$ and

c) $End(p_{\sigma'}) \notin \sigma'\downarrow D$.

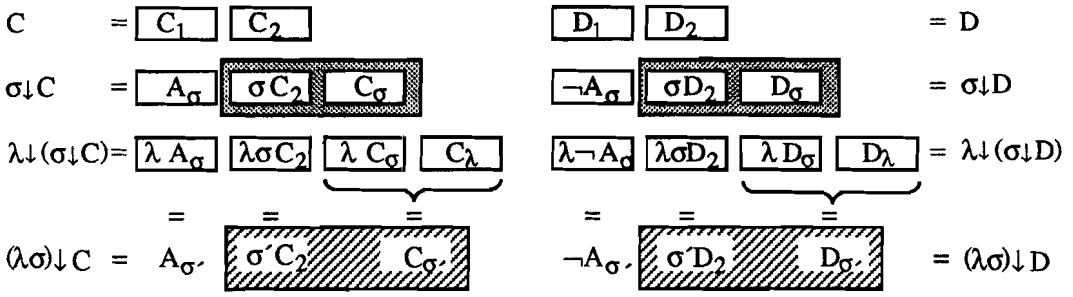
Let $E' = (\sigma'\downarrow C \setminus C') \cup (\sigma'\downarrow D \setminus D')$ be the resolvent of C and D Then there exists a resolvent E of C and D with a most general unifier such that an instance of E is a subset of E´.

Proof: We prove the two cases independently.

Case 1: Since $C' = \{A_{\sigma'}\}$ and $D' = \{\neg A_{\sigma'}\}$ are complementary but no End-literals, there are literals $C_1 = \{A_1,\ldots,A_k\} \subseteq C$ and $D_1 = \{\neg A_1',\ldots,\neg A_l'\} \subseteq D$ and a most general unifier $\sigma \geq \sigma'$ such that $\sigma\{A_1,\ldots,A_k\} = \sigma\{A_1',\ldots,A_l'\} =: \{A_\sigma\}$ (theorem 6.5.1). Let $E := \text{Residue}\{A_\sigma, \neg A_\sigma\} \cup \sigma C_2 \cup C_\sigma \cup \sigma D_2 \cup D_\sigma$ be a resolvent of C and D where $C_2 = C \setminus C_1$, $D_2 = D \setminus D_1$ and C_σ and D_σ are the corresponding End-literals generated by the conditioned instantiation with σ.

We must show that $\lambda\downarrow E = E'$ where $\sigma' =: \lambda\sigma$.

First of all, using the associativity of the conditioned instantiation (lemma 10.1.1), we know $\sigma'\downarrow C = (\lambda\sigma)\downarrow C = \lambda\downarrow(\sigma\downarrow C)$ and $\sigma'\downarrow D = (\lambda\sigma)\downarrow D = \lambda\downarrow(\sigma\downarrow D)$. Therefore the situation is as follows:



+ Residue = Resovent E = Resolvent E'

$C_{\sigma'}$ and $D_{\sigma'}$ are the End-literals generated by the conditioned instantiation of C and D with σ'.

Thus, $\lambda\downarrow E = \lambda\downarrow\text{Residue}\{A_\sigma, \neg A_\sigma\} \cup \lambda\downarrow(\sigma C_2 \cup C_\sigma \cup \sigma D_2 \cup D_\sigma)$

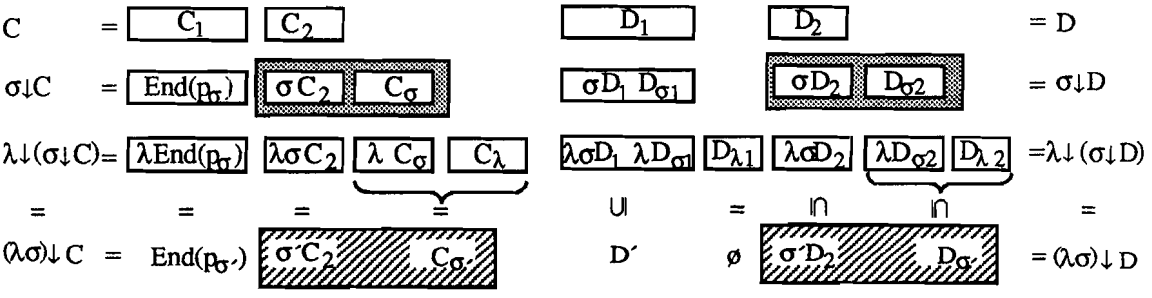
$= \lambda\downarrow\text{Residue}(A_\sigma) \cup \sigma' C_2 \cup C_{\sigma'} \cup \sigma' D_2 \cup D_{\sigma'}$

$= \lambda\downarrow\text{Residue}(A_\sigma) \cup E'$.

$= E'$ (lemma 10.1.2: $\lambda\downarrow\text{Residue}(A_\sigma) \subseteq C_\lambda \subseteq C_{\sigma'}$)

Case 2: Since $\text{End}(p_{\sigma'})$ is a direct instance of a literal in C (condition a) let $C_1 := \{\text{End}(p_1),\ldots,\text{End}(p_k)\} \subseteq C$ with $\sigma' C_1 = \text{End}(p_{\sigma'})$. Since σ' is a unifier for $\{p_1,\ldots,p_k\}$ and $\{p \mid [pa\ldots] \in D_1, \sigma' q = p_{\sigma'}\}$, there is also a most general unifier $\sigma \geq \sigma'$ for these sets of world-paths. Let $\sigma' =: \lambda\sigma$, let $p_\sigma := \sigma p_1 = \ldots = \sigma p_k = \ldots$. Let $C_2 := C \setminus C_1$ and $D_2 := D \setminus D_1$. Let $\sigma\downarrow C := \{\text{End}(p_\sigma)\} \cup \sigma C_2 \cup C_\sigma$ and let $\sigma\downarrow D := \sigma D_1 \cup D_{\sigma 1} \cup \sigma D_2 \cup D_{\sigma 2}$ where C_σ and $D_{\sigma 1} \cup D_{\sigma 2}$ are the End-literals, generated by the conditioned instantiation with σ. $D_{\sigma 1}$ contains those End-literals which are complementary to $\text{End}(p_\sigma)$. It is noted that $\sigma D_2 \cup D_{\sigma 2}$ does not contain any world-path $[qa\ldots]$, $a \neq []$, with $\lambda q = p_{\sigma'}$. All these literals are collected in $\sigma D_1 \cup D_{\sigma 1}$! (❖). Let $E := \text{Residue}(\{\text{End}(p_\sigma)\} \cup \sigma D_1 \cup D_{\sigma 1}) \cup \sigma C_2 \cup C_\sigma \cup \sigma D_2 \cup D_{\sigma 2}$ be a resolvent of C and D. We must show that $\lambda\downarrow E \subseteq E'$.

First of all, using again the associativity of the conditioned instantiation (lemma 10.1.1), we know $\sigma'\downarrow C = (\lambda\sigma)\downarrow C = \lambda\downarrow(\sigma\downarrow C)$ and $\sigma'\downarrow C = (\lambda\sigma)\downarrow D = \lambda\downarrow(\sigma\downarrow D)$. Therefore the situation is as follows:



+ Residue = Resolvent E = Resolvent E'

$D_{\lambda 1} \cup D_{\lambda 2}$ are the End-literals coming from the conditioned instantiation of $\sigma\downarrow D$ by λ. $D_{\lambda 1}$ contains those End-literals in $\sigma D_2 \cup D_{\sigma 2}$ with a world-path $[p_{\sigma'}a\ldots]$, $a \neq []$, as subterm. They would be complementary to $End(p_{\sigma'})$ and therefore they would probably be not be part of E' although they are part of $\lambda\downarrow E$. We must show that this set is empty. The idea is to prove that whenever $D_{\lambda 1}$ is not empty, there must be a literal $End(p_{\sigma'}) \in \sigma'\downarrow D$ which contradicts the assumption c).

Therefore let $L = End(q) \in D_{\lambda 1}$ with $p' := [p_{\sigma'}a\ldots] \in q$.

Since q is a λ-instance of some term t in $\sigma D_2 \cup D_{\sigma 2}$ and because of (❖) there are two possibilities to instantiate t by λ such that $p' \in q = \lambda t$:

Case 1: $[p'a\ldots] \in COD(\lambda)$.

In this case, according to def. 7.2.2,b: $End(p_{\sigma'}) \in \sigma'\downarrow D$. That contradicts assumption c).

Case 2: There is a world-path $[r\ u\ \ldots] \in \sigma D_2 \cup D_{\sigma 2}$ with a W-variable u such that $\lambda u = [a_1\ldots a_n]$ and $[\lambda r\ a_1\ldots a_i] = [p_{\sigma'}a]$ for some $i \in \{1,\ldots,n\}$.

In this case, according to def. 7.2.2,a: $End[\lambda r\ a_1\ldots a_{i-1}] = End(p_{\sigma'}) \in \sigma'\downarrow D$.

That contradicts again assumption d).

Thus, $D_{\lambda 1} = \emptyset$.

Let $Res := Residue(\{End(p_\sigma)\} \cup \sigma D_1 \cup D_{\sigma 1})$

Now, $\lambda\downarrow E = \lambda\downarrow Res \cup \lambda\downarrow(\sigma C_2 \cup C_\sigma \cup \sigma D_2 \cup D_{\sigma 2})$

$\subseteq \lambda\downarrow Res \cup \sigma' C_2 \cup C_{\sigma'} \cup \sigma' D_2 \cup D_{\sigma'}$

$= \lambda\downarrow Res \cup E'$.

According to lemma 10.1.2, $\lambda\downarrow Residue(End(p_\sigma)) \subseteq (\lambda\sigma)\downarrow C$ and $\lambda\downarrow Residue(D_1\sigma \cup D_{\sigma 1}) \subseteq (\lambda\sigma)\downarrow D$.

Clearly $\lambda\downarrow Residue(End(p_\sigma) \neq End(p_{\sigma'})$ and therefore $\lambda\downarrow Residue(End(p_\sigma)) \subseteq E'$.

Because of (❖) and again lemma 10.1.2, $\lambda\downarrow Residue(D_1\sigma \cup D_{\sigma 1})$ does not contain a world-path $[p_{\sigma'}a\ldots]$ that is complementary to $End(p_{\sigma'})$ and therefore $\lambda\downarrow Residue(D_1\sigma \cup D_{\sigma 1}) \subseteq \sigma' D_2 \cup D_{\sigma'} \subseteq E'$.

That proves finally $\lambda\downarrow E \subseteq E'$. ■

10.2 The Completeness Proof

The main idea in the completeness proof presented below is the same as in the corresponding completeness proof for resolution in first order predicate logic: Given an unsatisfiable set $\mathbb{C}$ of clauses, a finite closed semantic tree T for $\mathbb{C}$ is chosen, which exists according to theorem 9.5. The two failure nodes below an inference node N determine two instances $\sigma'\downarrow C$ and $\sigma'\downarrow D$ of clauses in $\mathbb{C}$ and two sets of literals $C' \subseteq \sigma'\downarrow C$ and $D' \subseteq \sigma'\downarrow D$ which are falsified by these failure nodes and by no other node closer to the root node in the branch of N. We can show that there is a most general unifier $\sigma \geq \sigma'$ for some literals in C and D such that C and D are resolvable and there is an instance of the resolvent containing no literal of C' and D'. Therefore the semantic tree T can be cut at least below the inference node N yielding a smaller tree T' with a new inference node etc. This process terminates with the empty clause after finitely many steps.

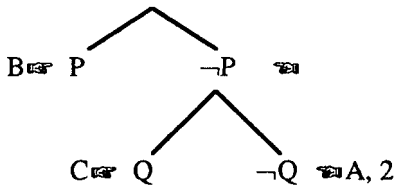
Note that we can only show the existence of a sequence of resolutions that terminates with the empty clause. Since the proof for the existence of a finite closed semantic tree is not constructive, it cannot be used in an actual implementation to guide a resolution based theorem prover. To find the right resolution sequence therefore, as always, requires search.

We illustrate the procedure that will be used in the completeness proof with a few examples.
In the figures below the inference node in the semantic trees are marked with "☞" whereas the failure nodes are marked with D,n ☞ denoting the n´th literal in the clause D which is falsified by the label of this node and by no other node above this one.

Example 1: The first example is a simple propositional one.

Clauses:
A) P[], Q[]
B) ¬P[]
C) ¬Q[]

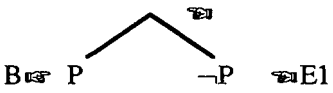
original closed semantic tree:



The inference node suggests a resolution with A and C.

1. Resolution:
A,2 & C → E1: P[]

reduced semantic tree:

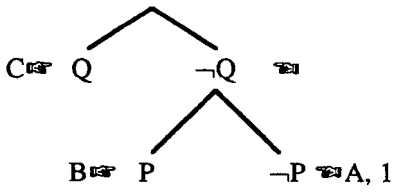


The next inference node suggests the final resolution yielding the empty clause.
B & E1 → E2: ↯

For the same clause set there is another closed semantic tree which generates the second possible deduction of the empty clause:

Clauses:
A) P[], Q[]
B) ¬P[]
C) ¬Q[]

another closed semantic tree:



1. Resolution: B & A,1 → E1: Q[]
2. Resolution: E1 & C → E2: ↯.

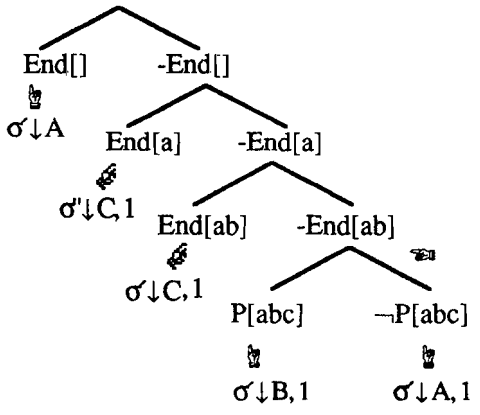
Example 2: The second example involves conditioned instantiation and the generation of a residue. Assume transitivity and non-seriality of the accessibility relation.

Clauses:
A) P[auw]
B) ¬P[vbz]
C) Q[xc]

σ´ = {u ↦ b, v ↦ a, w ↦ c, z ↦ c, x ↦ [ab] }
σ" = {x ↦ a }

σ´↓A = P[abc], End[a], End[ab]
σ´↓B = ¬P[abc], End[], End[ab]
σ´↓C = Q[abc], End[], End[a]
σ"↓C = Q[ac], End[]

a closed semantic tree:



(Note that we joined some individual substitutions for variable disjoint clauses into σ´).

The inference node suggests resolution with A and B, which is in fact the only possible one. The ground resolvent would be E = {End[], End[a], End[ab]} and actually three more ground resolutions of this style would be necessary to deduce the empty clause. Resolution with most general unifiers and the usage of more complementary literals than the failure nodes suggest, shortens the proof:
Unification of P[auw] and P[vbz] yields $\sigma = \{u \mapsto b, v \mapsto a, z \mapsto w\}$. Instantiation with σ yields:

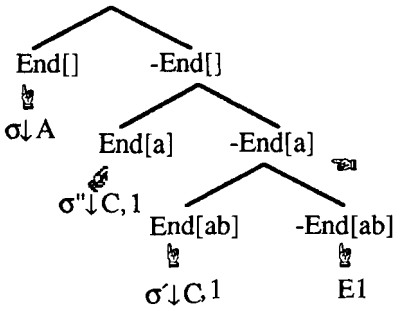
$\sigma\downarrow A$ = P[abw], End[a]

$\sigma\downarrow B = \neg$P[abw], End[].

All literals are complementary, however the residue is End[ab], therefore the resolvent is:

E1 = End[ab].

The corresponding reduced semantic tree is:



According to the new inference node, the literals E1 = End[ab] and C = Q[xc] must be made complementary. Therefore we unify the world-paths [ab] and [x]. The unifier is $\tau = \{x \mapsto [ab]\}$. The instances are:

$\tau\downarrow$E1 = End[ab]

$\tau\downarrow$C = Q[abc], End[], End[a]

All literals are complementary, thus the resolvent is empty and the corresponding semantic tree is also empty.

Example 3: The last example shows what happens when the two failure nodes below an inference node are not tip nodes.

Clauses:

A) Q([ab] f[cu])

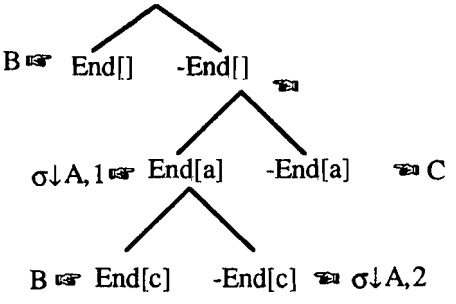
B) R[cd]

C) End[a] $\sigma = \{u \mapsto c\}$

$\sigma\downarrow A$ = Q([ab] f[cd]), End[c]

B = R[cd]

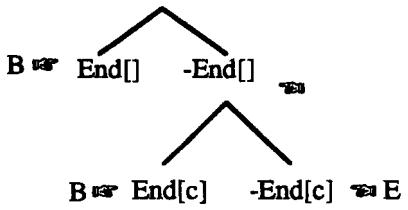
C = End[a]



Resolution between the literals suggested by the lowest failure nodes is not possible. Therefore the inference node labeled -End[] suggest resolution between A and C. The resolvent consists of the residue only: E = End[c].

Both nodes End[a] and -End[a] can now be removed from the tree. (In general more than one resolution is necessary before the tree can be shortened.) The situation is then:

Relevant Clauses:
B) R[cd]
E) End[c]

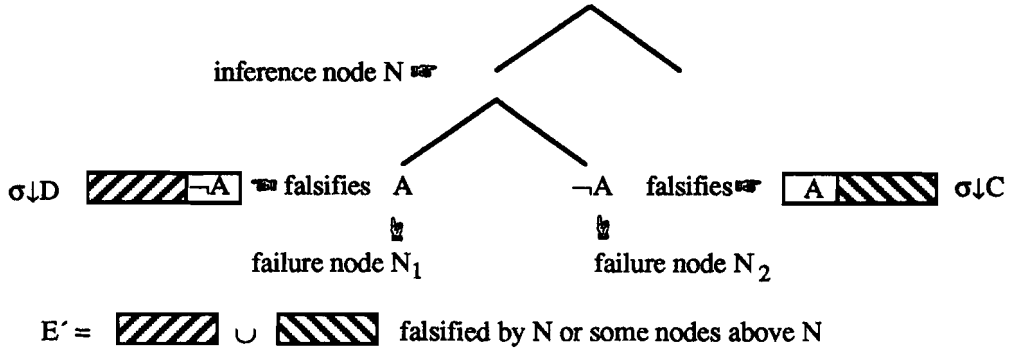


Resolution with B and E yields the empty clause. ■

The proof of the completeness theorem below is mainly a proof for non-serial interpretations. A corresponding proof for serial interpretations can be obtained just by ignoring the cases dealing with the End-predicate.

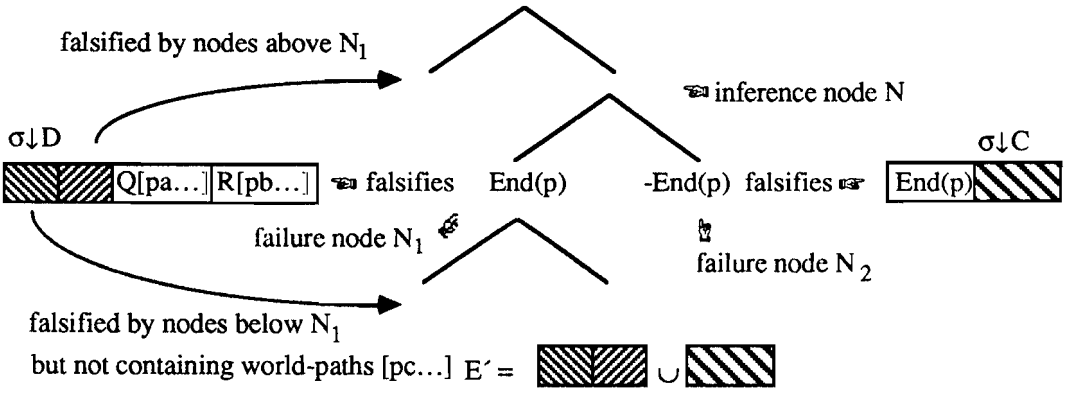
Theorem 10.2 (Completeness of the Resolution Refutation Procedure)
A finite set C of clauses is unsatisfiable if and only if the resolution refutation procedure deduces the empty clause after finitely many steps.
Proof: "⇒" Suppose C is unsatisfiable. Let T be a finite closed semantic tree for C which must exist according to theorem 9.5. If T consists only of the root node, then the empty clause must be in C, for no other clause can be falsified at the root node of a semantic tree. In this case the theorem is obviously true. Assume T consists of more than one node. According to theorem 9.9, T has at least one inference node N. Let N_1 and N_2 be the immediate descendent nodes of N. According to def. 9.6 we must distinguish two different cases.

Case 1: N_1 and N_2 are both failure nodes.
In this case we assume that N_1 and N_2 are labeled with ordinary literals A and ¬A, not with End-literals The End-literal case will be a trivial subcase of case 2. This case 1 is the usual "predicate logic case" and the only one that can occur when the accessibility relation is serial.
The situation around the inference node N can be visualized as follows:



The two failure nodes falsify the literals ¬A and A of two ground instances σ↓D and σ↓C of two variable disjoint clauses C and D in C (we can choose σ to be the same substitution for both clauses because they are variable disjoint. If C and D are not variable disjoint, we make copies of them) The literals in the resolvent $E' = \sigma\downarrow D \cup \sigma\downarrow C \setminus \{A, \neg A\}$ are all falsified by some nodes above N_1 and N_2. (The residue is empty because A is ground.) Now we can apply the lifting lemma 10.1.3, case 1 which states that there is a resolvent E between C and D such that E' is a ground instance of E. Putting this resolvent into C we obtain a unsatisfiable clause set $C \cup \{E\}$ with a corresponding closed semantic tree T' that is obtained by cutting the branches of T below the first node that falsifies E', at least below N. Clearly T' is smaller than T.

Case 2: Now we consider the case that the inference node N has two immediate descendent nodes N_1 and N_2 labeled End(p) and -End(p). N_2 is a failure node and N_1 may or may not have further descendent nodes. According to def. 9.6 and the lemmata 9.7 and 9.8, for N_1 there is at least one ground clause $\sigma\downarrow D$ falsified by the branch of N_1 such that D contains literals D_1 with world-paths $[p'a\ldots]$, $a \neq []$, a is no variable and $p = \sigma p'$, i.e. N_1 falsifies $\sigma K \in \sigma\downarrow D$ for each $K \in D_1$. Let $\mathbb{D}$ be the set of all these ground clauses. Clearly $\mathbb{D}$ is finite. N_2 falsifies a literal $End(p) = \sigma End(p') \in \sigma\downarrow C$ where $\sigma\downarrow C$ is a ground instance of a clause $C \in \mathbb{C}$. For a particular element of $\mathbb{D}$ the situation is as follows:



Obviously $\sigma\downarrow D$ cannot contain End(p) because it is falsified by the branch of N_1. Since the two literal sets that are falsified by the two nodes are complementary, a resolvent E' is possible that does not contain End(p) and no other literal with a world-path [pc...], $c \neq []$ that could be falsified by N_1. Therefore the failure node of $\sigma\downarrow D$ falsifies E' as well. Applying the lifting lemma 10.1.3, case 2 we get again a resolvent E between C and D such that a ground instance of E is a subset of E'. We put E into $\mathbb{C}$. In this way we resolve all elements of $\mathbb{D}$ with C getting finitely many new resolvents such that all failure nodes that falsified elements of $\mathbb{D}$ now either themselves or some of their predecessor nodes falsify the corresponding instances of the new resolvents. None of the resolvents has a literal that is falsified by N_1. For this new clause set we can obtain a new closed semantic tree T' from T by removing at least N_2 and lifting N_1 at the place of N. Clearly T' is again smaller than T.

The whole process is repeated until the closed semantic tree that consists of the root node only is generated. This is possible only when the empty clause is derived. Therefore, there is a deduction of the empty clause from $\mathbb{C}$.

"$\Leftarrow$" Conversely, suppose there is a deduction of the empty clause from $\mathbb{C}$. Let $E_1,\ldots,E_k$ be the resolvents in the deduction. Assume $\mathbb{C}$ is satisfiable. Then there is a P-model $\mathfrak{F}_P$ of $\mathbb{C}$. If a model satisfies the parentclauses, it satisfies the resolvent as well (theorem 7.2.9). Therefore $\mathfrak{F}_P$ satisfies $E_1,\ldots,E_k$. However, this is impossible because one of these resolvents is the empty clause. Hence, $\mathbb{C}$ must be unsatisfiable. ■

Our final theorem gathers all the results of this work and defines a semidecision procedure for the modal logics we have considered.

Last Theorem The following procedure is a semidecision procedure for first order modal logics with the two operators $\Box$ and $\Diamond$, constant-domain possible worlds semantics and an accessibility relation that may have no special properties or it may have any combination of the following properties: reflexivity, symmetry, transitivity, seriality.

Input: A modal logic formula $\mathcal{F}$ and an accessibility relation type $\mathcal{R}$,
Output: A resolution proof if the formula is a tautology in the corresponding logic.
If the formula is no tautology, the procedure may not terminate.

Step 1: Negate $\mathcal{F}$ in order to perform a refutation proof. Set $\mathcal{F} := \neg\mathcal{F}$.
Step 2: If $\mathcal{R}$ denotes an equivalence relation (S5), generate the modal degree 1 normal form for $\mathcal{F}$.
If $\mathcal{R}$ = {symmetric, non-serial} or $\mathcal{R}$ = {symmetric, transitive, non-serial},
split the problem into the predicate logic version and the serial modal logic version.
(See chapter 2.2). Both problems must be solved.
In the sequel we consider only the modal logic version.
Step 3: Eliminate implication and equivalence signs and transform $\mathcal{F}$ into negation normal form.
Step 4: Translate $\mathcal{F}$ into the P-logic syntax $\Pi(\mathcal{F})$.
Step 5: Generate the conjunctive normal form for $\Pi(\mathcal{F})$.
Step 6: Apply the resolution refutation procedure to the clauses.

Proof:

1. Completeness Suppose $\mathcal{F}$ is a tautology.

Step 1: The negated $\mathcal{F}$ is unsatisfiable.
Step 2: If $\mathcal{R}$ denotes an equivalence relation, each formula is equivalent to a formula with modal degree 1. A proof can for instance be found in [Fitting 83], proposition 13.1.
If $\mathcal{R}$ denotes a symmetric accessibility relation, either the initial world is the only one, that is the predicate logic case, or the relation is serial. In this case a symmetric and transitive relation is an equivalence relation. Thus, $\mathcal{F}$ must be unsatisfiable in both classes of models.
Step 3: The rules for transforming a formula into negation normal form preserves the equivalence, i.e. the normalized formula is still unsatisfiable.
Step 4: Corollary 4.3.4 confirms that $\Pi(\mathcal{F})$ is unsatisfiable.
Step 5: Theorem 5.1.3 confirms that the conjunctive normal form is unsatisfiable.
Step 6: The completeness theorem 10.2 confirms that the resolution refutation procedure terminates with the empty clause. The sequence of resolution operations represents a proof for $\mathcal{F}$.

2. Soundness Suppose $\mathcal{F}$ is not a tautology.

Step 1: The negated $\mathcal{F}$ is satisfiable.
Step 2: If $\mathcal{R}$ denotes an equivalence relation, each formula is equivalent to a formula with modal degree 1. $\mathcal{F}$ must also be satisfiable in this case.
If $\mathcal{R}$ denotes a symmetric accessibility relation, $\mathcal{F}$ must either be satisfied by a predicate logic model or by a serial model. In the first case, the soundness of predicate logic deduction calculi ensures that a proof fails. In the second case, $\mathcal{F}$ is satisfiable in a serial

model.

Step 3: The rules for transforming a formula into negation normal form preserve the equivalence, i.e. the normalized formula is still satisfiable.

Step 4: Corollary 4.3.4 confirms that $\Pi(\mathcal{F})$ is satisfiable.

Step 5: Theorem 5.1.3 confirms that the conjunctive normal form $\mathbb{C}$ is satisfiable.

Step 6: The soundness theorem 7.2.9 confirms that the model for $\mathbb{C}$ satisfies the resolvents. The empty clause can therefore never be deduced. The semidecision procedure fails. ■

A **last example** shall illustrate the whole procedure. The example proves that in the modal system K (non-serial accessibility relation) Löb's Axioms $\Box(\Box\mathcal{G} \Rightarrow \mathcal{G}) \Rightarrow \Box\mathcal{G}$ imply the formula $\Box P \Rightarrow \Box\Box P$ that characterizes transitive accessibility relations. Löb's Axioms axiomatize the modal system G (which we have not considered so far) that has a transitive and non-serial accessibility relation $\Re$ with no infinite $\Re$-chains. Let $\mathcal{G} := P \wedge \Box P$. The theorem to be proved is

$$\mathcal{F} := (\Box(\Box(P \wedge \Box P) \Rightarrow (P \wedge \Box P)) \Rightarrow \Box(P \wedge \Box P)) \Rightarrow (\Box P \Rightarrow \Box\Box P).$$

We apply our new semidecision procedure to $\mathcal{F}$. The accessibility relation type is $\mathcal{R}$= non-serial.

Step 1: Negation of $\mathcal{F}$ yields: $\neg((\Box(\Box(P \wedge \Box P) \Rightarrow (P \wedge \Box P)) \Rightarrow \Box(P \wedge \Box P)) \Rightarrow (\Box P \Rightarrow \Box\Box P))$.

Step 2: is skipped

Step 3: $\neg((\Box(\Box(P \wedge \Box P) \Rightarrow (P \wedge \Box P)) \Rightarrow \Box(P \wedge \Box P)) \Rightarrow (\Box P \Rightarrow \Box\Box P))$

$\rightarrow$ ...

$\rightarrow$ $(\Diamond(\Box(P \wedge \Box P) \wedge (\neg P \vee \Diamond\neg P)) \vee \Box(P \wedge \Box P)) \wedge (\Box P \wedge \Diamond\Diamond\neg P))$

Step 4: Translation into P-logic syntax:

$((\forall u\ (P[au] \wedge \forall v\ P[auv]) \wedge (\neg P[a] \vee \neg P[ab])) \vee \forall w(P[w] \wedge \forall x\ P[wx])) \wedge (\forall y\ P[y] \wedge \neg P[cd])$

Step 5: Conjunctive normal form (clause notation):

C1: P[au], P[w] C2: P[au], P[wx] C3: P[auv], P[w]

C4: P[auv], P[wx] C5: $\neg$P[a], $\neg$P[ab], P[w] C6: $\neg$P[a], $\neg$P[ab], P[wx]

C7: P[y] C8: $\neg$P[cd].

Step 6: Resolution refutation procedure:

1. Resolution between C6, literal 1 and C7 with unifier $\sigma = \{y \mapsto a\}$
   Instantiation of C7: P[a], End[]
   The two literal sets $\{\neg P[a]\}$ and {P[a], End[]} are complementary. The resolvent R1 is:

R1: $\neg$P[ab], P[w'x'].

2. Resolution between R1, literal 2 and C8 with unifier $\sigma = \{w' \mapsto c, x' \mapsto d\}$
   Instantiation of R1: $\neg$P[ab], P[cd], End[], End[c]
   The two sets $\{\neg P[cd]\}$ and {P[cd], End[], End[c]} are complementary. The resolvent is:

R2: $\neg$P[ab].

3. Resolution between R2 and C2, literal 2 with unifier: $\sigma = \{u \mapsto b\}$
   Instantiation of C2: P[ab], End[a], P[wx]
   The two literal sets {P[ab], End[a]} and $\{\neg P[ab]\}$ are complementary. The resolvent is:

R3: P[wx].

4. Resolution between R3 and C8 with unifier $\sigma = \{w \mapsto c, x \mapsto d\}$.
   Instantiation of R3: P[cd], End[], End[c].
   The two literal sets {P[cd], End[], End[c]} and $\{\neg P[cd]\}$ are complementary.

The resolvent is empty.

The procedure terminates successfully and actually no End-literal occurred in the resolvents.

Chapter Eleven

Conclusion

A clause based resolution calculus has been developed for a class of first-order modal logics including those with non-serial accessibility relation. The two most significant advantages of this calculus are:

- Instantiation and inference across modal operators can be controlled by a uniform and deterministic unification algorithm. The extensive search space generated by the usual instantiation rules and operator shifting rules in tableau based systems for instance is eliminated. Of course the inherent complexity of the underlying logical problem does not vanish: it may surface again in the many different unifiers that have to be computed. However here they can be much better controlled than with an undiscriminating set of inference rules.
- The method fits into the paradigm of the predicate logic resolution principle. Therefore it is no longer necessary to write specialized theorem provers for modal logics and only slight modifications of existing predicate logic resolution based theorem provers are sufficient. That means that most of the sophisticated implementation and search control techniques, for instance the connection graph idea [Kowalski 75, Eisinger 86], which have been developed for predicate logic can immediately be applied to modal logic as well. This is an indirect advantage which, however, should not be underestimated because it makes more than twenty five years of experience with the resolution principle available to modal logic theorem proving.

Since there are many extensions to the modal systems considered in this work, let us briefly recapitulate the whole procedure and point out which part depends on which assumption in order to gain a feeling for the limitations of the general ideas and about possibilities for extending the methods to other modal and temporal logics.

The first main step is the transformation of a modal logic formula into negation normal form.
This step depended on two assumptions about the semantics of the logical connectives:
1. Every binary logical operator, such as the implication or equivalence sign can be represented with $\wedge$, $\vee$ and the negation sign. The final conjunctive normal form of the translated P-logic formula can be generated with these operators only. If non-clausal resolution would be considered, it might be possible to relax this requirement.
2. To be able to move negation signs in front of the atoms, the negation of every operator and quantifier must be known. Moving negations into formulae is necessary because the negation normal form determines the final status of the variables, existential or universal. This gives the information where instantiation is allowed and where not.

The next main step is the translation of a modal logic formula into P-logic.
The restriction to constant-domain interpretations becomes obvious in this step because the domain variables lose the information about their modal context. On the other hand the definition of P-logic itself, the translation function and the soundness and completeness proof, do not depend on the properties of the accessibility relation.

Generation of conjunctive normal form:
This step relies on the fact that the remaining operators are just $\wedge$, $\vee$, $\neg$ plus the universal quantifier. Furthermore, it was possible to move universal quantifiers, even for W-variables, outside the formulae and to finally eliminate them altogether. Reasoning with clauses is just easier to implement because the data- and control structures are much more simpler. But in first-order predicate logic also resolution methods for formulae which are not in conjunctive normal form have been developed. It should be no problem to apply these methods to P-logic as well and possibly to extend them for formulae with other operators, for instance the binary temporal logic "until" operator.

The resolution operation:
The basic operation of a resolution step is the unification of the atoms. Unification depends on the syntactic structure of the terms as well as on the semantics of the symbols, in P-logic in particular on the properties of the accessibility relation. We have considered only reflexivity, symmetry and transitivity. For different properties other unification algorithms must be developed. The property of an accessibility relation to be non-serial found its expression in the fact that a formula can become true in an interpretation not because a predicate evaluates to a truth value, but because a quantification $\forall$ u $\mathcal{F}$ about an empty set is always considered to be true, regardless of $\mathcal{F}$. This problem also occurs in many-sorted logics when empty sorts are allowed and a formula $\forall$ x:S $\mathcal{F}$ with an empty sort S is true. Our solution for this problem in P-logic is the introduction of the "End"-predicate. The conditioned instantiation and the residue seems to be elegant and computationally efficient because in many cases the End-literals disappear already during the resolution step. An analogue solution to this problem in many-sorted logics, not with an End-predicate, but with an "Empty"-predicate is obvious.

Future Directions

My hope is that the basic ideas presented in this work are powerful enough to open the door to efficient theorem proving in a much larger class of non-standard logics than the relatively simple modal systems I have examined so far. Let me therefore sketch some ideas for further work in this area.

Epistemic Logics

Hintikka originally had the idea of formalizing the propositional attitude of belief with possible worlds [Hintikka 62]. The basic concept is that the propositions of an actor's (say A) belief are represented as a set of worlds, compatible with A's beliefs. Any member of this set is, according to the way A thinks, a candidate for the real world, that is

A beliefs $\mathcal{F}$ if and only if for all w $\in$ possible-worlds(A), $\mathcal{F}$ is true in w.

Levesque, Halpern and Moses, Konolige and others have developed this idea to a formal logic with a tableau based deduction calculus [Levesque 84], [Konolige 86], [Halpern&Moses 85]. The syntax of this logic is similar to modal logic, except that there are not only the two modal operators $\Box$ and $\Diamond$, but for each actor A there is an individual pair $\Box_A$ and $\Diamond_A$ of operators. $\Box_A\mathcal{F}$ may be interpreted: "A beliefs $\mathcal{F}$" and $\Diamond_A\mathcal{F}$ may be interpreted "A thinks $\mathcal{F}$ might be possible". The semantics is a Kripke semantics where an individual accessibility relation $\mathfrak{R}_A$ is associated with each pair of modal operators $\Box_A$ and $\Diamond_A$. Therefore there is no big conceptual difference to classical modal logics. The basic idea to "skolemize modal operators" which allowed to translate a modal formula into predicate logic syntax can be applied

straightforwardly to this kind of epistemic logics. A formula $\Box_A \mathcal{F}$ is translated into $\forall w(A)\ \mathcal{F}[w(A)]$ and $\Diamond_A \mathcal{F}$ is translated into $\mathcal{F}[c(A)]$ where the world variables and skolem terms depend on the actor A.

To demonstrate this idea let us try solve the wise man puzzle, a famous example from McCarthy that has been used to test the representation ability of formalisms for knowledge and belief. The traditional form is:

A certain king wishes to determine which of his three wise men is the wisest. He arranges them in a circle so that they can see and hear each other and tells them that he will put a white or black spot on each of their foreheads but at least one spot will be white. In fact all three spots are white. He then offers his favor to the one who first tells him the colour of his spot. After a while, the wisest announces that his spot is white. How does he know?

(Actually the information that all three spots are white is not necessary to solve the puzzle.)
The solution involves the wisest man reasoning about what his colleagues know and don't know from observations and the king's announcement.
To axiomatize this puzzle in epistemic logic, assume the three wise man are A, B and C and C is the wisest. First of all we need the three formulae:

C1: $A \neq B$

C2: $A \neq C$

C3: $B \neq C$

and assume the symmetry of the $\neq$-predicate.
At least one of them has a white spot and everybody knows that everybody else knows that his colleagues know this.

C4: $\forall S, S', S'':\ \Box_S \Box_{S'} \Box_{S''} W(A) \vee W(B) \vee W(C)$

(W(S) means S has a white spot.)
The three men can see each other and they know this. Therefore whenever one of them has a white or black spot, he knows that his colleagues know this and he knows also that his colleagues know this from each other.

C5: $\forall S,S':\ S \neq S' \Rightarrow \Box_S(\neg W(S) \Rightarrow \Box_{S'} \neg W(S))$

C6: $\forall S,S',S''\ S \neq S' \wedge S \neq S'' \wedge S' \neq S'':\ \Rightarrow \Box_S \Box_{S'} (\neg W(S) \Rightarrow \Box_{S''} \neg W(S))$

C7: $\forall S,S',S''\ S \neq S' \wedge S \neq S'' \wedge S' \neq S'':\ \Rightarrow \Box_S \Box_{S'} (\neg W(S') \Rightarrow \Box_{S''} \neg W(S'))$

(We give only the minimum number of axioms which are necessary for the proof.)
They can hear each other and they know this. B did not say anything, therefore C knows that B does not know the colour of his own spot.

C8: $\Box_C \neg \Box_B W(B) \qquad (\Leftrightarrow \Box_C \Diamond_B \neg W(B))$

C knows that B knows that A does not know the colour of his spot.

C9: $\Box_C \Box_B \neg \Box_A W(A) \qquad (\Leftrightarrow \Box_C \Box_B \Diamond_A \neg W(A))$.

We translate the formulae into predicate logic syntax:

C1: $A \neq B$ C2: $A \neq C$ C3: $B \neq C$

C4: $\forall S,u, S',u', S'',u'':\ W([u(S)\ u'(S')\ u''(S'')], A) \vee W([u(S)\ u'(S')\ u''(S'')], B) \vee$
$W([u(S)\ u'(S')\ u''(S'')], C)$

C5: $\forall S,u, S',u':\ S = S' \vee W([u(S)], S) \vee \neg W([u(S)\ u'(S')], S)$

C6: $\forall S,u, S',u', S'',u'':\ S = S' \vee S = S'' \vee S' = S'' \vee$
$W([u(S)\ u'(S')], S) \vee \neg W([u(S)\ u'(S')\ u''(S'')], S))$

C7: $\forall$ S,u, S',u', S'',u'': S = S' $\vee$ S = S'' $\vee$ S' = S'' $\vee$
W([u(S) u'(S')], S') $\vee$ $\neg$W([u(S) u'(S') u''(S'')], S'))

C8: $\forall$u $\neg$W([u(C) g(B)], B)

C9: $\forall$u,v $\neg$W([u(C) v(B) h(A)], A)

A deduction of the fact that C knows the colour of his own spot, i.e. $\Box_C$ W(C) is now a trivial exercise for any resolution theorem prover. The following UR-proof was found by our system [Eisinger&Ohlbach 86]:

C1,C2,C3,C7,C8	$\rightarrow$ R1:	$\forall$u,u'' $\neg$W([u(C) g(B) u''(A)], B)	($\Leftrightarrow \Box_C \Diamond_B \Box_A \neg$W(B))
C9, R1,C4	$\rightarrow$ R2:	$\forall$u W([u(C) g(B) h(A)], C)	($\Leftrightarrow \Box_C \Diamond_B \Diamond_A$ W(C))
C1,C2,C3,R2,C6	$\rightarrow$ R3:	$\forall$u. W([u(C) g(B)], C)	($\Leftrightarrow \Box_C \Diamond_B$ W(C))
C3,R3,C5	$\rightarrow$ R4:	$\forall$u W([u(C)], C)	($\Leftrightarrow \Box_C$ W(C)) ■

Equality Reasoning in Modal Logics

Equality can either be explicitly axiomatized with the corresponding set of equality axioms or it can be built into a deduction calculus with a special inference rule like paramodulation [Robinson&Wos 69]. Since paramodulation sharply increases the efficiency of reasoning systems for predicate logic [Wos 88], it is desirable to build paramodulation also into a reasoning system for modal logic. To see the difficulties consider the formula $\mathcal{F}$: a = b $\wedge$ $\Box$P(a). Since the second occurrence of 'a' is in the scope of the $\Box$-operator and may therefore be interpreted different to the first occurrence, it is not possible to replace 'a' by 'b' and to deduce $\Box$P(b). Thus, an unrestricted application of a replacement operation in the modal logic syntax is not sound. In P-logic syntax, the modal context is available at each term and can be used to influence a deduction operation. The translated formula $\Pi(\mathcal{F})$: a[] = b[] $\wedge$ $\forall$ u P([u], a[u]) therefore can safely be paramodulated when the accessibility relation is reflexive, the unifier for a[] and a[u] is $\{u \mapsto []\}$, and the paramodulant is P([], b[]). (We assume the equality predicate to be rigid!) Thus, equality reasoning by paramodulation should be no problem in P-logic. The paramodulation rule need not be changed, just the accessibility relation dependent unification algorithms must be applied for unifying one side of an equation, which is always a D-term, with the subterm of the literal to be paramodulated. The application of the unifier must of course be performed by conditioned instantiation in non-serial interpretations.

Many-Sorted Modal Logics

Resolution and paramodulation calculi for sorted first order predicate logic have been developed for instance by [Walther 87] and [Schmidt-Schauss 85, 88]. They have shown that only two slight modifications of the unification algorithm and one modification of the paramodulation rule are necessary for handling hierarchical sort structures: A variable x of sort S1 and a variable y of sort S2 can only be unified when there is a common subsort of S1 and S2. A variable x of sort S1 can only be unified with a term t of sort S2 if S2 = S1 or S2 is a subsort of S1. The paramodulation rule must take care that a paramodulation operation with an equation whose two sides have different sorts does not increase the sort of the paramodulated term. Adapting these ideas to P-logic should be no problem when the sort structure and the sort declarations for the function symbols do not depend on the modal context. This is the case in most applications where only fixed sorts like "Integer", "Real" etc. occur.

Varying-Domain Modal Logics

In varying-domain interpretations there is no universal domain, but each world has its own domain which may or may not intersect with the domain of other worlds. That means universally quantified domain variables depend on the modal context. At least for monotonically increasing domains the idea is now to modify the translation function Π such that the W-term that characterizes the modal context is attached to the domain variables as well. Unification of such a world-depending D-variable x[p] with a D-term $f([q],t_1,\ldots,t_n)$ is possible only when the world-paths p and q are unifiable. To demonstrate this, let us try to prove the Barcan formula $\forall x \Box Px \Rightarrow \Box \forall x Px$ which does not hold in varying-domain models. If the proof fails, we have some evidence that the idea is sufficient. The P-logic clause form of the negated Barcan formula $\forall x \Box Px \wedge \Diamond \exists x \neg Px$ is: C1: P([u] x[]) C2: $\neg$P([a] f[a])

In fact, the two world-paths [] and [a] of the variable x and the symbol f are not unifiable and no refutation is possible. In constant domain interpretations on the other hand, where x has no world-path, there is the unifier $\{u \mapsto a, x \mapsto f[a]\}$.

In case the domains vary arbitrarily, there is no satisfactory solution so far because terms and atoms containing variables may have no interpretation at all in a world where the domain element bound to a variable does not exist.

Modal Logics with Linear Accessibility Relations

Linear means that there is just one sequence of worlds. The interesting case, where the interpretation of the two modal operators is not identical is when the accessibility relation $\Re$ is transitive, i.e. a total ordering. In this case for two given worlds it can always be determined which one is farer away from the initial world. The consequence is that for example a formula like $\Diamond\Box P \wedge \Diamond\Box\neg P$ is unsatisfiable when in addition R is serial. The reason is that for the two worlds denoted by the two $\Diamond$-operators, all worlds "behind" that one which is farthest away from the initial world, are also accessible from the other world. In other words there is no linear and serial interpretation where the intersection of the worlds denoted by the two $\Box$-operators is empty, and the formula requires P and $\neg$P to hold in these worlds.

We present an idea which should be capable to reason explicitly about the order of the worlds in linear Kripke structures and illustrate it with the following example:

The formula $\Diamond(\Box(P \vee Q) \wedge R) \wedge \Diamond((\neg P \wedge \neg Q) \wedge \Box\neg R)$

is unsatisfiable when the accessibility relation is linear, reflexive and transitive.

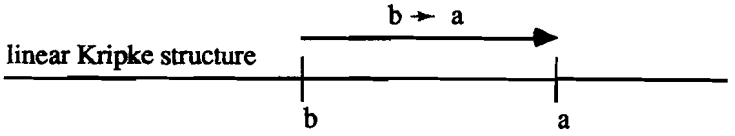
The corresponding clause set in P-Logic is:

C1: P[au] $\vee$ Q[au]

C2: R[a] C4: $\neg$Q[b]

C3: $\neg$P[b] C5: $\neg$R[bv]

In order to resolve between C2 and C5 we unify [a] and [bv]. The unifier is $\sigma := \{v \mapsto (b \twoheadrightarrow a)\}$ with the intended meaning: $(b \twoheadrightarrow a)$ maps 'b' to 'a', provided 'b' lies before 'a' or b = a (reflexivity):



Thus, the *conditioned* σ-instance of C5 is: $\sigma\downarrow C5 := (b \leq a \Rightarrow \neg R[b(b \twoheadrightarrow a)]) = (b > a \vee \neg R[a])$, where $[b(b \twoheadrightarrow a)]$ has been rewritten to [a] with an appropriate rewrite rule.

The resolvent between C2 and C5 is now: C2&C5 $\rightarrow$ R1: b > a.

In the same way we unify [au] and [b] of C1 and C3 and obtain a unifier $\{u \mapsto (a \twoheadrightarrow b)\}$.

The next resolvents are now:

C1,1&C3	→ R2:	a >b ∨ Q[b]
R2 & C4	→ R3:	a > b
R1&R3	→ R4:	empty. ■

This finishes my current collection of extensions to the basic modal logics where the translation method presented in this work should be immediately applicable. To extend this method also to more complex modal and temporal logics with Kripke semantics is subject of ongoing work.

Comparison with other Deduction Calculi for Modal Logics.

Classical Methods

The classical approaches to develop proof systems for logics are usually based on tableau systems, Gentzen sequent calculi and natural deduction calculi. Calculi of this kind are very flexible when applied to a new logic because they need not be based on a model theoretic semantics. An axiomatic semantics is completely sufficient to transform the axioms of the logic into inference rules. Therefore these were the first proof systems developed for modal logics before Kripke discovered a model theoretic semantics. A very good overview of the classical methods and further references are given in [Fitting 83]. Since a straightforward instantiation rule is not sound in the presence of flexible constant and function symbols, the classical methods can only be applied to the restricted case with rigid constant and function symbols only. Furthermore, from an implementation point of view, the classical methods are not very suitable for developing an automated reasoning system. The objects they are manipulating are, compared to clauses, very complicated things. Algorithms which make resolution theorem provers efficient are therefore not easily available, such as fast indexing techniques, fast tautology and subsumption recognition, macro operations like hyperresolution which avoid the generation of intermediate formulae, etc. Furthermore these calculi usually contain an instantiation rule for universally quantified variables which blows up the search space. The resolution rule on the other hand applies a unification algorithm to compute - and not to search - the necessary instantiation. Its search space has therefore always a finite branching rate which makes it clearly superior to methods with an uncontrolled instantiation rule.

Matrix Proof Methods

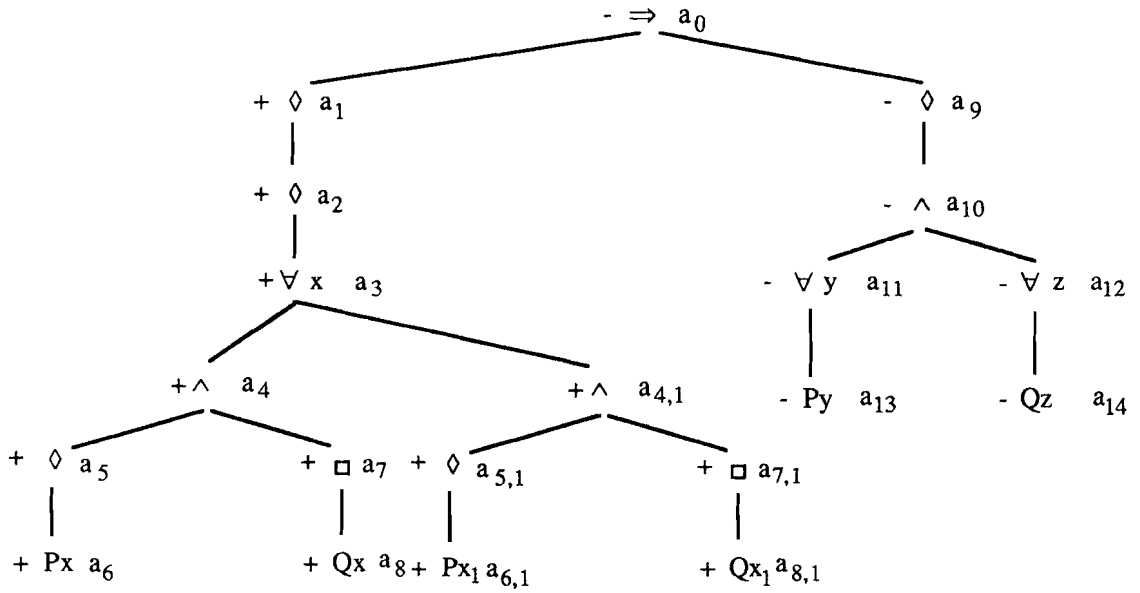
The matrix methods, pioneered by Prawitz [Prawitz 60], and further developed by Andrews [Andrews 81] and Bibel [Bibel 81] for predicate logic have recently been extended to modal logics without flexible constant and function symbols by Lincoln Wallen [Wallen 87]. The major features of this method may be summarized with Lincoln's words as follows. Validity within a logic is characterized by the existence of a set of connections (pairs of atomic formula occurrences: one positive, one negative) within a formula, with the property that every so-called atomic path through the formula contains (as a subpath) a connection from the set. Such a set of connections is said to span the formula. For classical propositional logic this condition suffices. For first-order logic a substitution must be found under which the (then propositional) connections in the spanning set are complementary. For modal logic additional conditions must ensure that, semantically, the two atomic formulae of a connection can be interpreted as inhabiting the same world.

The formula to be proved is represented as a formula tree in the usual way, but additional labels are attached to the nodes containing information about the polarity of the subformula and names for the positions of the subformula inside the formula.

An example: The annotated formula tree for our introductory example

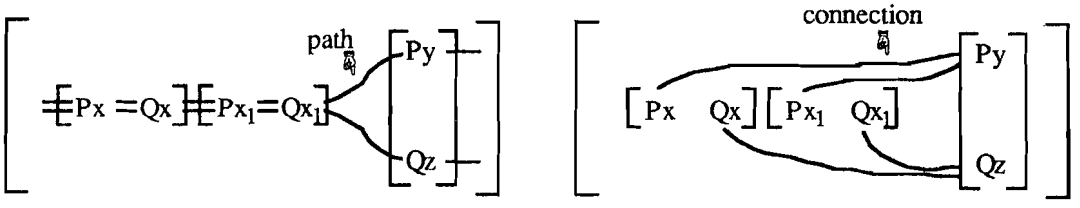
$$\Diamond\Diamond\forall x(\Diamond Px \wedge \Box Qx) \Rightarrow \Diamond(\forall y Py \wedge \forall z Qz)$$

(which actually stems from L. Wallen) is:



(Since the universal quantifier in $\forall x(\Diamond Px \wedge \Box Qx)$ is not moved over the conjunction and renamed, we need two copies of the subformula $(\Diamond Px \wedge \Box Qx)$.)

The corresponding matrix with the two possible paths and the four potential connections is:



Wallen uses strings consisting of position names a_i to represent the modal context information for the subformulae.The modal contexts of the six atoms are:

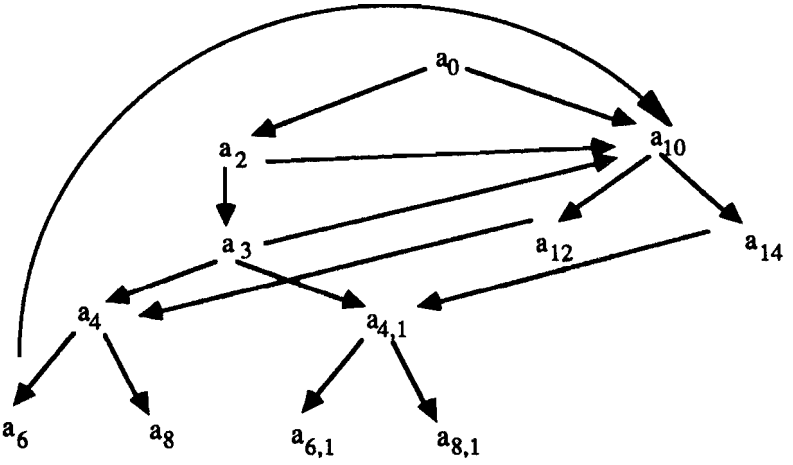
Px at position a_7:	$[a_0a_2a_3a_6]$	Px at position $a_{7,1}$:	$[a_0a_2a_3a_{6,1}]$
Qx at position a_8:	$[a_0a_2a_3\underline{a}_8]$	Qx at position $a_{8,1}$:	$[a_0a_2a_3\underline{a}_{8,1}]$
Py at position a_{12}:	$[a_0\underline{a}_{10}]$	Qz at position a_{14}:	$[a_0\underline{a}_{10}]$

According to the polarity and the modal operators, some of the a_i have the status of a constant and some have the status of a variable. The underlined names $\underline{a}_8$, $\underline{a}_{8,1}$ and $\underline{a}_{10}$ in this example have the status of a variable.

Note that from the domain variables only x and x_1 have the status of a universally quantified variable. y and z have, due to the negative polarity of the quantifier, the status of an existential variable.

In order to determine which of the potential connections form a spanning set, one possibility is to unify the atoms Px and Py as well as Qx and Qz simultaneously. This is not possible because y and z are different. The second possibility is to unify Px and Py as well as Qx_1 and Qz simultaneously yielding $\{x \mapsto y, x_1 \mapsto z\}$. In addition the corresponding modal contexts $[a_0a_2a_3a_6]$ and $[a_0\underline{a}_{10}]$ as well as $[a_0a_2a_3\underline{a}_{8,1}]$ and $[a_0\underline{a}_{10}]$ must be unified. In case the accessibility relation is transitive, this is possible

and the unifier is $\{\underline{a}_{10} \mapsto a_2a_3a_6, \underline{a}_8 \mapsto a_6, \underline{a}_{8,1} \mapsto a_6\}$. The predicate logic part and the modal part of the unifier, however, are not independent of each other but correlated over the formula tree. The substitutions determine new "is subterm of" relations which must be added to the original formula tree. The fact that x is mapped to y for example relates node a_{12} with node a_4 (the nodes below the corresponding quantifiers) and the other component $x_1 \mapsto z$ relates node a_{14} with node $a_{4,1}$. Thus, the following "is subterm of" relations between the various nodes are obtained:



Since this graph contains a cycle, the predicate and modal parts of the unifier are not compatible and the unification therefore fails.

The example has shown that Wallen's method to represent the modal context information explicitly as terms and to unify these terms is very similar to the method I presented in this monograph. The only difference is that no skolemization is performed. Neither the existential quantifiers nor the names denoting those worlds which depend on other variables are replaced by skolem functions. Therefore instead of an occurs check in the terms, a complicated cycle test in the formula tree must be performed.

Summarizing one can say that for the restricted case without flexible constant and function symbols Wallen's method is essentially to my one as Andrews's and Bibel's matrix methods are to the resolution calculus in predicate logic. The matrix methods for predicate logic have always the choice to work on the initial structure of the formula or on a clause form and to benefit from all the redundancy removing algorithms which work on clauses. To have a similar choice for the modal case one must skolemize the existential quantifiers as well as the $\Diamond$-operator, thus, one must translate modal formulae into P-logic. In this case the original matrix methods should - at least for the serial case - be immediately applicable. At the time being it is undecided which of these methods are to be preferred.

Nonclausal Modal Resolution

After earlier attempts of Fariñas del Cerro [Fariñas 85], Abadi and Manna have developed a resolution system for modal logics which works on the original modal syntax, but replaces uncontrolled instantiation by unification [Abadi&Manna 86].

The nonclausal resolution for classical propositional logic is:

$$A(\mathcal{F},\ldots,\mathcal{F}), B(\mathcal{F},\ldots,\mathcal{F}) \rightarrow A[\mathcal{F}/\text{true}] \vee B[\mathcal{F}/\text{false}]$$

That is, if the formulae $A(\mathcal{F},\ldots,\mathcal{F})$ and $B(\mathcal{F},\ldots,\mathcal{F})$ have a common subformula $\mathcal{F}$, then we can derive a resolvent $A[\mathcal{F}/\text{true}] \vee B[\mathcal{F}/\text{false}]$ by substituting 'true' for certain (one or more) occurrences of $\mathcal{F}$ in $A(\mathcal{F},\ldots,\mathcal{F})$ and 'false' for certain occurrences of $\mathcal{F}$ in $B(\mathcal{F},\ldots,\mathcal{F})$, and taking the disjunction of the

modal context. To bring potential resolution partners into the same modal context, additional move rules for modal operators are necessary, for example $\Box\mathcal{F}, \Diamond\mathcal{G} \rightarrow \Diamond(\mathcal{F} \wedge \mathcal{G})$. These rules introduce new formulae, thus wrecking the advantages of working in nonclausal form and avoiding the multiplication into conjunctive normal form.

Abadi and Manna defined a system for quantified modal logics which consists of the appropriate restricted resolution rule for the first-order case, rules for moving modal operators and quantifiers and rules for simplifying formulae containing 'true' or 'false'. The move rules for operators can be applied nondeterministically, i.e. they require additional search. In the modal resolution calculus I presented in this work exactly this additional amount of search is replaced by the deterministic unification algorithm for world-paths which *computes* the modal context for a sound resolution operation.

To compare both methods from a practical point of view, Abadi and Manna's proof for the Barcan formula is listed below [Abadi&Manna 86]:

"We prove that $\Box(\forall x\ P(x)) \Rightarrow (\forall x\ \Box P(x))$

in the resolution system for K. We will derive 'false' from

$$\neg(\neg\Box(\forall x\ P(x)) \vee (\forall x\ \Box P(x)))$$

By the negation rules we first get

$$\Box(\forall x\ P(x)) \wedge (\exists x\ \Diamond \neg P(x))$$

The rule for moving quantifiers of existential force yields

$$\exists x'\ (\Box(\forall x\ P(x)) \wedge \Diamond \neg P(x'))$$

The modality rule in the system K yields

$$\exists x'(\Box(\forall x\ P(x)) \wedge \Diamond \neg P(x') \wedge \Diamond((\forall x\ P(x)) \wedge \neg P(x')))$$

Weakening reduces this sentence to

$$\exists x'\ \Diamond((\forall x\ P(x)) \wedge \neg P(x'))$$

Take $A = \neg P(x')$, $B = P(x)$, $v_1 = P(x')$, $v_2 = P(x)$. Resolution yields

$$\exists x'\ \Diamond((\forall x\ P(x)) \wedge \neg P(x') \wedge (\neg true \vee false))$$

true-false simplification yields 'false'."

In the P-logic resolution calculus we get instead:

We prove that $\Box(\forall x\ P(x)) \Rightarrow (\forall x\ \Box P(x))$

Negation normal form of the negated theorem:

$$\Box(\forall x\ P(x)) \wedge (\exists x\ \Diamond\neg P(x))$$

Translation into P-logic:

$$\forall w,x\ P([w], x) \wedge \neg P([a], f[a])$$

Conjunctive normal form:

C1: $P([w], x)$

C2: $\neg P([a], f[a])$

(Up to this step, the transformations are perfectly deterministic. No search is necessary.)

Resolution: C1&C2 $\rightarrow$ empty clause.

Nonclausal Resolution for Epistemic Logics

Kurt Konolige has defined various calculi for his version of epistemic logics which are also applicable to modal logics [Konolige 86]. Among these calculi there is a theory resolution calculus with a so called B-resolution rule (some technical details are omitted):

$$\frac{\begin{array}{l}[S_i]\ \mathcal{F}_1 \vee \mathcal{G}_1 \\ \quad\vdots \\ \quad\vdots \\ [S_i]\ \mathcal{F}_n \vee \mathcal{G}_n \\ \neg\,[S_i]\ \mathcal{F} \vee \mathcal{G} \qquad\qquad \mathcal{F}_1,\ldots,\mathcal{F}_n \vdash_{i,\theta} \mathcal{F}\end{array}}{\theta\mathcal{G} \vee \theta\mathcal{G}_1 \vee \ldots \vee \theta\mathcal{G}_n}$$

$[S_i]$ is the "knowledge operator" for the agent i - corresponding to the $\Box$-operator for an accessibility relation $\Re_i$. "$\mathcal{F}_1,\ldots, \mathcal{F}_n \vdash_{i,\theta} \mathcal{F}$" means that $\theta\mathcal{F}$ is derivable from corresponding θ-instances of the $\mathcal{F}_i$ with the reasoning system for agent i, which may or may not be again a resolution system. The substitution θ is to be obtained by an answer extraction mechanism during the proof of $\mathcal{F}$.

The B-resolution rule is very flexible in combining different deduction systems for different agents. However, instead of a simple unification, it requires a separate proof of a subproblem to enable a single resolution operation on the higher level. This rises considerable scheduling problems to avoid getting lost in a nonterminating proof attempt for an irrelevant subproblem.

Clausal Modal Resolution

Man-chung Chan has published a resolution method which, in its kernel, contains already the idea to skolemize the $\Diamond$-operator in order to allow a transformation into conjunctive normal form [Chan 87]. It is only defined for the propositional case where it is not necessary to consider dependencies of the $\Diamond$-operator from universally quantified variables. (Actually so far Chan considered only S4.) The main obstacle that prevents the generation of a conjunctive normal form is that the $\Diamond$-operator cannot be moved over a conjunction, i.e. $\Diamond(\mathcal{F} \wedge \mathcal{G})$ is not equivalent to $\Diamond\mathcal{F} \wedge \Diamond\mathcal{G}$ because the information that there is only one world in which $(\mathcal{F} \wedge \mathcal{G})$ holds is lost. However, if each occurrence of a $\Diamond$-operator is marked with a unique index, we can safely move an indexed operator over a conjunction without losing the information that there is only one world , i.e. $\Diamond_i(\mathcal{F} \wedge \mathcal{G}) \rightarrow \Diamond_i\mathcal{F} \wedge \Diamond_i\mathcal{G}$. All modal operators can now be pushed far enough into the formulae to enable multiplication into clause form. A resolution rule can be defined where the modal operator prefixes of the literals are unified to get a common modal context for both resolution partners.

The correspondence of this method to resolution in P-logic is that the indices of the literals' modal operator prefixes are actually the world-paths in P-logic. For the propositional case, there is therefore neither a big conceptual difference nor a difference in the search behaviour.

Naive Translation into Predicate Logic.

There is a very simple method for translating a modal formula into a predicate logic formula [Moore 80]: A special predicate $\mathcal{R}$ is introduced which represents the accessibility relation. A formula $\Box\mathcal{F}$ is then translated into $\forall w\ \mathcal{R}(a,w) \Rightarrow \mathcal{F}[w]$ where 'a' denotes the current world and $\mathcal{F}[w]$ means adding 'w' as an additional argument to the literals and terms. Analogously $\Diamond\mathcal{F}$ is translated into $\exists w\ \mathcal{R}(a,w) \wedge \mathcal{F}[w]$. The properties of the accessibility relation can be expressed by simply adding the corresponding axioms for $\mathcal{R}$ to the formulae. This method is very flexible because all kinds of accessibility relations can easily be axiomatized. To see its drawbacks, let us try to prove the introductory example:

$$\Diamond\Diamond\forall x(\Diamond Px \wedge \Box Qx) \Rightarrow \Diamond(\forall y Py \wedge \forall z Qz)$$

The translated formula is:

$$\exists a\ \mathcal{R}(0,a) \wedge \exists b\ \mathcal{R}(a,b) \wedge \forall x\ (\exists c\ \mathcal{R}(b,c) \wedge P(c, x) \wedge \forall w\ \mathcal{R}(b,w) \Rightarrow Q(w, x))$$
$$\Rightarrow \exists v\ \mathcal{R}(0,v) \wedge \forall y\ P(v, y) \wedge \forall z Q(v,z))$$

The clause form of the negated formula is:

C1: $\mathcal{R}(0,a)$ C2: $\mathcal{R}(a,b)$ C3: $\mathcal{R}(b,c(x))$

C4: $P(c(x), x)$

C5: $\neg\mathcal{R}(b,w) \vee Q(w, x))$

C6: $\neg\mathcal{R}(0,v) \vee \neg P(v,f(v)) \vee \neg Q(v,g(v))$

In addition we need the transitivity law for $\mathcal{R}$ and a formula expressing its seriality:

C7: $\neg\mathcal{R}(u, v) \vee \neg\mathcal{R}(v, w) \vee \mathcal{R}(u, w)$

C8: $\mathcal{R}(u, h(u))$

With these clauses there are 15 resolution possibilities of level 0 and there is no chance that the resolution process ever stops and proves the satisfiability of the clause set. In chapter one, however, we have seen that after the translation into P-logic, there is only one resolution possibility. The process then stops and shows the satisfiability. The difference between these two methods is therefore essentially like the difference between equality handling with equality axioms and equality handling with paramodulation.

Full First-Order Clausal Modal Resolution

The basic idea for a clause based modal resolution technique is to skolemize the modal operators and then to translate modal formulae into predicate logic syntax. The earliest work in this spirit seems to be Nakamatsu and Suzuki's method for translating modal formulae into two-sorted predicate logic. They considered mainly the S4 and S5 case [Nakamatsu&Suzuki 82, 84].

In the last year two further groups have developed almost the same skolemization technique for modal operators as I did. They are Luis Fariñas del Cerro and Andreas Herzig from Toulouse and Patrice Enjalbert from Caen together with Yves Auffray from Saint-cloud [Fariñas&Herzig 88], [Enjalbert&Auffray 88]. Although the technical details are different, the net effect, a clause form with an explicit term representation of the modal context, is almost the same. Both groups, however, did not yet consider the non-serial case which is the real hard one. Furthermore they did not yet consider the effects of prefix-stability which allows to restrict the variable splitting rule in the transitive case and to obtain a terminating unification algorithm. On the other hand, Enjalbert and Auffray gave a purely predicate logic semantics for the transformed syntax which allows - in the serial case - to benefit from the results obtained for predicate logic.

Appendix A Procedural Version of the Unification Algorithms

In order to obtain implementable unification algorithms for P-logic terms the unification rules as defined in chapter 6 must be provided with a control structure. This section therefore contains a proposal for a control structure which is suitable for an immediate implementation.

Only a few parts of the unification algorithms for P-logic terms actually depend on the accessibility relation. Therefore only one algorithm is defined that gets the accessibility relation type $\mathcal{R}$, where the information about the seriality of the accessibility relation is ignored, as an additional parameter and branches internally to the $\mathcal{R}$-depending algorithm for world-paths. (Note that the accessibility relation type is just a list of key words like 'reflexive', 'symmetric' or 'transitive'.) The main control loop of the algorithm is similar to the Robinson algorithm for first-order terms.

There are two toplevel functions for unifying terms and termlists. Internally there is a function for unifying world-paths that branches to the $\mathcal{R}$-depending parts. In addition there are some auxiliary functions which are called from different places inside the main functions.

Function Unify-terms (s, t, $\mathcal{R}$)
Input: s and t are either empty lists or two prefix-stable terms or atoms.
$\mathcal{R}$ is the accessibility relation type.
Output: A complete set of idempotent and prefix-preserving unifiers for s and t.
If s = t then Return {ø}
If s is a variable then Return If $s \in t$ then ø else $\{\{s \mapsto t\}\}$
If t is a variable then Return If $t \in s$ then ø else $\{\{t \mapsto s\}\}$
If Vars(s, t) = ø or s = () or t = () or topsymbol(s) ≠ topsymbol(t) then Return ø.
If s and t are CW-terms then Return Unify-termlists(arguments(s), arguments(t), $\mathcal{R}$)
Let $s =: f(v,s_1,\ldots,s_n)$ and $t =: f(w,t_1,\ldots,t_n)$
Let Ξ := Unify-world-paths (v, w, $\mathcal{R}$)
Return $\bigcup_{\xi \in \Xi}\{\xi\theta \mid \theta \in$ Unify-termlists $(\xi(s_1,\ldots,s_n), \xi(t_1,\ldots,t_n), \mathcal{R})\}$

Function Unify-termlists (s, t, $\mathcal{R}$)
Input: Two prefix-stable termlists $s =: (s_1 \ldots s_n)$ and $t =: (t_1 \ldots t_m)$.
$\mathcal{R}$ is the accessibility relation type.
Output: A complete set of prefix-preserving idempotent unifiers for s and t.
If s = t then Return {ø}.
Let Ξ := Unify-terms $(s_1, t_1, \mathcal{R})$
Return $\bigcup_{\xi \in \Xi}\{(\xi\theta)_{|Vars(s,t)} \mid \theta \in$ Unify-termlists $(\xi(s_2 \ldots s_n), \xi(t_2 \ldots t_n), \mathcal{R})\}$.

Function Unify-world-paths (s, t, $\mathcal{R}$)
Input: Two world-paths and an accessibility relation type $\mathcal{R}$.
Output: A complete set unifiers for s and t.
If s = t then Return {ø}.
Return Case $\mathcal{R}$ is

ø	then Unify-termlists (s, t, $\mathcal{R}$)
{reflexive} or {symmetric} or both	then Unify-world-paths-reflexive-or-symmetric (s, t, $\mathcal{R}$)
{transitive}	then Unify-world-paths-transitive (s, t, $\mathcal{R}$)
{reflexive, transitive}	then Unify-world-paths-transitive (s, t, $\mathcal{R}$)
{reflexive, symmetric, transitive}	then Unify-world-paths-equivalence (s, t, $\mathcal{R}$).

Function Unify-world-paths-reflexive-or-symmetric (s, t, $\mathcal{R}$)
Input: Two world-paths s =: $[s_1 \ldots s_n]$ and t =: $[t_1 \ldots t_m]$ and $\emptyset \neq \mathcal{R} \subseteq$ {reflexive, symmetric}.
Output: A complete set of unifiers for s and t.
Let Λ := Unify-instantiated-wps (Unify-terms (s_1, t_1, $\mathcal{R}$), $[s_2 \ldots s_n]$, $[t_2 \ldots t_m]$, $\mathcal{R}$)
If $\mathcal{R}$ = {reflexive} then n' := 1, m' := 1 else n' := n, m' := m.
For i = 1, ..., n' $\Lambda := \Lambda \cup$ Unify-instantiated-wps (Unify-collapse ($[s_1 \ldots s_i]$, $\mathcal{R}$), $[s_{i+1} \ldots s_n]$, t, $\mathcal{R}$).
For i = 1, ..., m' $\Lambda := \Lambda \cup$ Unify-instantiated-wps (Unify-collapse ($[t_1 \ldots t_i]$, $\mathcal{R}$), s, $[t_{i+1} \ldots t_m]$, $\mathcal{R}$).
Return Λ.

Function Unify-world-paths-transitive (s, t, $\mathcal{R}$)
Input: Two world-paths s =: $[s_1 \ldots s_n]$ and t =: $[t_1 \ldots t_m]$
$\mathcal{R}$ = {transitive} or $\mathcal{R}$ = {reflexive, transitive}.
Output: A complete set of unifiers for s and t.
Let $\Lambda := \emptyset$
For i = 0, ..., m (i = 0 is the collapsing case when reflexive $\in \mathcal{R}$)
Let Ξ := Unify-prefix (s_1, $[t_1 \ldots t_i]$, $\mathcal{R}$)
$\Lambda := \Lambda \cup$ Unify-instantiated-wps (Ξ, $[s_2 \ldots s_n]$, $[t_{i+1} \ldots t_m]$, $\mathcal{R}$).
$\Lambda := \Lambda \cup$ Unify-split (s, t, i, $\mathcal{R}$).
Repeat the For loop with s and t exchanged.
Return $\{\lambda_{|Vars(s,t)} \mid \lambda \in \Lambda\}$.

Function Unify-world-paths-equivalence (s, t, $\mathcal{R}$)
Input: Two world-paths s and t, and $\mathcal{R}$ = {reflexive, symmetric, transitive}.
Output: A complete set of unifiers for s and t.
Return Case (s, t) =

([], [])	then {ø}
([], [w]) or ([w], []) where w is a variable	then {{w ↦ []}}
([], [r]) or ([r], [])	then ø
([r], [q])	then Unify-terms (r, q, $\mathcal{R}$).

Auxiliary Functions

Function Unify-instantiated-wps (Ξ, s, t, $\mathcal{R}$)
Input: Ξ is a set of substitutions, s and t are two world-paths, $\mathcal{R}$ is the accessibility relation type.
Output: A complete set of unifiers for s and t which are smaller than some element of Ξ.
Return $\bigcup_{\xi\in\Xi}\{(\xi\theta)_{|Vars(s,t)} \mid \theta \in$ Unify-world-paths (ξs, ξt, $\mathcal{R}$).

Function Unify-collapse(t, $\mathcal{R}$)
Input: t =: $[t_1,\ldots,t_n]$ is a world-path and $\emptyset \neq \mathcal{R} \subseteq$ {reflexive, symmetric}.
Output: A complete set of unifiers which collapse t into [].
Case n = 0 Return $\{\emptyset\}$.
n > 0 Let $\Lambda := \emptyset$
If reflexive $\in \mathcal{R}$ and t_1 is a variable then
$\tau := \{t_1 \mapsto []\}$; $\Lambda := \Lambda \cup \{\tau\theta \mid \theta \in$ Unify-collapse($\tau[t_2,\ldots,t_n]$, $\mathcal{R}$))
If symmetric $\in \mathcal{R}$ and n > 1 then
For i = 2,...,n
If t_i is a variable then
$\tau := \{t_i \mapsto [t_1^{-1}]\}$; $\Xi := \{\tau\theta \mid \theta \in$ Unify-collapse($[t_2,\ldots,t_{i-1}]$, $\mathcal{R}$))
$\Lambda := \Lambda \cup \bigcup_{\xi\in\Xi} \{\xi\theta \mid \theta \in$ Unify-collapse($\xi[t_{i+1},\ldots,t_n]$, $\mathcal{R}$))
Return Λ.

Function Unify-prefix (s_1, t, $\mathcal{R}$)
Input: A W-term s_1, a world-path t =: $[t_1\ldots t_n]$ and $\mathcal{R}$= {transitive} or $\mathcal{R}$= {reflexive, transitive}
Output: If either s_1 is a variable or n = 1: A complete set of unifiers for $[s_1]$ and $[t_1\ldots t_n]$.
Return If n = 1 then Unify-terms (s_1, t_1, $\mathcal{R}$).
elseif s_1 is a variable and $s_1 \notin$ t and either n > 0 or reflexive $\in \mathcal{R}$
then $\{\{s_1 \mapsto t\}\}$.
otherwise $\emptyset$.

Function Unify-split (s, t, i, $\mathcal{R}$)
Input: Two world-paths s =: $[s_1\ldots s_n]$ and t =: $[t_1\ldots t_m]$, a positive integer
and $\mathcal{R}$= {transitive} or $\mathcal{R}$= {reflexive, transitive}.
Output: A complete set of unifiers for s and t.
If i > 1 and t_i is a variable and t_{i-1} is no variable and $s_1 \notin [t_1\ldots t_i]$ then
Let $\xi := \{s_1 \mapsto [t_1\ldots t_{i-1}u], t_i \mapsto [u\ v]\}$ where u and v are new variables
Return $\{\theta_{|Vars(s,t)} \mid \theta \in$ Unify-instantiated-wps ($\{\xi\}$, $[s_2\ldots s_n]$, $[v\ t_{i+1}\ldots t_m]$, $\mathcal{R}$)$\}$
else Return $\emptyset$.

Acknowledgements

I am indebted to my supervisor Jörg Siekmann who took the risk to engage a physicist who did not have the faintest idea of artificial intelligence and automated theorem proving. His continous support gave me the chance to get used to this fascinating new area of work and to learn the things I needed to finish this work. With unfailing patience he read earlier drafts of this thesis and considerably helped to improve the formalism and its presentation. Remaining flaws are as always of course my responsibility.

I am also indebted to my colleagues in the "Markgraf Karl" group and the stimulating environment they provided. The discussions with my colleagues, in particular with Norbert Eisinger, contributed significantly to my current knowledge about logic and automated theorem proving. Andreas Nonnengart contributed to the development of the modal unification algorithms. Last but not least I wish to thank Lincoln Wallen whose talk, given in Spring 1987 in Munich, inspired this work.

References

Abadi&Manna 86 — M. Abadi, Z. Manna. *Modal Theorem Proving.* In Proc. of 8th Conference on Automated Deducntion, pp. 172-189, 1986.

Andrews 81 — P.B. Andrews. *Theorem-proving via general matings.* Journal of the Association for Computer Machinery, 28,2, pp. 193-214, 1981.

Bibel 81 — W. Bibel. *On matrices with connections.* Journal of the Association for Computer Machinery, 28,4, pp. 633-645, 1981.

Bibel 82 — W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, Braunschweig, 1982.

Enjalbert&Auffray 88 — P. Enjalbert, Y. Auffray. *Demonstration de Theoremes en Logique Modale un Point de Vue Equationel.* Internal Report. Avions Marcel Dassault-Breguet Aviation. Saint-cloud, 1988.

Chan 87 — M. Chan. *The Recursive Resolution Method.* New Generation Computing, 5 pp. 155-183, 1987.

Chang&Lee 73 — C.-L.Chang, R.C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Science and Applied Mathematics Series (ed. W. Rheinboldt), Academic Press, New York, 1973.

Digricoli 79 — V.J. Digricoli. *Resolution by Unification and Equality.* Proc. of 4th Workshop on Automated Deduction, Texas, 1979.

Eisinger 86 — N. Eisinger. *What you always wanted to know about connection graphs.* Proc. of 8th Conference on Automated Deduction, Oxford, 1986.

Eisinger 87 — N. Eisinger. *Completeness of Resolution without Self Resolution.* Internal Report, FB. Informatik, University of Kaiserslautern, 1987.

Eisinger & Ohlbach 86 — N. Eisinger, H.J.Ohlbach. *The Markgraf Karl Refutation Procedure.* Proc. of 8th Conference on Automated Deduction, pp. 682-683, 1986.

Fariñas 85 — L. Fariñas del Cerro. *Resolution modal logics.* In *Logics and Models of Concurrent Systems*, (K.R. Apt, ed.), Springer 1985, pp 27-55.

Fariñas&Herzig 88 — L.Fariñas del Cerro, A.Herzig *Quantified Modal Logic and Unification Theory* Langages et Systèmes Informatique, Université Paul Sabatier, Toulouse. Rapport LSI n° 293, jan. 1988. See also L. Fariñas del Cerro, A. Herzig *Linear Modal Deductions.* Proc. of 9th Conference on Automated Deduction, pp. 487-499, 1988.

Fitting 72 — M.C. Fitting. *Tableau methods of proof for modal logics.* Notre Dame Journal of Formal Logic, XIII:237-247,1972.

Fitting 83 — M.C. Fitting. *Proof methods for modal and intuitionistic logics.* Vol. 169 of Synthese Library, D. Reidel Publishing Company, 1983.

Halpern&Moses 85 — J.Y. Halpern and Y. Moses. *A guide to modal logics of knowledge and belief:* preliminary draft. In Proc. of 9th IJCAI, pp 479-490, 1985.

Herbrand 30 — Herbrand, J., *Recherches sur la théory de la démonstration.* Traveaux de la Soc. des Sciences et des Lettre de Varsovier, Nr. 33,128, 1930.

Herold 83 — A. Herold. *Some Basic Notions of First Order Unification Theory.* Seki-Reprot 15/83, Inst. f. Informatik, University of Karlsruhe, 1983.

Herold 87 — A. Herold. *Combination of Unification Algorithms in Equational Theories.* Ph.D thesis, FB. Informatik, University of Kaiserslautern, 1987.

Hughes&Cresswell 68 — G.E.Hughes, M.J.Cresswell. *An Introduction to Modal Logics,* Methuen &Co., London, 1986.

Hintikka 62 — J. Hintikka. *Knowledge and Belief.* Cornell University Press, Ithaca, New York, 1962.

Konolige 86 — K.Konolige. *A Deduction Model of Belief and its Logics.* Research Notes in Artificial Intelligence, Pitman, London, 1986.

Kowalski 75 — R. Kowalski: *A Proof Procedure Using Connection Graphs.* J.ACM Vol. 22, No.4, 1975.

Kripke 59 — S. Kripke. *A Completeness Theorem in Modal Logic.* J. of Symbolic Logic, Vol 24, 1959, pp 1-14.

Kripke 63 — S. Kripke. *Semantical analysis of modal logic I, normal propositional calculi.* Zeitschrift für mathematische Logik und Grundlagen der Mathematik, Vol. 9, 1963, pp 67-96.

Levesque 84 — H.J. Levesque. *A logic of knowledge and active belief.* Proc. of American Association of Artificial Intelligence, University of Texas, Austin 1984.

Loeckx 84 — J. Loeckx, K.Sieber. *Foundations of Program Verification.* Wiley-Teubner Series in Computer Science, 1984.

Loveland 78 — D. Loveland: *Automated Theorem Proving: A Logical Basis.* Fundamental Studies in Computer Science, Vol. 6, North-Holland, New York 1978.

Martelli&Montanari 82 — Martelli, A., Montanari, U., *An Efficient Unification Algorithm.* ACM Trans. Programming Languages and Systems 4, 2, pp. 258-282 (1982).

Moore 80 — R.C. Moore. *Reasoning about Knowledge and Action.* PhD Thesis, MIT, Cambridge 1980.

Nakamatsu&Suzuki 82 — K. Nakamatsu, A. Suzuki. *A mechanical theorem proving of first-order modal logic (S5).* Trans. Inst. Electron. & Commun. Eng. Jpn. Sect. E (Japan), Vol. E65, no 12, pp. 730-736, Dec. 1982.

Nakamatsu&Suzuki 84 — K. Nakamatsu, A. Suzuki. *Automatic theorem proving for modal predicatet logic.* Trans. Inst. Electron. & Commun. Eng. Jpn. Sect. E (Japan), Vol. E67, no 4, pp. 703-210, April. 1984.

Prawitz 60 — D. Prawitz. *An Improved Proof Procedure.* Theoria, 26 pp. 102-139, 1960.

Robinson 65 — J.A. Robinson. *A Machine Oriented Logic Based on the Resolution Principle* J.ACM, Vol. 12, No 1, 1965, 23-41.

Robinson & Wos 69 — Robinson, G., Wos, L. *Paramodulation and theorem provcing in first order theories with equality.* Machine Intelligence 4, American Elsevier, New York, pp. 135-150, 1969.

Schmidt-Schauß 85 — Schmidt-Schauß, M. *A Many-Sorted Calculus with Polymorphic Functions Based on Resolution and Paramodulation.* Proc. of 9th IJCAI, Los Angeles, 1985, 1162-1168.

Schmidt-Schauß 87 — Schmidt-Schauß, M. *Unification in a Combination of Arbitrary Disjoint Equational Theories.* SEKI-Report SR-86-16, FB. Informatik, University of Kaiserslautern, 1987.

Schmidt-Schauß 88 — Schmidt-Schauß, M. *Computational aspects of an order-sorted logic with term declarations.* Thesis, FB. Informatik, University of Kaiserslautern, 1988.

Siekmann 88 — J. Siekmann. *Unification Theory* J. of Symbolic Computation, 1988.

Smullyan 68 — R.M. Smullyan. *First Order Logic*, Springer Verlag, Berlin 1968.

Stickel 85 — M. Stickel. *Automated Deduction by Theory Resolution.* Journal of Automated Reasoning Vol. 1, No. 4, 1985, pp 333-356.

Wallen 87 — L.A.Wallen. *Matrix proof methods for modal logics.* In Proc. of 10th IJCAI, 1987. see also L.A.Wallen. *Automated Proof Search in Non-Classical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* Thesis, University of Edinburgh, 1987.

Walther 87 — C. Walther: *A Many-sorted Calculus Based on Resolution and Paramodulation.* Research Notes in Artifical Intelligence, Pitman Ltd., London, M. Kaufmann Inc., Los Altos, 1987.

Wos&Winker 84 — L. Wos, S. Winker. *Open Questions solved with the Assistence of AURA.* in Contemporary Mathematics, Vol. 29, pp. 73-88, Automated Theorem Proving: After 25 Years, ed. W.Bledsoe and D. Loveland, American Mathematical Society, Providence, RI, 1984.

Wos 88 — L.Wos. *Automated Reasoning. 33 Basic Research Problems.* Prentice Hall, 1988.

Index

Special Notation

Symbol		page
$\Diamond$	possibly	4, 9
$\Box$	necessarily	4, 9
$\forall$	for all	4, 9
$\exists$	there exists	9
$\Rightarrow$	implies	4, 9
$\Leftrightarrow$	equivalent	9
$\wedge$	and	4, 9
$\vee$	or	4, 9
$\neg$	not	4, 9
$\rightarrow$	replace by	9
$\mathbb{F}_D$	D-valued function symbols	10, 16
$\mathbb{F}_W$	D-valued function symbols	16
$\mathbb{P}$	predicate symbols	10,16
Σ_M	M-signature	10
Σ_P	P-signature	16
$\mathbb{T}_D$	D-terms	10,16
$\mathbb{V}_D$	D-variable symbols	10,16
$\mathbb{V}_W$	W-variable symbols	16
$\circ$	composition of world-access functions: $(\phi \circ \gamma)(\Im) := \gamma(\phi(\Im))$	20
$\circ$	composition of substitutions: $(\sigma \circ \lambda)(t) := (\sigma\lambda)(t) = \sigma(\lambda(\Im))$	42
[x/a]	variable assignment	12
$\models_M$	M-satisfiability	13
$\Vdash_M$	M-satisfiability	14
$\Vdash_P$	P-satisfiability	21
$\in$	is subterm of	17
$\leq_{\mathcal{F}}$	ordering relation on variables	18
$\leq_{\Re}[V]$	ordering on substitutions	42
$\perp$	bottom element	19
+	concatenation	23
[x ←s]	subterm replacement	23
|...|	length of a list	23
Π	translation M-logic → P-logic	24
[...]	world-paths	4, 16
x ↦ t	substitution component	42
$\sigma_{\|V}$	restriction of substitutions	42
$\sigma\downarrow C$	conditioned instantiation	6, 61
$\Im_M$	M-interpretation	13
$\Im_P$	P-interpretation	20
$\Im_P[\sigma]$	P-interpretation	43
$\Im_{\rightarrow}$	world-access functions	19
$\Im_{T\rightarrow}$	term-world-access functions	70
$\mathfrak{F}_M$	M-frame	12
$\mathfrak{F}_P$	P-frame	20
$\mathfrak{F}_T$	T-frame	69
$\Re$	accessibility relation	11,12
$U_{\Re}(\Gamma)$	set of unifiers for Γ	49
$cU_{\Re}(\Gamma)$	complete set of unifiers for Γ	49
$\mu U_{\Re}(\Gamma)$	minimal set of unifiers for Γ	49

Rectification

The translation from M-Logic into P-Logic (def. 4.1.1) contains a strong Skolemization rule for the ◊-operator. Its Skolem functions depend on the universally quantified domain variables only, but not on the W-variables generated from embracing □-operators. A counterexample from Patrice Enjalbert, however, has shown that at least for the first-order case when the accessibility relation is symmetric this is not sound.

The example is $\Box\, \exists x\, (P(x) \wedge \Box\Diamond\neg P(x))$.

The formula is satisfiable, but the translated formula $\forall u\, P([u]\ a[u]) \wedge \forall v\, \neg P([u\ v\ c]\ a[u])$, where the Skolem function c for the ◊-operator does not depend on u and v, is unsatisfiable in P-logic and would be refuted by the modal resolution calculus (The unifier is $\{u \mapsto c, v \mapsto c^{-1}\}$.) There is a strong conjecture that this effect occurs only in the first-order case when the accessibility relation is symmetric. However, as long as the correct condition is not known, it is therefore safer to generate in any case Skolem functions for the ◊-operator which depend also on the embracing W-variables. In order to preserve prefix stability, however, instead of the W-variables themselves, their prefixes can be taken as arguments of the Skolem functions. The above formula has then to be translated into

$$\forall u\, P([u]\ a[u]) \wedge \forall v\, \neg P([u\ v\ c([u], [u\ v])]\ a[u]).$$

Unification of the two literals produces now an occur check clash, i.e. the formula is not refutable.