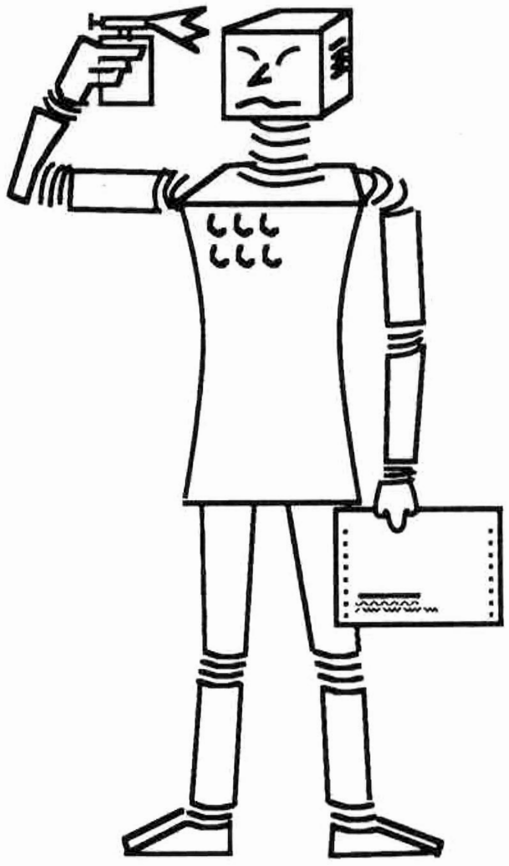
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT



Semantics Based Translation Methods for Modal Logics

H.J. Ohlbach

SEKI Report SR-90-11

Semantics Based Translation Methods for Modal Logics[1]

H.J. Ohlbach
FB. Informatik, University of Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern, Germany
email: ohlbach@informatik.uni-kl.de

Abstract A general framework for translating logical formulae from one logic into another logic is presented. The framework is instantiated with two different approaches to translating modal logic formulae into predicate logic. The first one, the well known "relational" translation makes the modal logic's possible worlds structure explicit by introducing a distinguished predicate symbol to represent the accessibility relation. In the second approach, the "functional" translation method, paths in the possible worlds structure are represented by compositions of functions which map worlds to accessible worlds. On the syntactic level this means that every flexible symbol is parametrized with particular terms denoting whole paths from the initial world to the actual world. The "target logic" for the translation is a first order many sorted logic with built in equality. Therefore the "source logic" may also be first order many sorted with built in equality. Furthermore flexible function symbols are allowed. The modal operators may be parametrized with arbitrary terms and particular properties of the accessibility relation may be specified within the logic itself.

Key words: Modal Logic, Translation of Logics, Logic Calculi.

[1] This work has been supported by the SFB 314 of the German Science Foundation (DFG) and the ESPRIT project MEDLAR.

Table of Contents

1 INTRODUCTION

For many applications in Artificial Intelligence and computer science, predicate logic is not an adequate basis to represent knowledge. In particular for systems where states and state transitions are a basic phenomenon, modal logic and its extensions turned out to be much more suitable than predicate logic. Typical examples are the logical specification of processes (programs) with discrete states and actions which transfer the process into a new state. The language which supports these concepts in a most natural way is temporal logic with possible worlds semantics. Another example is the treatment of knowledge and belief. An adequate formalization of these notions has to take into account that besides the real world, other worlds or frames of mind have to be considered where different facts may hold. Many aspects of knowldege and belief can be modelled with epistemic logics based on modal logic where the modal operators are interpreted as "belief" or "knows" operator.

Classical calculi for modal logic and all its extensions, temporal logic, epistemic logic, dynamic logic, action logic etc., are in general of Hilbert, Gentzen or Tableaux type. Most of these calculi are not very efficient for the first-order case, i.e. the branching rate in the search space is very high. Usually it is even infinite. Moreover for some variants, for example for quantified modal logic with flexible[2] function symbols and equality, no calculus of this type exists at all. A further disadvantage is that these calculi require special implementations of deduction systems. Therefore almost none of the sophisticated implementation and search control techniques developed in the last 25 years for predicate logic resolution based deduction systems can be applied.

Recently a new idea came up which changed the situation considerably. The kernel of the idea is very simple: Instead of working on the original syntax with the modal operators, modal formulae are translated into predicate logic syntax such that standard predicate logic deduction systems are applicable. The idea is very similar to the compilation method for programming languages: instead of using an interpreter for the programming language itself, a compiler translates the program into a language for which a more efficient interpreter exists, usually the operation code of a processor. This provides the freedom to design the programming language with respect to user friendliness only. It is the task of the compiler to arrange the information contained in the generated code to suit the target processor. The speedup of compiled programs compared to the interpreted programs shows that this idea works.

One of the messages of this article is therefore to learn from programming language design and to view nonclassical logics with all these fancy operators only as a user friendly surface language. Instead of calculi for these logics directly, compilers should be developed which translate formulae into a logic which enjoy an efficient calculus. In the meantime experience has shown that the standard argument against this approach, namely that efficiency is lost because the original structure of the formula is destroyed through the translation, does not hold. If the compiler is carefully designed such that the translation arranges relevant information in the right way, just the opposite is the case.

In this paper two different translation methods from modal logic into predicate logic are presented. Both of them use the possible worlds semantics of the two modal operators $\Box$ and $\Diamond$:

$\Box$F is true in a world *w* iff F is true all worlds accessible from *w*.
$\Diamond$F is true in a world *w* iff there is a world accessible from *w* and F is true in this world.

The first translation method, called "relational translation" goes back at least to [Moore 80]. The idea is to introduce a special predicate symbol R which represents the accessibility relation and to translate

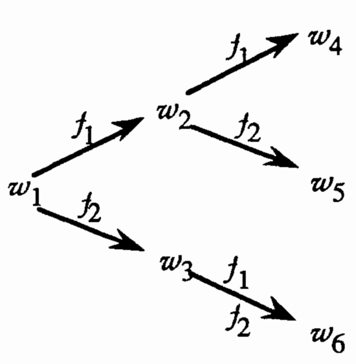
[2] Flexible designators may change their meaning from world to world.

a formula directly according to its semantics. For example $\Box P$ is translated into $\forall w\ R(0,w) \Rightarrow P'(w)$ where 0 denotes the initial world and P' is like P, but depends on a "modal context" argument. Correspondingly, $\Diamond P$ is translated into $\exists w\ R(0,w) \wedge P'(w)$. The properties of the accessibility relation can now be easily axiomatized using the R-predicate, at least in as much as they are first order axiomatizable at all.

The disadvantage of this translation method is the appearance of the R-literals. Each deduction with a normal predicate has therefore to be accompanied by a chain of deductions with the R-literals. For example a resolution operation where two world arguments of a predicate or term are unified has to be accompanied by a chain of resolutions with R-literals which "unify" in a certain sense the paths to the unified world argument. It is not easy to write a strategy which does this in a controlled way.

The second translation method, the "functional translation" uses the fact that a binary relation can be represented by the domain-range relation of a set of one place functions.

Example:



relational representation	functional representation	
$\mathcal{R}(w_1,w_2)$	$\{f_1,f_2\}$	
$\mathcal{R}(w_1,w_3)$	f_1: $w_1 \to w_2$	f_2: $w_1 \to w_3$
$\mathcal{R}(w_2,w_4)$	$w_2 \to w_4$	$w_2 \to w_5$
$\mathcal{R}(w_2,w_5)$	$w_3 \to w_6$	$w_3 \to w_3$
$\mathcal{R}(w_3,w_6)$	This set is minimal (but not unique). A maximal set has to contain all functions mapping worlds to accessible worlds.	

The path from the initial world to the actual world where a predicate or a term is to be interpreted can therefore be represented by a composition of these "modal context access functions". For example the path from w_1 to w_5 can be described by $f_1 \circ f_2$. On the syntactic level we introduce terms which denote context access functions such that sequences of these terms denote compositions of context access functions. A formula '$\Box\Diamond P$' is now translated into '$\forall f\, \exists g\ P(\downarrow(f \circ g,0))$'[3] where '$f \circ g$' denotes a composition of two context access functions and '$\downarrow(f \circ g,0)$' represents the application of the composed function to the initial world ($\downarrow$ is the application function).

The description of the properties of the accessibility relation in the functional representation is more indirect, but surprisingly it turned out to be computationally more efficient. For example reflexivity of the accessibility relation is represented by the existence of an identity function among the context access functions. Transitivity for example is represented by requiring the closure of the set of context access functions under composition. Most of these properties can be described with equations, which, in a further step, can be translated into a theory unification algorithm. It is applied to the terms denoting paths through the possible worlds structures. Equipped with these unification algorithms, the resolution rule [Robinson 65] comprises a whole bunch of steps of other calculi and therefore realizes a much better look ahead in the search space.

In order to illustrate the functional translation and the effect of the resolution on the translated formulae, consider the following formula: '$\Diamond\Diamond\forall x(\Diamond Px \wedge \Box\, Qx) \Rightarrow \Diamond\, (\forall y Py \wedge \forall z Qz)$'. Assuming

---

[3] This is not the whole truth. The translation is slightly more complicated when the accessibility relation is not serial, i.e. there may be worlds from which there are no further worlds accessible.

seriality of the accessibility relation we translate the negated formula

$\Diamond\Diamond\forall x(\Diamond Px \wedge \Box\, Qx) \wedge \Box\, (\exists y\neg Py \vee \exists z\neg Qz)$ into

$\exists f\exists g\forall x(\exists h\ P(\downarrow(f\circ g\circ h,0),x) \wedge \forall i\ Q(\downarrow(f\circ g\circ i,0),x)) \wedge \forall j(\exists y\neg P(\downarrow(j,0), y) \vee \exists z\neg Q(\downarrow(j,0), z))$.

The translation of the ◊-operators yields the additional existential quantifications whereas the translation of the □-operators yields the universal quantifications. For a particular x, the term $\downarrow(f\circ g\circ h,0)$, where h depends on x, for example denotes the world accessed by the sequence of the first three ◊-operators in the original formula. Skolemized and translated into clause form, three clauses are obtained:

C1:	$\forall x$	$P(\downarrow(f\circ g\circ h(x),0),x)$
C2:	$\forall x,i$	$Q(\downarrow(f\circ g\circ i,0),x)$
C3:	$\forall j$	$\neg P(\downarrow(j,0), a(j)) \vee \neg Q(\downarrow(j,0), b(j))$

In this example no resolution step is possible at all, unless the accessibility relation is transitive. In the transitive case the variable j which denotes a function mapping worlds to worlds accessible in one step can be bound to the term $f\circ g\circ i$ which denotes a function mapping worlds to worlds accessible in three steps, and, by transitivity, also in a single step. Therefore $\{j \mapsto f\circ g\circ i, x \mapsto b(f\circ g\circ i)\}$ is a unifier for '$Q(\downarrow(f\circ g\circ i,0),x)$' and '$Q(\downarrow(j,0), b(j))$'. The corresponding resolvent is '$\neg P(\downarrow(f\circ g\circ i,0), a(f\circ g\circ i))$', and this is in fact the only possible resolvent. Since no empty clause can be deduced, the given formula is no theorem. No classical calculus is able to detect this in such a simple way.

In this paper, however, we shall not present the special theory unification algorithm for transitivity, but stop with the corresponding equational axiomatization. A unification algorithm which is sufficient for the simple example above, i.e. modal logic D4, has been presented in [Ohlbach 88]. The equations developed in this paper, however, hold for a more general case and have not yet been transformed into theory unification algorithms.

Unlike compiler building, the construction of translators for logics has not yet been systematized and supported by standard notions and methods. Before we come to the definition of translators for specific logics we therefore try to systematize in the next chapter some of the well known notions about logics with respect to the description of these translators. The presented schema should cover all kinds of two-valued logics with model theoretic semantics. It permits the defintion and verification of the soundness and completeness of translators using the model theoretic semantics of the "source logic" and the "target logic".

In the third chapter we present the target logic we are using. It is a first order many sorted predicate logic with built in equality. The particular modal logic we are considering is defined in the fourth chapter. It is also a many sorted first order logic with built in equality. In order to allow interpretations of the logic as epistemic or action logic, the modal operators □ and ◊ may be parametrized with arbitrary terms denoting for example agents or actions. The properties of the accessibility relation are specified within the logic itself by further built in predicates, for example REFLEXIVE or SYMMETRIC. The two different translation methods are described in the following two chapters. Finally some further optimizations of the functional translation are presented.

The reader is assumed to be familiar with predicate logic and modal logic. Some knowledge about resolution and paramodulation is needed to understand the examples. Introductory textbooks are [Chang & Lee73], [Loveland 78], [Chellas 80], [Fitting 83].

In the sequel we use the following notational conventions for writing formulae: Syntactic objects are written in standard letters, whereas semantic objects are written in italics. For example if x is a variable symbol, then *x* denotes its interpretation with respect to a given variable assignment. Predicate symbols with a fixed meaning, for example SERIAL, are written with small capitals. F, G, H are used as meta symbols for formulae. '⇨' is used as a meta implication sign.

2 LOGICS AND LOGIC MORPHISMS

The kind of logics we are considering can be described by giving the syntax and its model theoretic semantics. The syntax is specified by describing the signature, i.e. the basic alphabet of nonlogical symbols, and by giving formation rules for terms and formulae. The description of the signature may already contain logical statements as for example the subsort declaration 'integer $\sqsubseteq$ real' in a sorted logic. The formation rules for terms and formulae are in general also not so straightforward as in pure predicate logic. In some of the order-sorted logics extra mechanisms have to ensure that the terms and formulae are well-sorted. The model theoretic semantics is usually defined in three steps. The first step is to define the signature interpretation, i.e. the interpretation of the nonlogical symbols. The signature interpretation itself is very often separated into the interpretation of the non variable symbols, which is the basic information necessary to interpret closed formulae, some context information as for example the initial world in modal logics, and into variable assignments which change dynamically when a quantified formula is interpreted. The second step is to turn the signature interpretation into an interpreter for terms by following the formation rules for terms. The last step is the definition of the satisfiability relation. The satisfiability relation actually fixes the meaning of the logical symbols and permits the evaluation of formulae to 'true' or 'false'.

Definition 2.1 Logics

A (two-valued) **logic** (with model theoretic semantics) is a pair (syntax, semantics) where **syntax** is a triple ($\mathbb{\Sigma}$, θ, φ) consisting of

- a set $\mathbb{\Sigma}$ of **signatures**,
- a function θ that maps a signature Σ to a set of **Σ-terms** (or terms for short) and
- a function φ that maps a signature Σ to a set of **Σ-formulae** (or formulae for short)

and **semantics** is a triple (I, Θ, $\models$) consisting of

- a function I that maps a signature Σ to the set of **signature interpretations** over Σ (or Σ-interpretations for short). Each signature interpretation consists of a symbol interpretation $\mathcal{F}$, a **context** $\mathcal{C}$ and a **variable assignment** $\mathcal{V}$,
- a function Θ that turns a signature interpretation into an **interpreter** for terms and
- a **satisfiability relation** $\models \in$ signature-interpretations $\times$ formulae. ♦

Example: With the above notions, pure predicate logic would be described as follows:

- A signature is a set of variable, function and predicate symbols. They are separated according to their arity. $\mathbb{\Sigma}$ is the set of all these signatures.
- The function θ is essentially the inductive definition of terms.
  For a given signature Σ, $\theta(\Sigma)$ yields the set of all terms built with the symbols in Σ.
- The function φ is essentially the inductive definition of formulae.
- The function I assigns to each signature Σ the set of Σ-structures (which are essentially Σ-algebras, see below) and variable assignments. Contexts are irrelevant for predicate logic.
  Given a signature Σ, an element of $I(\Sigma)$ is a particular Σ-structure which in turn relates function symbols with functions, predicate symbols with relations etc. (c.f. def. 3.6 and def. 3.7).
- The function Θ turns a signature interpretation into an interpreter for terms by lifting variable assignments to the induced homomorphisms from the term algebra into the Σ-structure.
- $\models$ is the usual satisfiability relation. ♦

A **specification** $(\Sigma, \mathbb{F})$ in a logic $\mathcal{L}$ is a signature Σ together with a set $\mathbb{F}$ of Σ-formulae.

(For instance in sorted logics, the signature itself contains nontrivial information. Therefore it is more than a technicality to keep the pair $(\Sigma, \mathbb{F})$ explicitly together.)

Definition 2.2 Satisfiability

- Given a logic $\mathcal{L}$ and an $\mathcal{L}$-signature Σ, a Σ-formula F is called **$\mathcal{L}$-satisfiable** (or simply satisfiable) iff $\mathfrak{I} \models F$ for *some* signature interpretation $\mathfrak{I}$ ($\mathfrak{I}$ **satisfies** F).
- A Σ-interpretation satisfies a specification $S = (\Sigma, \mathbb{F})$ iff it satisfies all formulae in $\mathbb{F}$.
  S is **unsatisfiable** iff it is not satisfiable
- A signature interpretation satisfying a formula or specification S is called a **model** for S.
- A Σ-formula F is a **theorem** (or tautology) iff $\mathfrak{I} \models F$ for *all* Σ-interpretations $\mathfrak{I}$. ♦

Usually there is a notion of **closed formulae** in a logic. In a closed formula all variables are bound by quantifiers. Models for closed formulae are independent of variable assignments. That means whenever a closed formula F is satisfied by an interpretation $\mathfrak{I} = (\mathcal{F}, \mathcal{C}, \mathcal{V})$ then $(\mathcal{F}, \mathcal{C}, \mathcal{V}')$ satisfies F for all variable assignments $\mathcal{V}'$. This is in general not the case for contexts. In modal logic, for example, satisfiability of closed formulae is usually defined relative to an initial context, i.e. an initial world. Therefore contexts are in general an essential part of models for formulae and specifications. Variable assignments are used in the satisfiability relation for recording (semantical) bindings to variables during a recursive descent into formulae.

We are now going to define logic morphisms as satisfiability preserving mappings between logics. They consist of a syntactic component, a mapping of signatures and formulae, and a semantic component, a mapping of interpretations. The syntactic component is essentially the "compiler" that translates specifications from one logic into another. The existence of the semantic component ensures that the syntactic translations map satisfiable specifications to satisfiable specifications (soundness) and unsatisfiable specifications to unsatisfiable specifications (completeness). With predicate logic as a target logic, a logic morphism allows **theorem proving** by **translation** (into predicate logic) **and refutation** (for example with predicate logic resolution and paramodulation).

Definition 2.3 Logic Morphisms

A **logic morphism** is a mapping Ψ between two logics $\mathcal{L}_i = ((\Sigma_i, \theta_i, \varphi_i), (I_i, \Theta_i, \models_i))$, i = 1,2. It consists of the two components $(\Psi_S, \Psi_{\mathfrak{I}})$ where

- Ψ_S is a **specification morphism** mapping $\mathcal{L}_1$-specifications to $\mathcal{L}_2$-specifications.
  Ψ_S contains the two components,
  - Ψ_Σ, a signature morphism mapping $\mathcal{L}_1$-signatures to $\mathcal{L}_2$-signatures, and
  - Ψ_F, a formula morphism mapping $\mathcal{L}_1$-formulae to $\mathcal{L}_2$-formulae such that Σ_1-formulae are mapped to $\Psi_\Sigma(\Sigma_1)$-formulae, i.e. $\forall \Sigma \in \Sigma_1$: $F \in \varphi_1(\Sigma) \Rightarrow \Psi_F(F) \in \varphi_2(\Psi_\Sigma(\Sigma))$.

  (In general, Ψ_S not only translates formulae, but adds new symbols and formulae.)
- $\Psi_{\mathfrak{I}}$ is a bidirectional **interpretation morphism**, a mapping between $\mathcal{L}_1$-interpretations and $\mathcal{L}_2$-interpretations ensuring satisfiability preservation, i.e.
  - if an $\mathcal{L}_1$-specification S is satisfied by $\mathfrak{I}_1$ then $\Psi_S(S)$ is satisfied by $\Psi_{\mathfrak{I}}(\mathfrak{I}_1)$ (soundness) and
  - if $\Psi_S(S)$ is satisfied by $\mathfrak{I}_2$ then S is satisfied by $\Psi_{\mathfrak{I}}^{-1}(\mathfrak{I}_2)$ (completeness) ♦

Examples: Transformation into negation normal form and Skolemization is a logic morphism from predicate logic into the fragment of predicate logic without existential quantifier. Notice that only the preservation of satisfiability is required. Skolemization is the typical example that transforms tautologies not necessarily into tautologies. For example the tautology $\exists xPx \vee \forall x\neg Px$ is transformed into $Pa \vee \forall x\neg Px$ which is satisfiable, but not a tautology. Transformations of Skolemized formulae into clauses is an example for a logic morphism which preserves tautologies. ♦

Proposition 2.4 The composition of two logic morphisms is again a logic morphism.
The proof is straightforward. ♦

This property permits the linking of translation steps or, the other way round, the breaking of complicated translations down into a sequence of simpler ones.

2.1 A Recipe for Logic Compilers

According to the definition of logic morphisms, the following steps are necessary for the development of compilers for logics:

- Definition of a specification morphism:
  This consists of three steps:
  1. Definition of a signature morphism Ψ_Σ that generates the necessary nonlogical symbols and appropriate declarations. For example, if the target logic is a sorted logic, then the subsort declarations and sort declarations for function symbols have to be generated. It has to be shown that the generated signature is really a signature in the target logic, which means for example that consistency of the sort declarations with the arity of function symbols has to be ensured or properties like regularity of the sort declarations have to be shown etc.
  2. Definition of a formula morphism Ψ_F. This is actually the translator for the logical formulae. It has to be shown that the translated formulae are well defined in the target logic.
  3. Definition of the specification morphism Ψ_S itself. Given a specification (Σ,F) in the source logic, the specification morphism generates a translated specification $\Psi_S((\Sigma,F)) = (\Psi_\Sigma(\Sigma), \Psi_F(F) \cup A)$ where A are some additional axioms.
- In order to prove soundness and completeness of the translation, an interpretation morphism $\Psi_\Im$ and its inverse $\Psi_\Im^{-1}$ have to be defined. $\Psi_\Im$ translates interpretations for specifications in the source logic to interpretations for the translated specifications. $\Psi_\Im$ is well defined if for any given Σ-interpretation $\Im$, $\Psi_\Im(\Im)$ is really an interpretation over the translated signature $\Psi_\Sigma(\Sigma)$. Usually this means to show that for example the denotation of sort symbols is not empty, or that subsort declarations are realized by corresponding set inclusions etc.

  The inverse interpretation morphism $\Psi_\Im^{-1}$ needs not translate arbitrary interpretations in the target logic. In most cases, translated specifications only require a fragment of the target logic. Therefore $\Psi_\Im^{-1}$ needs only to be defined for interpretations of translated specifications. Again, it has to be shown that $\Psi_\Im^{-1}$ is well defined, i.e. that $\Psi_\Im^{-1}(\Im')$ is really an interpretation in the source logic. Notice that although the notation suggests it, $\Psi_\Im \circ \Psi_\Im^{-1}$ needs not be the identity. In particular for logic morphisms like the Skolemization which translate into a fragment of the source logic, $\Psi_\Im^{-1}$ itself is the identity and therefore $\Psi_\Im \circ \Psi_\Im^{-1} = \Psi_\Im \neq$ identitiy.

  $\Psi_\Im$ and $\Psi_\Im^{-1}$ can now be used to show soundness and completeness of the translation. To this end it has to be proven - in general by structural induction - that a model $\Im$ for a specification S is translated into an interpretation $\Psi_\Im(\Im)$ satisfying $\Psi_S(S)$ (soundness) and a model $\Im'$ for a translated specification $\Psi_S(S)$ is translated back into an interpretation $\Psi_{\Im}^{-1}(\Im')$ satisfying S (completeness). In chapter 5 and 6 proofs of this type are given in detail.

3 ORDER SORTED PREDICATE LOGIC (OSPL)

As a target logic for the translation methods we are going to present we need a sorted version of predicate logic. Sorts are in principle not necessary because formulae in sorted logics can be translated into unsorted logic. For example the unsorted version of the formula '$\forall x{:}S\ P(x)$' is '$\forall x\ S(x) \Rightarrow P(x)$'. Besides the fact that sorted formulae are more compact and allow for a more efficient calculus, there is another advantage of sorts: the sorted version of an equation can in general be turned into theory a unification algorithm whereas the unsorted equivalent cannot. For example it is quite straightforward to turn the sorted commutativity axiom '$\forall x,y{:}S\ f(x,y) = f(y,x)$' into a unification algorithm. For the unsorted equivalent '$\forall x,y\ S(x) \wedge S(y) \Rightarrow\ f(x,y) = f(y,x)$' this is almost impossible.

The functional translation method produces in the translated specifications may equations with sorted variables (see def. 6.2 and 6.3). Since the ultimate goal of the translation is to turn these equations into unification algorithms, I therefore decided to use a many sorted predicate logic as target logic for the translation. A number of many sorted predicate logics have been developed so far. The different versions can be distinguished by the structure of the sort information they can represent, whether it is just a flat list of sorts, a semilattice or lattice, a feature sort structure etc., and by the kind of sort information for function and predicate symbols they can handle. The minimal requisites we need to apply the translation idea to more complex source logics are: A hierarchical sort structure, i.e. a partial order, and overloaded (polymorphic) function sort declarations. A typical example for overloaded sort declarations is '$+{:}\mathrm{real}\times\mathrm{real}\to\mathrm{real},\ \mathrm{integer}\times\mathrm{integer}\to\mathrm{integer}$'. The currently most advanced many sorted predicate logic with these features and a fully developed resolution and paramodulation calculus is that of Manfred Schmidt-Schauß [Schmidt-Schauß 89], which is an extension of Walther's many-sorted predicate logic [Walther 87].

Since Schmidt-Schauß' logic is quite complex, we briefly introduce its main notions. The reader who is familiar with sorted logics or who is only interested in the main ideas of the translation technique and not the corresponding proofs may skip this chapter. We follow in principle the usual Tarski scheme for defining syntax and semantics of predicate logic, but we have to include extra devices for handling the sort information.

In the sequel $\mathcal{DOM}(f)$ denotes the the **domain** of the function f.

3.1 Syntax of OSPL

According to the schema in def. 2.1 we have to define signatures, formation rules for terms and formation rules for formulae.

Definition 3.1 Sorted Signatures
A signature in OSPL consists of sets V_Σ, F_Σ, P_Σ and S_Σ of variable, function, predicate and sort symbols. (Constant symbols are 0-ary function symbols.) All these sets are disjoint. Function and predicate symbols are distinguished according to their arity.

Furthermore there is

- a function $S: V_\Sigma \to S$ which gives a sort to each variable symbol,
- a set of **subsort declarations** of the form $R \sqsubseteq S$,

- a set of **sort declarations** for terms. A sort declaration for a term is a tuple t:S, where t is a non variable term and S is a sort symbol. We sometimes abbreviate sort declarations $f(x_1,\ldots,x_n)$:S as $f:S_1\times\ldots\times S_n\rightarrow S$, where S_i is the sort of the variable x_i.
- a set of **sort declarations** for predicates. A sort declaration for a predicate is of the form $P:S_1\times\ldots\times S_n$, where the S_i are sorts. ♦

We assume that the equality predicate = is in $\mathbf{P}_\Sigma$ and that for all sorts R,S the predicate declaration $=:R\times S$ is also in Σ. For a signature Σ, let $\sqsubseteq_\Sigma$ be the quasi-ordering on S_Σ defined by the reflexive and transitive closure of the subsort declarations.

In the sequel we assume the existence of a single top sort D, i.e. for all $S \in S_\Sigma$: $S \sqsubseteq_\Sigma D$. This is not necessary in general, but simplifies some definitions.

While the definition of sorted signatures is essential for understanding of the functional translation technique, the subsequent definitions of well sorted terms and formulae are only needed for technical reasons in this paper.

Definition 3.2 Well Sorted Terms

The set of **well sorted terms** $\mathbf{T}_{\Sigma,S}$ of sort S in the signature Σ is (recursively) constructed by the following three rules:

- $x \in \mathbf{T}_{\Sigma,S}$ if $S(x) \sqsubseteq_\Sigma S$
- $t \in \mathbf{T}_{\Sigma,S}$ if $t:R \in \Sigma$ and $R \sqsubseteq_\Sigma S$
- $t[x/r] \in \mathbf{T}_{\Sigma,S}$ if $t \in \mathbf{T}_{\Sigma,S}$, $r \in \mathbf{T}_{\Sigma,R}$ and $x \in \mathbf{V}_\Sigma$ such that $R \sqsubseteq_\Sigma S(x)$.
  (t[x/r] means substituting r for x in t.)

The set $\mathbf{T}_\Sigma$ of all Σ-terms (well sorted terms) is defined as the union $\cup\{\mathbf{T}_{\Sigma,S} \mid S \in S_\Sigma\}$.
The sort S(t) of a term t is the sort S of the greatest (with respect to set inclusion) $\mathbf{T}_{\Sigma,S}$ containing t.
We sometimes use t:S to denote that S = S(t) is the sort of t.

The function θ of def. 2.1 can now be defined to map a signature Σ to $\mathbf{T}_\Sigma$. ♦

Definition 3.3 Well Formed Formulae

An **atom** $P(t_1,\ldots,t_n)$ is **well sorted** if $t_i \in T_{\Sigma,S_i}$ for i =1,...,n and $P:S_1\times\ldots\times S_n$ is a predicate declaration in Σ. A **well formed formula** (well sorted) is a formula built with well sorted atoms and the logical connectives and quantifiers ¬, ∧, ∨, ⇒, ⇔, ∀, ∃ in the usual way. We write ∀x:S F and ∃x:S F to indicate that the sort of the variable is S.

For convenience we assume that the quantified variables are standardized apart (renamed), i.e. formulae like ∀x∃xF or ∀xF ∨ ∃xG do not occur. ♦

3.2 Semantics of OSPL

Algebras and homomorphisms [Grätzer 79] are the basic building blocks for the definition of the semantics of well sorted terms and well formed formulae. A Σ-algebra $\mathcal{A}$ for a signature Σ consists of a carrier set $D_\mathcal{A}$ and a set of functions which correspond to Σ in the right way. The carrier set is divided into subsets according to Σ's sort structure and the domain-range relations of the functions in $\mathcal{A}$ match the corresponding sort declarations for terms. A special Σ-algebra is the algebra of free terms where the carrier set consists of the well sorted terms themselves and the functions are constructor functions for terms, i.e. they take n terms $t_1,\ldots,t_n$ and create a new term $k(t_1,\ldots,t_n)$. This fact can be exploited to define the semantics of terms just by an homomorphism from the free term algebra into a corresponding Σ-algebra. Such an homomorphism is actually an interpreter which evaluates terms in a given algebra.

Definition 3.4 Σ-Algebras

Σ-algebras are defined in three steps. As an auxiliary definition we first introduce Σ-quasi-algebras as algebras which need not respect the subsort and sort declarations in Σ.

- A **Σ-quasi-algebra** $\mathcal{A}$ for a signature Σ consists of a carrier set $D_{\mathcal{A}}$, a partial function $k_{\mathcal{A}}:D_{\mathcal{A}}^{\text{arity}(k)} \to D_{\mathcal{A}}$ for every function symbol k in Σ, a nonempty set $S_{\mathcal{A}} \subseteq D_{\mathcal{A}}$ for every sort S, such that $D_{\mathcal{A}}$ is the union of the denotations for the sort symbols in Σ, i.e. $D_{\mathcal{A}} = \bigcup \{S_{\mathcal{A}} \mid S \in S_\Sigma\}$.
- Let $\mathcal{A}$ be a Σ-quasi-algebra. We say a partial mapping $\varphi:V_\Sigma \to D_{\mathcal{A}}$ is a **partial Σ-assignment**, iff $\varphi(x) \in S(x)_{\mathcal{A}}$ for every variable $x \in \mathcal{DOM}(\varphi)$. If φ is a total function, we call it a **Σ-assignment**. The **homomorphic extension** φ_h of a (partial) Σ-assignment $\varphi:V_\Sigma \to D_{\mathcal{A}}$ on T_Σ is defined as a (partial) function $\varphi_h:T_\Sigma \to D_{\mathcal{A}}$ as follows:
  - $\varphi_h(x) =_{\text{def}} \varphi(x)$ for all Σ-variables $x \in \mathcal{DOM}(\varphi)$ and
  - for every $k(s_1,\ldots,s_n) \in T_\Sigma$:
    if $s_i \in \mathcal{DOM}(\varphi_h)$ for $i = 1,\ldots,n$ and $(\varphi_h s_1,\ldots, \varphi_h s_n) \in \mathcal{DOM}(k_{\mathcal{A}})$
    then $k(s_1,\ldots,s_n) \in \mathcal{DOM}(\varphi_h)$ and $\varphi_h(k(s_1,\ldots,s_n)) =_{\text{def}} k_{\mathcal{A}}(\varphi_h s_1,\ldots, \varphi_h s_n)$.

A Σ-assignment assigns values to variable symbols and therefore completes the interpretation of terms in a given algebra. Thus, the corresponding homomorphic extension allows interpretation of arbitrary non-ground terms in that algebra. Now we can give the final definition for Σ-algebras.

- A **Σ-algebra** $\mathcal{A}$ for a signature Σ is defined as a Σ-quasi-algebra $\mathcal{A}$ that satisfies the following additional conditions:
  - If $R \sqsubseteq S$ is in Σ then $R_{\mathcal{A}} \subseteq S_{\mathcal{A}}$
  - For all declarations $t{:}S \in \Sigma$ and for every partial Σ-assignment $\varphi:V_\Sigma \to D_{\mathcal{A}}$ with Variables(t) $\subseteq \mathcal{DOM}(\varphi)$: $t \in \mathcal{D}(\varphi_h)$ and $\varphi_h(t) \in S_{\mathcal{A}}$. ♦

There is actually a many sorted equivalent of the well known fact that first-order terms constitute the domain (Herbrand universe) of the Herbrand interpretations: The term algebra of well-sorted terms is a Σ-algebra with carrier set T_Σ if we define:

- $S_{T_\Sigma} =_{\text{def}} T_{\Sigma,S}$ for every sort $S \in S_\Sigma$.
- $\mathcal{DOM}(k_{T_\Sigma}) =_{\text{def}} \{(s_1,\ldots,s_n) \mid k(s_1,\ldots,s_n) \in T_\Sigma\}$.
- $k_{T_\Sigma}(s_1,\ldots,s_n) =_{\text{def}} k(s_1,\ldots,s_n)$.

Definition 3.5 Σ-Homomorphisms

Let Σ be a signature. A **Σ-homomorphism** is a mapping $\varphi:\mathcal{A}\to\mathcal{B}$ from a Σ-algebra $\mathcal{A}$ to a Σ-algebra $\mathcal{B}$ such that:

- $\varphi(S_{\mathcal{A}}) \subseteq S_{\mathcal{B}}$ for all $S \in S_\Sigma$.
- $\varphi(\mathcal{DOM}(k_{\mathcal{A}})) \subseteq \mathcal{DOM}(k_{\mathcal{B}})$ for all $k \in F_\Sigma$.
- If $(a_1,\ldots,a_n) \in \mathcal{DOM}(k_{\mathcal{A}})$ then $\varphi(k_{\mathcal{A}}(a_1,\ldots,a_n)) = k_{\mathcal{B}}(\varphi a_1,\ldots,\varphi a_n)$. ♦

In particular the homomorphic extensions of variable assignments $V_\Sigma \to D_{\mathcal{A}}$ are Σ-homomorphisms from the term algebra into the algebra $\mathcal{A}$. They will be used for the interpretation of formulae.

Endomorphisms on the term algebra which move only finitely many variables are usually called **substitutions**. They can be presented as a set $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where the variables $\{x_1,\ldots,x_n\}$ are the **domain** and the terms $\{t_1,\ldots,t_n\}$ are the **codomain**. A substitution σ is **idempotent** if $\sigma\sigma = \sigma$. This definition of substitutions automatically ensures that only well sorted substitutions are considered, i.e. substitutions which always produce well sorted instantiations of clauses.

A Σ-algebra does not contain objects that correspond to predicate symbols. These will be added in the definition of **Σ-structures** before we can go on and define the semantics for OSPL-formulae.

Definition 3.6 Σ-Structures
A **Σ-structure** $\mathcal{A}$ is a Σ-algebra which has additional denotations $P_{\mathcal{A}}$ for every predicate symbol $P \in \mathbf{P}_\Sigma$, such that

- $P_{\mathcal{A}}$ is a relation with $P_{\mathcal{A}} \subseteq \mathcal{A}^{\text{arity}(P)}$
- $=_{\mathcal{A}}$ is the identity on $\mathcal{A}$.

A Σ-homomorphism of Σ-structures $\varphi:\mathcal{A}\to\mathcal{B}$ is a Σ-homomorphism of the underlying Σ-algebras satisfying in addition $(a_1,\ldots,a_n) \in P_{\mathcal{A}} \Rightarrow (\varphi a_1,\ldots,\varphi a_n) \in P_{\mathcal{B}}$ ♦

Now we can define the semantics of well formed formulae consisting of a **Σ-interpretation** for terms and atoms and a satisfiability relation for formulae.

Definition 3.7 Σ-Interpretations
Let $S = (\Sigma, F)$ be a Σ-specification. A **Σ-interpretation** $\Im = (\mathcal{M}, \mathcal{V})$ for S is Σ-structure $\mathcal{M}$ together with a Σ-assignment $\mathcal{V}: V_\Sigma \to D_{\mathcal{M}}$.

Since T_Σ is a free Σ-structure, $\mathcal{V}$ induces a Σ-homomorphism $\mathcal{V}_h:T_\Sigma\to\mathcal{M}$. Therefore we need not distinguish between $(\mathcal{M}, \mathcal{V})$ and $(\mathcal{M}, \mathcal{V}_h)$. We use $\Im(t)$ as an abbreviation for $\mathcal{V}_h(t)$. ♦

Definition 3.8 The Satisfiability Relation
The **satisfiability relation**[4] $\models_P$ between **Σ-interpretations** $\Im$ and formulae is defined as follows:

$\Im \models_P P(t_1,\ldots,t_n)$ iff $(\Im(t_1),\ldots,\Im(t_n)) \in P_{\mathcal{M}}$
$\Im \models_P \forall x{:}S\ F$ iff for all $\chi \in S_{\mathcal{M}}$: $\Im[x/\chi] \models_P F$ where $\Im[x/\chi]$ is like $\Im$ but maps x to χ.
$\Im \models_P \exists x{:}S\ F$ iff there is an $\chi \in S_{\mathcal{M}}$ such that $\Im[x/\chi] \models_P F$
The remaining logical connectives are interpreted as usual. ♦

Summarizing we compress the above definitions into the following definition of order-sorted predicate logic (OSPL):

Definition 3.9 OSPL
Following the definition scheme of def. 2.1 for logics, we define OSPL as the tuple (syntax, semantics) where syntax consists of

- the set of all sorted signatures (def. 3.1),
- the formation rules for well sorted terms (def. 3.2), i.e. the function $\Sigma \to T_\Sigma$.
- the formation rules for well sorted formulae (def. 3.3)

[4] The index $_P$ (for predicate logic) in $\models_P$ is to distinguish this satisfiability relation from others to be defined later on.

and semantics consists of

- the function that maps a signature Σ to all Σ-interpretations (def. 3.7)
- the function that takes a Σ-interpretation $(\mathcal{M}, \mathcal{V})$ with a Σ-assignment $\mathcal{V}$ and maps it to the induced Σ-homomorphism $\mathcal{V}_h:T_\Sigma \rightarrow \mathcal{M}$.
- the satisfiability relation $\models_P$ of def. 3.8. ♦

As a reminder we give the definitions of the resolution and paramodulation rules. For OSPL they are only slightly different to the original rules for unsorted predicate logic [Robinson 65, Robinson & Wos 69].

Resolution:

$$\frac{\begin{array}{ll} L_1 \vee \ldots \vee L_k \vee C & \\ \neg L'_1 \vee \ldots \vee \neg L'_n \vee D & \sigma \text{ is a most general unifier for } L_1,\ldots,L_k, L'_1,\ldots,L'_n.\end{array}}{\sigma C \vee \sigma D.}$$

Paramodulation:

$$\frac{\begin{array}{ll} L[s'] \vee C & \sigma \text{ is a most general unifier for } s' \text{ and } s \\ s = t \vee D & \sigma L[\sigma s' \rightarrow \sigma t] \text{ is well sorted.}\end{array}}{\sigma L[\sigma s' \rightarrow \sigma t] \vee \sigma C \vee \sigma D \quad \text{(one occurrence of } \sigma s' \text{ is replaced by } \sigma t \text{ in } \sigma L).}$$

A unification algorithm for sorted terms can be found in [Schmidt-Schauß 89].

3.3 Quantification over Functions

In sorted predicate logics it is no problem to allow quantification over domain elements as well as over functions at the same time. If for example there is a sort D, a sort 'D→D' can be introduced and axiomatized such that it really describes functions over D. The second-order syntax with variables in functional positions can be avoided by introducing an explicit 'apply'- function. Instead of '∀f:'D→D' P(f(a))' for example '∀f:'D→D' P(apply(f,a))' is written which is first-order. With this trick second-order logic can't really be encoded in first-order logic. What is lost is that a second-order quantification ∀f:D→D... quantifies over *all* functions over D whereas a sorted first-order quantification ∀f:'D→D'... quantifies only over the functions which are in the interpretation of the sort 'D→D'. With first-order axioms it is in general not possible to enforce that these are always all functions over D. That means, given an interpretation $D_{\mathfrak{A}}$ of a sort D, the sort D → D is in second order logic completely determined as the set of *all* functions $D_{\mathfrak{A}} \rightarrow D_{\mathfrak{A}}$, whereas the sort 'D→D' can in first order logic at most be axiomatized to denote *some* functions $D_{\mathfrak{A}} \rightarrow D_{\mathfrak{A}}$.

For the functional translation methods to be presented below we need the "functional sorts" to quantify over context access functions, for example functions 'W→W' which map worlds to accessible worlds in possible worlds structures. In this application it is not only not necessary to quantify over all functions W → W, it would even be wrong. A second-order quantification ∀u:W→W ... which might be used as a replacement for the □-operator would denote not only the accessible worlds, but all worlds. Therefore with our usage of functional sorts where 'W→W' is axiomatized explicitly, we are on the safe first-order side.

Introducing first-order functional sorts means axiomatizing the 'apply'-function and the functional composition appropriately. This can be done in the following way:

Definition 3.10 (Functional OSPL-Specifications)
A **functional specification** in OSPL is a specification $(\Sigma,\mathbb{F})$ consisting of a **functional signature** and the axiomatization for the two distinguished symbols $\downarrow$ (application) and $\circ$ (composition). We assume a set S_F of sort symbols in Σ such that $\sqsubseteq_\Sigma$ is a semilattice on S_F, i.e. for each pair S_i, $S_k \in S_F$, the greatest lower bound $GLB(S_i,S_k)$ is unique if it exists. The functions we are going to define operate on the denotations of the sorts in S_F. The functional part of Σ is as follows:

1. The **functional sorts** may be '$S_1,\ldots,S_n \xrightarrow{q} S$', S_i and $S \in S_F$.
   Different symbols q may be used to distinguish different sets of functions $S_1 \times,\ldots,\times S_n \to S$.
   (We use this suggestive notation for the functional sorts to ease reading of formulae and to simplify some definitions. In subsequent chapters '$W \to W$' is used to denote functions mapping worlds to accessible worlds in the basic accessibility relation and '$W \xrightarrow{*} W$' denotes functions corresponding to its reflexive and transitive closure.)
2. Whenever a declaration '$S_1,\ldots,S_n \xrightarrow{q} S$' $\sqsubseteq$ '$D_1,\ldots,D_n \xrightarrow{r} D$' $\in \Sigma$ then $D_i \sqsubseteq_\Sigma S_i$ for $i = 1,\ldots,n$ and $S \sqsubseteq_\Sigma D$ and '$S_2,\ldots,S_n \xrightarrow{q} S$' $\sqsubseteq$ '$D_2,\ldots,D_n \xrightarrow{r} D$' $\in \Sigma$ (see comment below).
3. The sort declarations for $\downarrow$ are: $\downarrow$:'$S_1,\ldots,S_n \xrightarrow{q} S$' $\times S_1 \to$ '$S_2,\ldots,S_n \xrightarrow{q} S$'
   for every sort '$S_1,\ldots,S_n \xrightarrow{q} S$', (and '$S_2,\ldots,S_n \xrightarrow{q} S$' has to exist as a sort symbol). (For our purposes we need only application to one argument at a time, i.e. we use function currying when necessary.)
4. The sort declarations for $\circ$ have the following structure:
   $$\circ:\text{`}D_1,\ldots,D_n,S_1 \xrightarrow{i} S_2\text{'} \times \text{`}E_1,\ldots,E_n,S_2 \xrightarrow{j} S_3\text{'} \to \text{`}G_1,\ldots,G_n,S_1 \xrightarrow{k} S_3\text{'}$$
   where $G_l =_{def} GLB(D_l,E_l)$ for $l = 1,\ldots,n$, and the set of all these declarations is associative, i.e. whenever $\circ:F_1 \times F_2 \to F_{12}$, $\circ:F_{12} \times F_3 \to F_{123}$, $\circ:F_2 \times F_3 \to F_{23}$, $\circ:F_1 \times F_{23} \to F'_{123}$ are defined for functional sorts F_1, F_2 and F_3 then $F_{123} = F'_{123}$, or simply $(F_1 \times F_2) \times F_3 = F_1 \times (F_2 \times F_3)$. Furthermore the declarations are maximal. All combinations which are possible according to these rules have to be allowed for $\circ$.
   (For the composition of m-ary functions the first m-1 arguments are treated as parameters and only functions with the same parameters are composed, c.f. comment below.)
5. $\mathbb{F}$ contains all axioms of the following kind:
   a) $\forall f,g$:'$S_1,\ldots,S_n \xrightarrow{q} S$' $(\forall x{:}S_1 \downarrow(f, x) = \downarrow(g, x)) \Rightarrow f = g$.
   (Functions operating in the same way are identical.)
   b) $\forall f$:'$S_1 \xrightarrow{i} S_2$' $\forall g$:'$S_2 \xrightarrow{j} S_3$' $\forall x{:}S_1 \downarrow((f \circ g), x) = \downarrow(g, \downarrow(f, x))$ (definition of composition.)
   c) Whenever $\circ$:'$D_1,\ldots,D_n,S_1 \xrightarrow{i} S_2$' $\times$ '$E_1,\ldots,E_n,S_2 \xrightarrow{j} S_3$' is defined and $n > 0$ then
   $\forall f$:'$D_1,\ldots,D_k,S_1 \xrightarrow{i} S_2$' $\forall g$:'$E_1,\ldots,E_k,S_2 \xrightarrow{j} S_3$' $\forall x_1{:}GLB(D_1,E_1), \ldots, \forall x_k{:}GLB(D_k,E_k)$
   $\downarrow((f \circ g), x_1,\ldots,x_k) = \downarrow(f, x_1,\ldots,x_k) \circ \downarrow(g, x_1,\ldots,x_k)$ ♦

Condition 2 which reverses the sort hierarchy of functional sorts compared to the sort hierarchy of its component sorts on the domain side seems to be counter intuitive. A simple example, however, convinces that this is okay. Suppose we have a sort 'A', the two sorts 'integer' $\sqsubseteq$ 'real' and the two functional sorts 'integer$\to$A' and 'real$\to$A'. Now, every 'real$\to$A'-function is certainly applicable to integers and is therefore also an 'integer$\to$A'-function, but not vice versa. Thus, the subsort relationship must be 'real$\to$A' $\sqsubseteq$ 'integer$\to$A'.

The composition of functions with more than one argument is asymmetric in the arguments. The first n-1 arguments are treated differently to the last argument. For example the composition of the two arithmetic functions + and * yields a function $(+ \circ *)$ with $(+ \circ *)(3, 4) = 3 * (3 + 4) = 21$. Actually it is not the intention here to describe functions like + and *. The kind of n-place functions we need are essentially parametrized one-place functions, i.e. $f(x_1,\ldots,x_n) = f_{x1,\ldots,xn-1}(x_n)$, and we compose only one-place functions with the same parameter vector. The one-place functions will be used to describe general context transitions and the parameters will be used in logics with parametrized operators to enforce context switches only along labelled transitions in the possible worlds structure where the labels serve as parameters in the transition functions. The semantics of a parametrized modal operator like $\Box_p$ for example is "$\Box_p P$ is true in a world *w* iff P is true in all worlds which are accessible via *p*-labelled transitions". $\Box_p P$ can therefore be translated into $\forall x$:'D,W→W' P(↓(↓(x,p),0)). The 'D,W→W'-functions are axiomatized such that a function $\chi \in$ 'D,W→W'$_{\mathcal{A}}$, applied to *p* yields a function W → W which moves only along *p*-labelled transitions. The composition of two such functions with the same parameters therefore corresponds to moves in the context structure along transitions with the same label.

In the sequel we shall use sort symbols 'S→S' as well as expressions S → S with the usual meaning. 'S→S' in quotation marks is a syntax element and S → S is a semantic expression. Don't mix them up.

Lemma 3.11 ∘ **is associative in functional OSPL-specifications.**
Proof: Let f:S_f, g:S_g and h:S_h where $S_f =_{def}$ '$D_{f_1},\ldots,D_{f_n},S_1 \xrightarrow{i} S_2$', $S_g =_{def}$ '$D_{g_1},\ldots,D_{g_n},S_2 \xrightarrow{j} S_3$', $S_h =_{def}$ '$D_{h_1},\ldots,D_{h_n},S_3 \xrightarrow{k} S_4$', and $(S_f \times S_g) \times S_h = S_f \times (S_g \times S_h)$ in the sense of 3.10,4 holds.
We show the associativity of ∘ for the two cases:
Case: n = 0.

Let x:S_1	↓((f ∘ g) ∘ h), x)	(f ∘ g): '$S_1 \xrightarrow{ij} S_3$' for some ^{ij} according to the declaration for ∘	
	= ↓(h, ↓((f ∘ g), x))		(def. 3.10,5b)
	= ↓(h, ↓(g, ↓(f, x))	↓(f, x):S_2	(def. 3.10,5b)
	= ↓((g ∘ h), ↓(f, x))	(g ∘ h):'$S_2 \xrightarrow{jk} S_3$'	(def. 3.10,5b)
	= ↓((f ∘ (g ∘ h)), x)		
⇨	(f ∘ g) ∘ h) = (f ∘ (g ∘ h))		(def. 3.10,5a)

Case: n>0

Let x_i:D_{f_i}	↓((f ∘ g) ∘ h), $x_1,\ldots,x_n$)		
	= ↓((f ∘ g), $x_1,\ldots,x_n$) ∘ ↓(h,$x_1,\ldots,x_n$)		(def. 3.10,5c)
	= (↓(f,$x_1,\ldots,x_n$) ∘ ↓(g,$x_1,\ldots,x_n$)) ∘ ↓(h,$x_1,\ldots,x_n$)		(def. 3.10,5c)
	= ↓(f,$x_1,\ldots,x_n$) ∘ (↓(g,$x_1,\ldots,x_n$) ∘ ↓(h,$x_1,\ldots,x_n$))		
		↓(f,$x_1,\ldots,x_n$):'$S_1 \xrightarrow{i} S_2$' (first case)	
	= ↓(f ∘ (g ∘ h), $x_1,\ldots,x_n$)		(def. 3.10,5c)
⇨	(f ∘ g) ∘ h) = (f ∘ (g ∘ h))		(def. 3.10,5a) ♦

This lemma allows to write compositions of "functional terms" as strings without parentheses. Therefore from now on we write $t_1 \circ t_2 \circ \ldots \circ t_n$ instead of $(\ldots(t_1 \circ t_2) \circ \ldots) \circ t_n$.

The next theorem confirms that a functional specification really axiomatizes functions. Although the proof is quite technical, we give it in full detail because it is an important part of the functional translation technique.

Theorem 3.12 Every model $\mathcal{A}$ for a functional specification $(\Sigma,\mathbb{F})$ is isomorphic to a model $\mathcal{C}$ where the terms of sort '$S_1,\ldots,S_n \xrightarrow{i} S$' are interpreted as total functions $S_{1\mathcal{C}} \times\ldots\times S_{n\mathcal{C}} \to S_{\mathcal{C}}$.

Proof: Let $\mathcal{A}$ be a model for $(\Sigma,\mathbb{F})$.
We construct $\mathcal{C}$ and the isomorphism Ξ between $\mathcal{A}$ and $\mathcal{C}$ as follows:

a) For the nonfunctional sorts S, identify $S_{\mathcal{C}}$ with $S_{\mathcal{A}}$.
This guarantees that the subset relationships in $\mathcal{A}$ are properly transferred to $\mathcal{C}$.

b) To every $f_{\mathcal{A}} \in$ '$S_1,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{A}}$ we assign the function $\Xi(f_{\mathcal{A}}) = f_{\mathcal{C}}: S_{1\mathcal{C}} \times \dots \times S_{n\mathcal{C}} \to S_{\mathcal{C}}$ which satisfies the following condition: $\forall x \in S_{1\mathcal{A}}\ f_{\mathcal{C}}(x) = \Xi(\downarrow_{\mathcal{A}}(f_{\mathcal{A}}, x))$ ($\downarrow_{\mathcal{A}}$ is the interpretation of $\downarrow$ in $\mathcal{A}$, $S_{1\mathcal{A}} = S_{1\mathcal{C}}$). Using the axioms 3.10,5a it can be shown with induction on n that $f_{\mathcal{C}}$ is unique.

Let '$S_1,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{C}} =_{\text{def}} \{\Xi(f_{\mathcal{A}}) \mid f_{\mathcal{A}} \in$ '$S_1,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{A}}\}$. This guarantees again that the subset relationships in $\mathcal{A}$ are properly transferred to $\mathcal{C}$. However, we have to show that the definition is consistent with the subsort relationships of the functional sorts, i.e. whenever $f_{\mathcal{A}} \in$ '$S_1,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{A}} \subseteq$ '$D_1,\dots,D_n \xrightarrow{r} D'$'$_{\mathcal{A}}$ then $\Xi(f_{\mathcal{A}})$ is a total function on $D_{1\mathcal{C}} \times \dots \times D_{n\mathcal{C}} \to D_{\mathcal{C}}$.

Therefore let $f_{\mathcal{A}} \in$ '$S_1,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{A}} \subseteq$ '$D_1,\dots,D_n \xrightarrow{r} D'$'$_{\mathcal{A}}$ and $f_{\mathcal{C}} =_{\text{def}} \Xi(f_{\mathcal{A}})$.

We perform induction on n.

<u>Base Case</u>: n = 1, i.e. $f_{\mathcal{A}} \in$ '$S_1 \xrightarrow{q} S'$'$_{\mathcal{A}}$.

Let $x \in D_{1\mathcal{C}} = D_{1\mathcal{A}}$. According to def. 3.10,2, $D_1 \sqsubseteq_{\Sigma} S_1$, $S \sqsubseteq_{\Sigma} D$ and we have $D_{1\mathcal{A}} \subseteq S_{1\mathcal{A}}$. Thus, $x \in S_{1\mathcal{A}} = S_{1\mathcal{C}}$ and therefore $f_{\mathcal{C}}(x) = \Xi(\downarrow_{\mathcal{A}}(f_{\mathcal{A}}, x)) \in S_{\mathcal{A}} = S_{\mathcal{C}} \subseteq D_{\mathcal{C}}$.
Hence, $f_{\mathcal{C}} \in D_{1\mathcal{C}} \to D_{\mathcal{C}}$.

<u>Induction Step</u>: n > 1. The induction hypothesis is

$\forall g_{\mathcal{A}} \in$ '$S_2,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{A}}$: $\Xi(g_{\mathcal{A}})$ is a total function on $D_{2\mathcal{C}} \times \dots \times D_{n\mathcal{C}} \to D_{\mathcal{C}}$
Let again $x \in D_{1\mathcal{C}} \subseteq S_{1\mathcal{C}}\ f_{\mathcal{C}}(x) = \Xi(\downarrow_{\mathcal{A}}(f_{\mathcal{A}}, x)) \in$ '$S_2,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{C}}$
Since $\downarrow_{\mathcal{A}}(f_{\mathcal{A}}, x)) \in$ '$S_2,\dots,S_n \xrightarrow{q} S'$'$_{\mathcal{C}}$, according to the induction hypothesis
$f_{\mathcal{C}}(x) = \Xi(\downarrow_{\mathcal{A}}(f_{\mathcal{A}}, x)) : D_{2\mathcal{C}} \times \dots \times D_{n\mathcal{C}} \to D_{\mathcal{C}}$ and therefore $f_{\mathcal{C}} \in D_{1\mathcal{C}} \times D_{2\mathcal{C}} \times \dots \times D_{n\mathcal{C}} \to D_{\mathcal{C}}$.

From a) and b) we obtain that Ξ is a bijection between the sets $S_{\mathcal{A}}$ and $S_{\mathcal{C}}$ which are associated with the sorts S.

c) $\Xi(\downarrow_{\mathcal{A}}) =_{\text{def}} \downarrow_{\mathcal{C}}$ where $\downarrow_{\mathcal{C}}$ is the application function, i.e. $\downarrow_{\mathcal{C}}(f_{\mathcal{C}}, c) = f_{\mathcal{C}}(c)$ holds for all $f_{\mathcal{C}}$ and corresponding arguments c. It is easy to verify that this definition matches the sort declarations for $\downarrow$ (def. 3.10,3).

d) $\Xi(\circ_{\mathcal{A}}) =_{\text{def}} \circ_{\mathcal{C}}$ where $\circ_{\mathcal{C}}$ is the composition function on the interpretations of the functional sorts. It is defined as follows:

$\forall f_{\mathcal{C}} \in$ '$D_1,\dots,D_n,S_1 \xrightarrow{i} S_2$'$_{\mathcal{C}}$ $g_{\mathcal{C}} \in$ '$E_1,\dots,E_n,S_2 \xrightarrow{j} S_3$'$_{\mathcal{C}}$:
    $f_{\mathcal{C}} \circ_{\mathcal{C}} g_{\mathcal{C}} \in G_{1\mathcal{C}} \times \dots \times G_{n\mathcal{C}} \times S_{1\mathcal{C}} \to S_{3\mathcal{C}}$ where $G_{i\mathcal{C}} = D_{i\mathcal{C}} \cap E_{i\mathcal{C}}$ for i = 1,…,n such that
    $(f_{\mathcal{C}} \circ_{\mathcal{C}} g_{\mathcal{C}})(x_1,\dots,x_n, x) = g_{\mathcal{C}}(x_1,\dots,x_n, f_{\mathcal{C}}(x_1,\dots,x_n, x))$

Using the correspondences between $f_{\mathcal{C}}$ and $f_{\mathcal{A}}$, and the fact that $\text{GLB}(D_i, E_i)_{\mathcal{C}} \subseteq D_{i\mathcal{C}} \cap E_{i\mathcal{C}}$ we can prove by induction on n:
if $\circ$:$D_1,\dots,D_n,S_1 \xrightarrow{i} S_2$' × '$E_1,\dots,E_n,S_2 \xrightarrow{j} S_3$' → '$G_1,\dots,G_n,S_1 \xrightarrow{k} S_3$' is a declaration in Σ
then $f_{\mathcal{C}} \circ_{\mathcal{C}} g_{\mathcal{C}} \in$ '$G_1,\dots,G_n,S_1 \xrightarrow{k} S_3$'$_{\mathcal{C}}$.

It is easy to verify that these definitions satisfy the axioms 3.10,5 and the homomorphism conditions (in both directions): $\Xi(\mathcal{DOM}(f_{\mathcal{A}})) = \mathcal{DOM}(f_{\mathcal{C}})$ and if $a \in \mathcal{DOM}(f_{\mathcal{A}})$ then $\Xi(f_{\mathcal{A}}(a)) = f_{\mathcal{C}}(\Xi(a))$

e) Since Ξ is a bijection between the sets $S_{\mathcal{A}}$ and $S_{\mathcal{C}}$ which are associated with the sorts S we can now assign to each other function symbol k and its corresponding function $k_{\mathcal{A}}$ a unique function $k_{\mathcal{C}} = \Xi(k_{\mathcal{A}})$ such that the homomorphism conditions hold in both directions.

f) Similarly we assign to each predicate symbol P and its corresponding relation $P_{\mathcal{A}}$ a relation $P_{\mathcal{C}} = \Xi(P_{\mathcal{A}})$ such that $(a_1,\dots,a_n) \in P_{\mathcal{A}}$ if and only if $(\Xi(a_1),\dots,\Xi(a_n)) \in P_{\mathcal{C}}$.

Hence, we have constructed the desired "functional" Σ-structure for the context specification which is isomorphic to $\mathcal{A}$. The isomorphism ensures that the functional Σ-structure is also a model for the specification. ♦

In the sequel we shall use only the functional interpretation of functional specifications. We can exploit this interpretation also to simplify the syntax of terms by writing x(y) instead of ↓(x, y).

4 MODAL 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 going to use for demonstrating the one step translation techniques. Let us call this logic M-Logic for the moment (M for modal). M-Logic is an extension of OSPL and has the two modal operators $\Box$ and $\Diamond$, however parametrized with arbitrary terms. For example '$\forall$x:employee $\Box_x$ $\Box_{employer(x)}$ too-lazy(x)' is a well formed sentence in M-Logic. In an epistemic interpretation this sentence might express: every employee believes that his employer believes that he, the employee, is too lazy. The semantics is an extension of Kripke's possible worlds semantics [Kripke 59,63] where the accessibility relation is parametrized with domain elements.

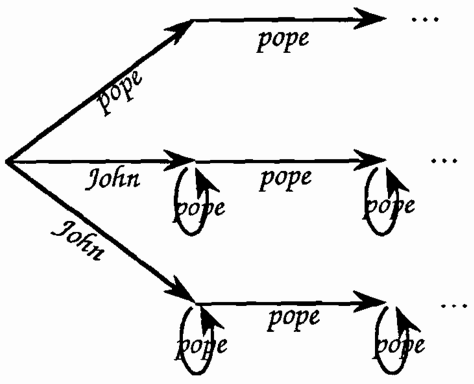
Each world in a possible worlds structure is essentially a predicate logic interpretation, and in the most general version of M-Logic the domains of these interpretations may be different from world to world (varying domain logic). That means a term like 'king-of(france)' may denote an object in one world and not denote any object in another world. M-Logic provides three different possibilities to state that a particular object exists in a particular world. The first possibility is to make a statement about this object. For example the formula 'rich(king-of(france))' not only states that the king of france is rich, but also that the king of france exists in the actual world, which is in this case the initial world. Consequently the corresponding negated formula '¬rich(king-of(france))' holds if either the king of france is not rich or if it does not exist at all. The second possibility to ensure the existence of an object is to use a special predicate 'EXISTS'. For example, 'EXISTS(king-of(france))' simply states that the king of france exists in the initial world. As another example, '$\Box_{John}$ EXISTS(king-of(france))' where '$\Box_{John}$' is interpreted as the 'belief' operator, means that the king of france exists in all worlds, which John considers in his mind. The third possibility is provided by another built in predicate 'PERSISTENT'. This predicate may be used to ensure the existence of objects not only in one world, but in all accessible worlds. 'PERSISTENT(John,p)' for example means that John exists in all worlds *p*-accessible in finitely many steps from the current world.

If a term t denotes an object that exists in a particular world it is not necessary for t's subterms to exist in that world. An example where this makes sense is 'father-of(John)'. In a world before John's birth, father-of(John) exists, although John himself does not exist. The other way round, in a world where John's father has died, John exists and father-of(John) does not. Therefore we do not build in any assumptions about the correlations of terms and subterms with respect to the EXISTS predicate.

It is common to distinguish modal logics according to the properties of the accessibility relation, for example no special properties, seriality (there is an accessible world from each world) reflexivity, symmetry etc. With parametrized operators and corresponding parametrized accessibility relations $\mathcal{R}_p$ there is the freedom to fix particular properties for each parameter separately. For example we can require $\Box_a$ to correspond to a reflexive accessibility relation, i.e. $\mathcal{R}_a$ is reflexive in all considered interpretations, while at the same time, say, $\Box_b$ corresponds to a transitive relation.

In the classical versions of modal logics, the desired properties of the accessibility relation are either included directly in the definition of the possible worlds semantics or they are implicitly specified by giving a characteristic axiom schema in an Hilbert calculus. For example the schema '$\Box P \Rightarrow P$' axiomatizes reflexive accessibility relations. Since the translation methods we are going to present make the accessibility relation an explicit part of the syntax, the specification morphism can add characteristic axioms - and not axiom schemas - describing its properties. This is controlled by certain key words like 'SERIAL', 'REFLEXIVE' etc. which we include as built in predicates. For example 'REFLEXIVE(p)' as a formula in M-Logic means that the *p*-parametrized accessibility relation is reflexive, i.e. every world which is accessed via *p*-parametrized transitions is *p*-accessible from itself. The relational translation, for example, translates 'REFLEXIVE(p)' into '$\forall$u R*(p,0,u) $\Rightarrow$ R(p,u,u)' where 0 denotes the initial world and R* denotes the reflexive and transitive closure of R which in turn stands for the accessibility relation. The meaning of this axiom is just what was said before: every world *u* which is accessible via arbitrary *p*-parametrized transitions (R*(p,0,u)) is accessible from itself (R(p,u,u)).

As soon as the description of the properties of the accessibility relation is part of the syntax, we can go one step further and allow these characteristic literals to occur as subformulae of bigger formulae. For example '$\Box_{John}$ REFLEXIVE(pope)' is a correct formula in M-Logic. In an epistemic interpretation where $\Box$ is the 'knows' or 'believes' operator, the distinction between knowledge and belief can be made by requiring the accessibility relation corresponding to the 'knows' operator to be reflexive; what is true in the knower's potential worlds is true in the real world. The interpretation of the above formula in this context is therefore: 'John believes ($\Box_{John}$) that the pope is infallible (REFLEXIVE(pope)). A possible worlds structure where this statement is true looks like:



i.e. the part of the pope's accessibility relation which is accessible by *John*-labelled transitions is reflexive. In the relational translation, this formula is translated into '$\forall$w R(John,0,w) $\Rightarrow$ $\forall$u R*(pope,w,u) $\Rightarrow$ R(pope,u,u)' where R* corresponds to the reflexive and transitive closure of R.

Definition 4.1 Syntax of M-Logic

The syntax of M-Logic is an extension of OSPL:

The signature Σ consists of the variable, function and predicate symbols, together with the various sort declarations. Besides the equality symbol we include the following unary predicate symbols (of sort D) with fixed meaning: EXISTS, SERIAL, REFLEXIVE, SYMMETRIC, TRANSITIVE, EUCLIDEAN[5], LINEAR, INCREASING-DOMAIN, and DECREASING-DOMAIN. The two place predicate symbol 'PERSISTENT' is of sort D×D. We call these special predicate symbols, except the 'EXISTS' and 'PERSISTENT' symbols, the **accessibility relation property predicates** or ARP-predicates for short. Furthermore we distinguish **rigid** and **flexible** function and predicate symbols.

Terms and formulae are built just as in OSPL, however with two additional operators $\Box_p$ and $\Diamond_p$: If F is a formula and p is a term then F' = $\Box_p$F and F' = $\Diamond_p$F are formulae. The free variables of F' are the free variables of F together with the free variables of p. ♦

[5] A relation $\mathcal{R}$ is called euclidean iff whenever $\mathcal{R}(a, b)$ and $\mathcal{R}(a, c)$ holds then $\mathcal{R}(b, c)$ and $\mathcal{R}(c, b)$.

Definition 4.2 Semantics of M-Logic

The triple semantics = $(\mathcal{I},\Theta,\models)$ (def.2.1) defines a possible worlds semantics for M-Logic as follows:

- $\mathcal{I}$ maps a signature Σ to a Σ-interpretation $\mathfrak{I} = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ where
  - $\mathcal{W}$ is a set of "worlds"
  - $\mathcal{R}$ is the accessibility relation. It is parametrized with domain elements, i.e. for a particular domain element (parameter) the accessibility relation is a usual binary relation on worlds. For a particular parameter p, $\mathcal{R}_p^*$ denotes the reflexive and transitive closure of $\mathcal{R}_p$. The tuple $(\mathcal{W}, \mathcal{R})$ is sometimes called a **frame** [Fitting 83].
    For a world $w \in \mathcal{W}$: $\mathcal{R}_p(w) =_{\text{def}} \{w' \mid \exists w_1 \ldots w_n\ \mathcal{R}_p(w,w_1)$ and $\ldots$ and $\mathcal{R}_p(w_n,w')\} \cup \{w\}$.
    $\mathcal{R}_p(w)$ is the part of the possible worlds structure which is accessible from w via finitely many p-labelled transitions.
  - $\mathcal{P}$ maps worlds to Σ-structures (def. 3.6) over the same carrier set. The interpretation of sorts and rigid symbols is the same in all worlds.
  - The context w (the actual world) is a particular world in $\mathcal{W}$ and
  - $\mathcal{V}$ is a variable assignment, i.e. a mapping from variables to domain elements.

In the sequel $\mathfrak{I}[w]$ is like $\mathfrak{I}$ except that the actual world is now w and $\mathfrak{I}[x/\chi]$ is like $\mathfrak{I}$ except that $\mathcal{V}$ maps the variable x to the value χ.

- The homomorphism $\Theta(\mathfrak{I})$ interprets terms in the actual world w, more precisely in $\mathcal{P}(w)$ which is a standard predicate logic interpretation. We usually write $\mathfrak{I}(t)$ instead of $\Theta(\mathfrak{I})(t)$ to denote t's value in the particular interpretation $\mathfrak{I}$.
- The satisfiability relation $\models_M$ is defined as follows: Let $\mathfrak{I} = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ and $\mathcal{A} = \mathcal{P}(w)$.
  - $\mathfrak{I} \models_M$ EXISTS(t) iff $\mathfrak{I}(t) \in \text{EXISTS}_{\mathcal{A}}$.
  - $\mathfrak{I} \models_M$ PERSISTENT(t,p) iff $\mathfrak{I}(t) \in \text{EXISTS}_{\mathcal{P}(u)}$ for all all $u \in \mathcal{R}_{\mathfrak{I}(p)}(w)$
  - The interpretation of the ARP-predicates is:
    $\mathfrak{I} \models_M$ SERIAL(p) iff the $\mathcal{R}_{\mathfrak{I}(p)}(w)$-part of $\mathcal{R}$ is serial,
    similar for the predicates REFLEXIVE, SYMMETRIC, TRANSITIVE, EUCLIDEAN and LINEAR.
  - $\mathfrak{I} \models_M$ INCREASING-DOMAIN(p) iff for all w_1, w_2 which are accessible from w in $\mathcal{R}^*_{\mathfrak{I}(p)}$:
    $\mathcal{R}_{\mathfrak{I}(p)}(w_1,w_2)$ implies $\text{EXISTS}_{\mathcal{P}(w_1)} \subseteq \text{EXISTS}_{\mathcal{P}(w_2)}$
  - $\mathfrak{I} \models_M$ DECREASING-DOMAIN(p) iff for all w_1, w_2 which are accessible from w in the $\mathcal{R}^*_{\mathfrak{I}(p)}$:
    $\mathcal{R}_{\mathfrak{I}(p)}(w_1,w_2)$ implies $\text{EXISTS}_{\mathcal{P}(w_1)} \supseteq \text{EXISTS}_{\mathcal{P}(w_2)}$.
  - $\mathfrak{I} \models_M P(t_1,\ldots,t_n)$ iff $\mathfrak{I}(t_i) \in \text{EXISTS}_{\mathcal{A}}$ for i = 1,...,n and $(\mathfrak{I}(t_1),\ldots,\mathfrak{I}(t_n)) \in P_{\mathcal{A}}$.
  - The interpretation of the logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ is as usual.
  - $\mathfrak{I} \models_M \forall x{:}S\ F$ iff for all $\chi \in S_{\mathcal{A}} \cap \text{EXISTS}_{\mathcal{A}}$ $\mathfrak{I}[x/\chi] \models_M F$.
  - $\mathfrak{I} \models_M \exists x{:}S\ F$ iff there is an $\chi \in S_{\mathcal{A}} \cap \text{EXISTS}_{\mathcal{A}}$ with $\mathfrak{I}[x/\chi] \models_M F$.
  - $\mathfrak{I} \models_M \Box_p F$ iff for all worlds w', $\mathcal{R}_{\mathfrak{I}(p)}(w, w')$ implies $\mathfrak{I}[w'] \models_M F$.
  - $\mathfrak{I} \models_M \Diamond_p F$ iff there is a world w' with $\mathcal{R}_{\mathfrak{I}(p)}(w, w')$ and $\mathfrak{I}[w'] \models_M F$. ♦

The encoding of varying domains is somewhat tricky as the interpretation of the quantifiers shows. The domain $D_{\mathcal{A}}$ and the interpretation of the sort symbols is the same in all worlds, but the EXISTS-predicate may be different in different worlds. The EXISTS-predicate is used to filter out in each world those objects which are considered as existing. This idea solves a serious problem which comes up in the alternative approach where the domain as well as the denotation of the sort symbols varies from world to world. In this approach, function symbols need to be interpreted as partial functions. But how can we interpret the term 'father-of(John)' in a world where John does not exist, i.e. 'John's interpretation is undefined, but his father exists and therefore 'father-of(John)' should not be undefined? In the approach with the EXISTS-predicate this is trivial: 'John' and 'father-of(John)' denote domain elements in the usual way, but EXISTS(John) is false and EXISTS(father-of(John))' is true in the given world.

In the sequel we sometimes omit the parameters of the modal operators and the corresponding accessibility relations. In this case we go back to the traditional versions of modal logic.

Remarks 4.3: According to def. 2.2, a closed formula F is satisfiable iff it is satisfiable by some interpretation, i.e. by some frame $(\mathcal{W}, \mathcal{R})$ and some **initial world** $w \in \mathcal{W}$. It is well known that in this case only the part of the possible worlds structure is necessary which is accessible from w in finitely many steps [Chellas 80]. Therefore we usually assume the frame to consist only of the restricted possible worlds structure with w as origin.

5 RELATIONAL TRANSLATION

The first translation method we present is a well known method for mapping modal logic formulae into predicate logic by introducing explicitly a special predicate symbol R which represents the accessibility relation [Moore 80]. A formula '$\Box$P' for example is translated into '$\forall$w R(0,w) $\Rightarrow$ P'(w)' where P' compared to P has an additional "world context argument". For the more interesting case where the modal operators are parametrized with terms, a ternary predicate symbol 'R(p,u,v)' is needed which represents p-parametrized transitions from u to v. Correspondingly a formula '$\Box_p$P' is translated into '$\forall$w R(p,0,w) $\Rightarrow$ P'(w)'.

In the sequel the word "domain" is used to distinguish between domain elements coming from the domain elements in M-Logic and the additional "world" elements which are introduced by the translation into predicate logic. For example domain sorts are the sort symbols used in the original modal logic specification whereas "world sorts" are introduced by the translation and denote worlds, world context access functions etc. The same with "domain variables" and "world variables".

If $\mathcal{L}_1$ is M-Logic and $\mathcal{L}_2$ is OSPL, we can now define a "relational" logic morphism (def. 2.3) $\Psi_{\mathcal{R}}$ which maps M-Logic-specifications to OSPL-specifications such that satisfiability and unsatisfiability is preserved. We follow the rules given in 2.1.

Definition 5.1 The Signature Morphism Ψ_Σ
The signature morphism Ψ_Σ maps an M-Logic signature with top sort D to an OSPL-signature (def. 3.1) as follows:

- The sorts and the sort hierarchy remains unchanged. One distinguished sort W for worlds is introduced. W is isolated from the translated sort hierarchy.
- A special constant symbol 0:W is introduced. (It denotes the initial world.)
- Each flexible n-place domain function or predicate symbol and the EXISTS predicate is mapped to an 1+n-place function or predicate symbol. (The additional argument takes the world context parameter.)
  In the sequel, for a flexible M-Logic symbol f, f' $=_{\text{def}} \Psi_\Sigma$(f) denotes the corresponding translated symbol.

- In the term declarations t:S, the term t is modified by the following recursive function π_Σ which declares W to be the sort of the additional argument for the translated flexible function symbols
  - $\pi_\Sigma(x) = x$ where x is a variable. (❖)
  - $\pi_\Sigma(f(t_1,\dots,t_n)) = f'(w, \pi_\Sigma(t_1),\dots,\pi_\Sigma(t_1))$ where f is flexible and w is a new variable of sort W.
  - $\pi_\Sigma(f(t_1,\dots,t_n)) = f(\pi_\Sigma(t_1),\dots,\pi_\Sigma(t_1))$ where f is rigid.
- The predicate declarations are modified according to the above rules.
- Furthermore Ψ_Σ adds two distinguished ternary relation symbols R:D×W×W and R*:D×W×W which represent the parametrized accessibility relation and its reflexive and transitive closure (the first argument is the parameter). ♦

Definition 5.2 The Formula Morphism Ψ_F

The formula morphism Ψ_F is defined inductively over the structure of terms:

For a term or formula F, we use F_w as an abbreviation for $\Psi_F(F,w)$, i.e. $F_w =_{def} \Psi_F(F,w)$.

- $x_w = x$ if x is a variable.
- $f(t_1,\dots,t_n)_w = f'(w, t_{1w}, \dots, t_{nw})$ if f is an n-place flexible function symbol.
- $\text{EXISTS}(t)_w = \text{EXISTS}'(w, t_w)$.
- $\text{PERSISTENT}(t,p)_w = \forall u{:}W\ R^*(p_w,w,u) \Rightarrow \text{EXISTS}'(u,t_w)$.
- $\text{SERIAL}(p)_w = \text{EXISTS}'(w,p_w) \wedge \forall u{:}W\ R^*(p_w,w,u) \Rightarrow \exists v{:}W\ R(p_w,u,v)$
- $\text{REFLEXIVE}(p)_w = \text{EXISTS}'(w,p_w) \wedge (\forall u{:}W\ R^*(p_w,w,u) \Rightarrow R(p_w,u,u))$
- $\text{TRANSITIVE}(p)_w = \text{EXISTS}'(w,p_w) \wedge$ $(\forall u{:}W\ R^*(p_w,w,u) \Rightarrow (\forall v_1,v_2{:}W\ R(p_w,u,v_1) \wedge R(p_w,v_1,v_2) \Rightarrow R(p_w,u,v_2)))$
- $\text{SYMMETRIC}(p)_w = \text{EXISTS}'(w,p_w) \wedge (\forall u{:}W\ R^*(p_w,w,u) \Rightarrow \forall v{:}W\ R(p_w,u,v) \Rightarrow R(p_w,v,u))$
- $\text{EUCLIDEAN}(p)_w = \text{EXISTS}'(w,p_w) \wedge$ $(\forall u{:}W\ R^*(p_w,w,u) \Rightarrow (\forall v_1,v_2{:}W\ R(p_w,u,v_1) \wedge R(p_w,u,v_2) \Rightarrow R(p_w,v_1,v_2)))$
- $\text{LINEAR}(p)_w = \text{EXISTS}'(w,p_w) \wedge$ $(\forall u,v{:}W\ R^*(p_w,w,u) \wedge R^*(p_w,w,v) \Rightarrow R^*(p_w,u,v) \vee R^*(p_w,v,u))$
- $\text{INCREASING-DOMAIN}(p)_w = \text{EXISTS}'(w,p_w) \wedge (\forall u{:}W\ R^*(p_w,w,u) \Rightarrow$ $(\forall v{:}W\ R(p_w,u,v) \Rightarrow \forall x{:}D\ (\text{EXISTS}'(u,x) \Rightarrow \text{EXISTS}'(v,x))))$
- $\text{DECREASING-DOMAIN}(p)_w = \text{EXISTS}'(w,p_w) \wedge (\forall u{:}W\ R^*(p_w,w,u) \Rightarrow$ $(\forall v{:}W\ R(p_w,u,v) \Rightarrow \forall x{:}D\ (\text{EXISTS}'(u,x) \Leftarrow \text{EXISTS}'(v,x))))$
- $P(t_1,\dots,t_n)_w = \text{EXISTS}'(w,t_{1w}) \wedge\dots\wedge \text{EXISTS}'(w,t_{nw}) \wedge P'(w,t_{1w},\dots,t_{nw})$ if P is a flexible predicate symbol.
- $(\forall x\ F)_w = \forall x\ \text{EXISTS}'(w,x) \Rightarrow F_w$.
- $(\exists x\ F)_w = \exists x\ \text{EXISTS}'(w,x) \wedge F_w$.
- $(\Box_p\ F)_w = \text{EXISTS}'(w,p_w) \Rightarrow (\forall u{:}W\ R(p_w,w,u) \Rightarrow F_u)$.
- $(\Diamond_p\ F)_w = \text{EXISTS}'(w,p_w) \wedge \exists u{:}W\ R(p_w,w,u) \wedge F_u$.
- For all other cases Ψ_F is the usual homomorphic extension.

The toplevel call of the translation algorithm is: $\Psi_F(F,0)$. ♦

Example: $\Psi_F(\forall x\ \Box_{f(x)}\ P(x)\ ,0)$

$$\begin{aligned}
&= (\forall x\ \Box_{f(x)}\ P(x))_0\\
&= \forall x\ \text{EXISTS}'(0,x) \Rightarrow (\Box_{f(x)}\ P(x))_0\\
&= \forall x\ \text{EXISTS}'(0,x) \Rightarrow (\text{EXISTS}'(0,f(x)_0) \Rightarrow (\forall u{:}W\ \text{R}(f(x)_0,0,u) \Rightarrow (P(x))_u))\\
&= \forall x\ \text{EXISTS}'(0,x) \Rightarrow (\text{EXISTS}'(0,f(0,x)) \Rightarrow (\forall u{:}W\ \text{R}(f(0,x),0,u) \Rightarrow P(u,x)))\ \blacklozenge
\end{aligned}$$

Definition 5.3 The Specification Morphism Ψ_S

Ψ_Σ and Ψ_F specify how to translate symbols and formulae. In addition the specification morphism Ψ_S has to add the characteristic formulae to the translated specifications which axiomatize the reflexive and transitive closure of $\mathcal{R}$:

❶ $\forall p{:}D,u{:}W\ \ \text{R}^*(p,0,u)$ (see the remark after the definition of M-Logic's semantics)
❷ $\forall p{:}D,u{:}W\ \ \text{R}^*(p,u,u)$
❸ $\forall p{:}D,u,v,w{:}W\ \ \text{R}(p,u,v) \land \text{R}^*(p,v,w) \Rightarrow \text{R}^*(p,u,w)$ ♦

The axiom ❶ restricts the possible worlds structure to the initial world's connected component (see remark 4.3). It is not necessary in principle, but it can be used to simplify translated formulae somewhat.

It is easy to verify that Ψ_S is really a specification morphism, i.e. it translates well formed modal logic specifications into well formed predicate logic specifications. Since flexible symbols get a new argument, all term declarations have to be updated. This is done with rule (❖) of def. 5.1.

The so defined specification morphism from modal logic to predicate logic permits the use of a standard predicate logic resolution calculus for proving modal logic theorems. To see how this works, let us try to prove the Barcan formula '$\forall x\Box P(x) \Rightarrow \Box\forall x P(x)$' which holds in the decreasing domain case. Since the operators are not indexed, we drop the first argument of the R-predicate. The translated version of the negated formula '$\forall x\Box P(x) \land \Diamond\exists x\neg P(x)$' is therefore:

$$\begin{gathered}
\forall x\ \text{EXISTS}'(0,x) \Rightarrow \forall u\ (\text{R}(0,u) \Rightarrow (\text{EXISTS}'(u,x) \land P'(u,x))\land\\
\exists u\ \text{R}(0,u) \land \exists x\ \text{EXISTS}'(u,x) \land \neg(\text{EXISTS}'(u,x) \land P'(u,x))
\end{gathered}$$

The clause version of the formula is:

C1: $\neg\text{EXISTS}'(0,x) \lor \neg\text{R}(0,u) \lor \text{EXISTS}'(u,x)$
C2: $\neg\text{EXISTS}'(0,x) \lor \neg\text{R}(0,u) \lor P'(u,x)$
C3: $\text{R}(0,a)$
C4: $\text{EXISTS}'(a,b)$
C5: $\neg\text{EXISTS}'(a,b) \lor \neg P'(a,b)$

The translated clause version of the decreasing domain axiom DECREASING-DOMAIN() is:

C0': $\neg\text{R}^*(0,u) \lor \neg\text{R}(u,v) \lor \neg\text{EXISTS}(v,x) \lor \text{EXISTS}(u,x)$

which simplifies with ❶ of def.5.3 to

C0: $\neg\text{R}(u,v) \lor \neg\text{EXISTS}'(v,x) \lor \text{EXISTS}'(u,x)$

A resolution refutation proceeds as follows:

C5,1&C1,3	$\rightarrow$ R1: $\neg\text{EXISTS}'(0,b) \lor \neg\text{R}(0,a) \lor \neg P'(a,b)$
R1,3& C2,3	$\rightarrow$ R2: $\neg\text{EXISTS}'(0,b) \lor \neg\text{R}(0,a)$
C3,&R2;2	$\rightarrow$ R3: $\neg\text{EXISTS}'(0,b)$
C3& C0,1	$\rightarrow$ R4: $\neg\text{EXISTS}'(a,x) \lor \text{EXISTS}'(0,x)$
R3& R4,2	$\rightarrow$ R5: $\neg\text{EXISTS}'(a,b)$
R5&C4	$\rightarrow$ R6: empty clause. ♦

The relational translation method is very flexible because many kinds of accessibility relations can easily be handled. The introduction of the R-literals and the EXISTS'-literals, on the other hand, considerably blows up the number and the size of the translated clauses. Information about chains of accessible worlds is scattered around the whole clause set. Moreover, for the standard resolution strategies there is no difference between the normal predicates and the special predicates. Therefore there is no simple and natural strategy which performs resolutions between R-literals and the EXISTS'-literals only when they are needed to enable a resolution with normal literals. In general, without such a strategy, too many useless resolutions between these literals are possible.

In order to prove soundness and completeness of the relational translation, the first thing to do is to define the interpretation morphism $\Psi_{\mathfrak{I}}$ for translating modal logic models into predicate logic models and vice versa. The proof is listed in full detail because it is prototypic for all the soundness and completeness proofs for logic translators.

Definition 5.4 The Interpretation Morphism $\Psi_{\mathfrak{I}}$
Let $\mathfrak{I}_M = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ be an M-Logic interpretation. The interpretation morphism $\Psi_{\mathfrak{I}}$ constructs a predicate logic interpretation $\mathfrak{I}_P = (\mathcal{M}, \mathcal{V})$ as follows:

- The carrier set of $\mathcal{M}$ and the interpretation of the domain sorts is the same as in $\mathcal{P}(\mathcal{W})$ or in all other worlds respectively. The interpretation of the new sort is $W_{\mathcal{M}} = \mathcal{W}$.
- The interpretation of rigid symbols is the same as in $\mathfrak{I}_M$. The special constant symbol 0 is mapped to w.
- If k is a flexible n-place M-Logic function symbol and $k' = \Psi_\Sigma(k)$ is the corresponding OSPL version, then $k'_{\mathcal{M}}(u,x_1,\ldots,x_n)) =_{\text{def}} k_{\mathcal{P}(u)}(x_1,\ldots,x_n)$ for all $u \in W_{\mathcal{M}}$ and $x_i \in D_{\mathcal{M}}$ where $k_{\mathcal{P}(u)}$ is defined.
- If P is a flexible n-place M-Logic predicate symbol and $P' = \Psi_\Sigma(P)$ is the corresponding OSPL version, then $(u,x_1,\ldots,x_n)) \in P'_{\mathcal{M}}$ iff $(x_1,\ldots,x_n) \in P_{\mathcal{P}(u)}$.
- The interpretation of the R-predicate is:
  $R_{\mathcal{M}}(p,u,v)$ iff $\mathcal{R}_p(u,v)$. $R^*_{\mathcal{M}}$ is the reflexive and transitive closure of $R_{\mathcal{M}}$.

Now let $\mathfrak{I}_P = (\mathcal{M}, \mathcal{V})$ be a predicate logic interpretation for a translated specification. The inverse interpretation morphism $\Psi_{\mathfrak{I}}^{-1}$ constructs an M-Logic interpretation $\mathfrak{I}_M = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ as follows:

- $\mathcal{W} = W_{\mathcal{M}}$, $w = 0_{\mathcal{M}}$.
- $\mathcal{P}$ maps worlds to Σ-structures which are like $\mathcal{M}$ except that the interpretation of flexible symbols in each world u is: $k_{\mathcal{P}(u)}(x_1,\ldots,x_n) =_{\text{def}} k'_{\mathcal{M}}(u,x_1,\ldots,x_n))$ and the interpretation of flexible predicate symbols in each world u is: $(x_1,\ldots,x_n) \in P_{\mathcal{P}(u)}$ iff $(u, x_1,\ldots,x_n)) \in P'_{\mathcal{M}}$.
- $\mathcal{R}_p(u,v)$ iff $R_{\mathcal{M}}(p,u,v)$.
- The interpretation of the ARP-predicates is: $\text{SERIAL}_{\mathcal{P}(w)}(p)$ iff $\mathcal{R}_p(w)$ is serial.
  The other ARP-predicates are interpreted correspondingly. ♦

Theorem 5.5 Soundness of the Translation
If the M-Logic interpretation $\mathfrak{I}_M$ satisfies the M-Logic specification $S = (\Sigma,G)$ then the translated interpretation $\Psi_{\mathfrak{I}}(\mathfrak{I}_M)$ satisfies the translated specification $\Psi_S(S)$.

Proof: Let $\mathfrak{I}_M = (\mathcal{W},\mathcal{R},\mathcal{P},w,\mathcal{V})$ and $\Psi_{\mathfrak{I}}(\mathfrak{I}_M) = \mathfrak{I}_P = (\mathcal{M},\mathcal{V})$. First of all we have to show that $\mathfrak{I}_P$ is really an OSPL-interpretation, i.e. $\mathcal{M}$ is a Σ-structure. The main thing to check is that the sort hierarchy is reflected by corresponding set inclusion relations and that the functions match the corresponding sort declarations. Both properties follow immediately from the facts that $\mathfrak{I}_M$ is an M-Logic interpretation, the interpretations of the domain sorts are not changed and the interpretation of the functions obviously matches the changed term declarations (c.f. def.5.1, ❖).

By induction on the term structure we show for a term t the following property:

If $\Im_P(\mathrm{w}) = w$ then $\Im_M(t) = \Im_P(t_{\mathrm{w}})$. ①

where t_{w} is again an abbreviation for $\Psi_F(t,\mathrm{w})$. The base case, t is a domain variable, is trivial because the assignment of domain variables does not differ between $\Im_M$ and $\Im_P$. Now let $t = k(t_1,\ldots,t_n)$. If k is a rigid symbol, its interpretation is not changed and the induction hypothesis can be applied immediately. If k is flexible we have:

$\Im_P(t_{\mathrm{w}}) = \Im_P(k'(\mathrm{w}, t_{1\mathrm{w}}, \ldots, t_{n\mathrm{w}}))$ (def. of Ψ_F)

$= k'_{\mathcal{M}}(\Im_P(\mathrm{w}), \Im_M(t_1),\ldots,\Im_M(t_n))$ (ind. hyp.)

$= k_{\mathcal{P}(w)}(\Im_M(t_1),\ldots,\Im_M(t_n))$ ($\Im_P(\mathrm{w}) = w$ and def. of $\Psi_\Im$)

$= \Im_M(t)$ (M-Logic semantics)

By induction on the structure of the formula G we show the corresponding property:

If $\Im_P(\mathrm{w}) = w$ and $\Im_M \models_M G$ then $\Im_P \models_P G_{\mathrm{w}}$. ②

Since the initial call to the translation is $\Psi_F(G,0)$ and $\Im_P(0) = 0_{\mathcal{M}} = w$, i.e. the assumption $\Im_P(\mathrm{w}) = w$ holds, this completes the soundness proof.

The base case of the induction is the atomic level. We have to prove ② for atoms with the user defined predicates and with the special ARP-predicates. Now let $\Im_P(\mathrm{w}) = w$ and $\Im_M \models_M G$.

<u>Case</u> G = EXISTS(t)

$\Im_M \models_M \mathrm{EXISTS}(t)$ (assumption)

⇨ $\Im_M(t) \in \mathrm{EXISTS}_{\mathcal{P}(w)}$ (M-Logic semantics)

⇨ $(w, \Im_M(t)) \in \mathrm{EXISTS}'_{\mathcal{M}}$ (def. of $\Psi_\Im$)

⇨ $(\Im_P(\mathrm{w}), \Im_P(t_{\mathrm{w}}))) \in \mathrm{EXISTS}'_{\mathcal{M}}$ (① and assumption)

⇨ $\Im_P \models_P \mathrm{EXISTS}'(\mathrm{w},t_{\mathrm{w}})$ (OSPL semantics)

⇨ $\Im_P \models_P G_{\mathrm{w}}$ (def. of Ψ_F)

<u>Case</u> G = PERSISTENT(t,p)

$\Im_M \models_M \mathrm{PERSISTENT}(t,p)$ (assumption)

⇨ $\Im_M(t) \in \mathrm{EXISTS}_{\mathcal{P}(u)}$ for all $u \in \mathcal{R}_{\Im(p)}(w)$ (M-Logic semantics)

⇨ $(u, \Im_M(t)) \in \mathrm{EXISTS}'_{\mathcal{M}}$ for all all $u \in \mathcal{R}_{\Im(p)}(w)$ (def. of $\Psi_\Im$)

⇨ $\forall u\, R^*_{\mathcal{M}}(\Im_M(p), w, u)$ ⇨ $(u, \Im_M(t)) \in \mathrm{EXISTS}'_{\mathcal{M}}$ (semantics of R*)

⇨ $\forall u\, R^*_{\mathcal{M}}(\Im_P(p_{\mathrm{w}}), w, u)$ ⇨ $(\Im_P(u), \Im_P(t_{\mathrm{w}})) \in \mathrm{EXISTS}'_{\mathcal{M}}$ (① and assumption)

⇨ $\Im_P \models_P \forall u{:}W\ R^*(p_{\mathrm{w}},\mathrm{w},u) \Rightarrow \mathrm{EXISTS}'(u,t_{\mathrm{w}})$ (OSPL semantics)

⇨ $\Im_P \models_P G_{\mathrm{w}}$ (def. of Ψ_F)

<u>Case</u> G = $P(t_1,\ldots,t_n)$

$\Im_M \models_M P(t_1,\ldots,t_n)$ (assumption)

⇨ $\Im_M(t_i) \in \mathrm{EXISTS}_{\mathcal{P}(w)}$ for i = 1,…,n and $(\Im_M(t_1),\ldots,\Im_M(t_n)) \in P_{\mathcal{P}(w)}$ (M-Logic semantics)

⇨ $\Im_P \models_P \mathrm{EXISTS}'(\mathrm{w},t_{i\mathrm{w}})$ for i = 1,…,n (previous case)

and $(w, \Im_M(t_1),\ldots,\Im_M(t_n)) \in P'_{\mathcal{M}}$ (def. of $\Psi_\Im$)

⇨ $\Im_P \models_P \mathrm{EXISTS}'(\mathrm{w},t_{i\mathrm{w}})$ for i = 1,…,n

and $(\Im_P(\mathrm{w}), \Im_P(t_{1\mathrm{w}}),\ldots,\Im_P(t_{n\mathrm{w}})) \in P'_{\mathcal{M}}$ (① and assumption)

⇨ $\Im_P \models_P (\mathrm{EXISTS}'(\mathrm{w},t_{1\mathrm{w}}) \wedge \ldots \wedge \mathrm{EXISTS}'(\mathrm{w},t_{n\mathrm{w}}) \wedge P'(\mathrm{w},t_{1\mathrm{w}},\ldots,t_{n\mathrm{w}}))$ (OSPL semantics)

⇨ $\Im_P \models_P G_{\mathrm{w}}$ (def. of Ψ_F)

<u>Case</u> G = SERIAL(p)

$\Im_M \models_M \mathrm{SERIAL}(p)$ (assumption)

⇨ $\mathcal{R}_{\Im_M(p)}(w)$ is serial (M-Logic semantics)

⇨ $\Im_M(p) \in \mathrm{EXISTS}_{\mathcal{P}(w)}$ and (otherwise $\mathcal{R}_{\Im_M(p)}(w)$ is not defined)

$\forall u\, \mathcal{R}^*_{\Im_M(p)}(w, u)$ ⇨ $\exists v\, \mathcal{R}_{\Im_M(p)}(u,v)$ (def. of seriality)

⇨ $\Im_P \models_P \mathrm{EXISTS}'(w,p_w)$ and (first case)
$\forall u\, R^*_{\mathcal{M}}(\Im_M(p),w,u)$ ⇨ $\exists v\, R_{\mathcal{M}}(\Im_M(p),u,v)$ (def. of $\Psi_\Im$)
⇨ … and $\forall u\, R^*_{\mathcal{M}}(\Im_P(p_w),\Im_P(w),u)$ ⇨ $\exists v\, R_{\mathcal{M}}(p_w,u,v)$ (① and assumption)
⇨ … and $\forall u \in W_{\mathcal{M}}\ (\Im_P[u/u] \models_P R^*(p_w,w,u))$ ⇨
$(\exists v \in W_{\mathcal{M}}\, \Im_P[u/u,v/v] \models_P R(p_w),u,v))$ (OSPL semantics)
⇨ … and $\Im_P \models_P (\forall u{:}W\ R^*(p_w,w,u) \Rightarrow \exists v{:}W\ R(p_w),u,v))$ (OSPL semantics)
⇨ $\Im_P \models_P \mathrm{EXISTS}'(w,p_w) \wedge (\forall u{:}W\ R^*(p_w,w,u) \Rightarrow \exists v{:}W\ R(p_w),u,v))$
⇨ $\Im_P \models_P G_w$ (def. of Ψ_F)

The proofs for the cases with the other ARP-predicates are similar.

Induction Step: The induction hypothesis is: if $\Im_P(w) = w$ and $\Im_M \models_M F$ then $\Im_P \models_P F_w$.

Case $G = \forall x{:}S\ F$
$\Im_M \models_M \forall x{:}S\ F$ (assumption)
⇨ $\forall \chi \in S_{\mathcal{P}(w)} \cap \mathrm{EXISTS}_{\mathcal{P}(w)}{:}\ \Im_M[x/\chi] \models_M F$ (M-Logic semantics)
⇨ $\forall \chi\, \chi \in S_{\mathcal{M}} \wedge (\Im_P(w), \Im_P[x/\chi](x)) \in \mathrm{EXISTS}'_{\mathcal{M}}$ ⇨ $\Im_P[x/\chi] \models_P F_w$ (case EXISTS and ind.hyp.)
⇨ $\Im_P \models_P \forall x{:}S\ \mathrm{EXISTS}'(w,x) \Rightarrow F_w$ (OSPL semantics)
⇨ $\Im_P \models_P G_w$ (def. of Ψ_F)

Case $G = \exists x{:}S\ F$. This case is analogous to the previous one.

Case $G = \Box_p F$
$\Im_M \models_M \Box_p F$ (assumption)
⇨ $\forall u\, \mathcal{R}_{\Im_M(p)}(w,u)$ ⇨ $\Im_M[u] \models_M F$ (M-Logic semantics)
⇨ $\forall u\, R_{\mathcal{M}}(\Im_M(p),w,u)$ ⇨ $\Im_M[u] \models_M F$ (def. of $\Psi_\Im$)
⇨ $\forall u\, R_{\mathcal{M}}(\Im_P(p_w),\Im_P(w),u)$ ⇨ $\Im_M[u] \models_M F$ (① and assumption)
⇨ $\forall u\, \Im_P[u/u] \models_P R(p_w,w,u)$ ⇨ $\Im_P[u/u] \models_P F_u$ (ind. hyp. and $\Im_P[u/u](u) = u$)
⇨ $\Im_P \models_P \forall u{:}W\ R(p_w,w,u) \Rightarrow F_u$ (OSPL semantics)
⇨ $\Im_P \models_P G_w$ (def. of Ψ_F)

Case $G = \Diamond_p F$. This case is analogous to the previous one.

The cases with normal logical connectives are straightforward applications of the induction hypothesis.

To complete the soundness proof, it has to be shown that the three axioms ❶,❷ and ❸ for R*, added by the specification morphism Ψ_S, are satisfied by $\Im_P$. This is obvious for ❷ and ❸. ❶ is a consequence of the fact that only the part of the possible worlds structure needs to be considered which is accessible in finitely many steps from the initial world (c.f. remark 4.3). ♦

Theorem 5.6 Completeness of the Translation
If the OSPL interpretation $\Im_P$ satisfies a translated specification $\Psi_S((\Sigma,G))$ then the back translated interpretation $\Im_M = \Psi_\Im^{-1}(\Im_P)$ satisfies the original specification (Σ,G).

Proof: Let $\Im_P = (\mathcal{M},\mathcal{V})$ and $\Im_M = \Psi_\Im^{-1}(\Im_P) = (\mathcal{W},\mathcal{R},\mathcal{P},w,\mathcal{V})$. First of all we have to show that $\Im_M$ is really an M-Logic interpretation. This is even more trivial than in the soundness proof and therefore skipped. In the same way as in the soundness proof the property ①:

If $\Im_P(w) = w$ then $\Im_M(t) = \Im_P(\Psi_F(t, w))$

can be shown. By induction on the structure of the formula G the corresponding property can then be shown:

If $\Im_P(w) = w$ and $\Im_P \models_P \Psi_{\mathcal{F}}(G,w)$ then $\Im_M \models_M G$

Because the initial call to the translation algorithm is $\Psi_F(G,0)$ and since $\Im_P(0) = w$ this proves the theorem.
In principle all proof steps for this induction are the same as the corresponding proof steps of the

soundness proof, however read backwards. There is only one small difficulty. The characteristic axioms for the R*-predicate (def. 5.3) specify a superset of the reflexive and transitive closure of the $\mathcal{R}$-relation[6]. R* plays a rôle in the translation of the ARP-predicates. The argument here is, what holds for worlds accessible in a superset of the reflexive and transitive closure of $\mathcal{R}$ holds in particular for the subsets of worlds accessible in the reflexive and transitive closure itself. Therefore nothing is lost in the step back from the translated formula to the original formula. ♦

6 FUNCTIONAL TRANSLATION

As already said, the relational translation is applicable to many variants of modal logic, but the efficiency of the resulting calculus[7] is not very good. In order to improve its efficiency, a representation of the information about accessible worlds has to be found which permits the reasoning about whole chains of possible worlds in a single step. Resolution between normal literals should only be possible when the two resolution literals are interpreted definitely in the same world. To this end, a new translation technique has been developed where the relevant information about the chain of worlds leading to the actual world is concentrated in one single term and reasoning about possible worlds can be done with unification algorithms operating on these terms.

The basic idea of the functional translation has already been explained in the introduction. It uses the fact that a binary relation can be represented as the domain - range relation of a set of one place functions. On the syntactic level this means that a concatenation of terms $t_1 \circ \ldots \circ t_n$, each of which denoting a function mapping worlds to accessible worlds, can be interpreted as a composition of such **context access functions**. Applied to the initial world, this composition describes a path through the possible worlds structure where the last element serves as the actual world in which a term or atom is to be interpreted. A formula $\Box\Diamond Q$ is now translated into '$\forall f\, \exists g\; Q(\downarrow(f \circ g, 0))$'[8] where $f \circ g$ denotes a composition of two context access functions and $\downarrow(f \circ g, 0)$ represents the application of the composed function to the initial world.

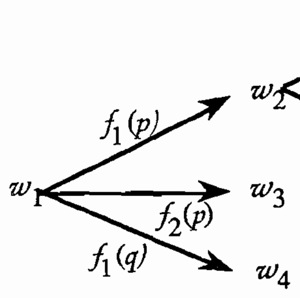
There are two problems with this kind of translation. The first problem comes from the parametrized modal operators. In '$\Box_p Q$' for example, the $\Box$-operator does not denote all accessible worlds, but only those, accessible via p-labelled transitions. In order to describe this with functions, we need context access functions depending additionally on domain elements. Therefore, besides the sort 'W→W', a sort 'D,W→W' is introduced whose interpretation in an algebra $\mathcal{A}$ is '$D,W \to W$'$_{\mathcal{A}} \subseteq D_{\mathcal{A}} \times W_{\mathcal{A}} \to W_{\mathcal{A}}$, i.e. 'D,W→W' denotes two-place functions which depend on domain elements (parameters) and worlds. Now '$\Box_p \Diamond_q Q$' can be translated into '$\forall f$:'D,W→W' $\exists g$:'D,W→W' $Q(\downarrow(\downarrow(f,p) \circ \downarrow(g,q), 0))$'. The axiomatization of the apply function $\downarrow$ and the composition function $\circ$ guarantees that a term '$\downarrow(f,p)$' means $f(p)$ where $f \in D_{\mathcal{A}} \times W_{\mathcal{A}} \to W_{\mathcal{A}}$ such that $f(p) \in W_{\mathcal{A}} \to W_{\mathcal{A}}$ is obtained. Thus, '$\downarrow(f,p)$' denotes an ordinary context access function and the term $\downarrow(f,p) \circ \downarrow(g,q)$ denotes again a path through the possible worlds structure.

[6] The transitive closure of a relation is not finitely axiomatizable in first order logic.

[7] The efficiency of a calculus is usually measured in the branching rate of the search space it generates.

[8] This time again seriality of the accessibility relation is assumed.

A graphical picture of parametrized context access functions:



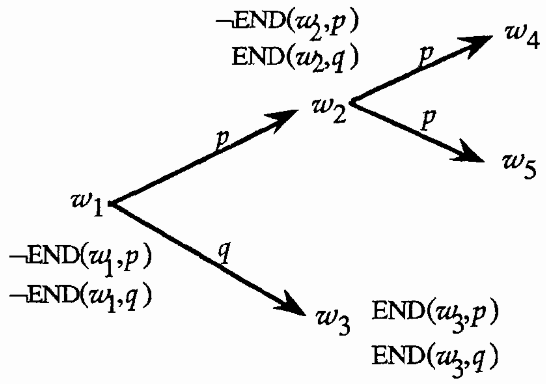
$f_1(p)$: $w_1 \rightarrow w_2$ $f_2(p)$: $w_1 \rightarrow w_3$
$w_2 \rightarrow w_5$ ...

$f_1(q)$: $w_1 \rightarrow w_4$
$w_2 \rightarrow w_6$
...

For the translation of the ARP-predicates the auxiliary sorts '$W\xrightarrow{*}W$' and '$D,W\xrightarrow{*}W$' are needed. They denote functions mapping worlds to worlds accessible in the reflexive and transitive closure of the basic accessibility relation. In the unparametrized case, for example 'REFLEXIVE()' is translated into '$\forall f$:'$W\xrightarrow{*}W$' $\exists g$:'$W\xrightarrow{*}W$' $\downarrow(f\circ g,0) = \downarrow(f,0)$'. The quantification '$\forall f$:'$W\xrightarrow{*}W$'...' accesses all worlds at all. Therefore this formula can be read: for all worlds there exists a context access function which maps the world to itself; and this is nothing else than reflexivity. (We shall see that it makes no difference whether the function g denotes really the identity function or depends on the world.)

The second problem in the functional translation comes from the fact that in non serial possible worlds structures there may be "end worlds", i.e. worlds where there is no further accessible world at all. In these structures context access functions are necessarily partial functions. Unfortunately partial functions cannot be handled in standard predicate logic. To overcome this restriction, partial functions have to be made total by the usual (strict) ω-extension mechanism [Loecks & Sieber 84] which adds an artificial bottom element $\perp$ and maps all arguments for which the function is not defined to the bottom element. On the syntactic side we introduce a special predicate END: W×D such that $\text{END}_{\mathcal{A}}(w,p)$ is true in an Σ-structure $\mathcal{A}$ if w is an end world for parameter p, i.e. from w there is no p-accessible world.

Example:



A formula '$\Box_p Q$' is now translated into '$\forall f$:'$D,W\rightarrow W$' $\neg\text{END}(0,p) \Rightarrow Q(\downarrow(\downarrow(f,p),0))$' with the meaning: if the initial world is not the end world for parameter p, Q holds in all p-accessible worlds.

We now formally define a "functional" logic morphism Φ which maps modal logic specifications to *functional* OSPL-specifications such that satisfiability and unsatisfiability is preserved. The definitions and proofs follow exactly the same schema as the corresponding parts in the last chapter. In the sequel the letters f,g,h,i and j are used as variable symbols for denoting context access functions.

Definition 6.1 The Signature Morphism Φ_Σ
Φ_Σ generates a functional OSPL signature from an M-Logic signature with top domain sort D as follows:

- The domain sort hierarchy remains unchanged.
  Five additional **context sort** symbols are introduced:
  W, '$W \to W$', '$W \overset{*}{\to} W$', '$D,W \to W$', '$D,W \overset{*}{\to} W$'
  The subsort relationships are: '$W \to W$' $\sqsubseteq$ '$W \overset{*}{\to} W$' and '$D,W \to W$' $\sqsubseteq$ '$D,W \overset{*}{\to} W$'
- The symbols specific for functional OSPL specifications, composition $\circ$ and application $\downarrow$ (def. 3.10) are introduced:
  $\circ$: '$W \overset{*}{\to} W$' $\times$ '$W \overset{*}{\to} W$' $\to$ '$W \overset{*}{\to} W$' $\quad\quad$ $\downarrow$: $D \times$ '$D,W \to W$' $\to$ '$W \to W$'
  '$D,W \overset{*}{\to} W$' $\times$ '$D,W \overset{*}{\to} W$' $\to$ '$D,W \overset{*}{\to} W$' $\quad\quad$ $D \times$ '$D,W \overset{*}{\to} W$' $\to$ '$W \overset{*}{\to} W$'
- Φ_Σ translates variable, function and predicate symbols in the same way as Ψ_Σ (def. 5.1).
- The following constant, function and predicate symbols are added: $0{:}W$, id:'$W \overset{*}{\to} W$', END:$W \times D$ ♦

Definition 6.2 The Formula Morphism Φ_F
For a term or formula F let $F_w =_{def} \Phi_F(F,w)$ and let $w' =_{def} \downarrow(w,0)$.

- $x_w = x$ if x is a variable.
- $f(t_1,\ldots,t_n)_w = f'(w',t_{1w},\ldots,t_{nw})$ if f is an n-place flexible function symbol.
- $\text{EXISTS}(t)_w = \text{EXISTS}'(w',t_w)$
- $\text{PERSISTENT}(t,p)_w = \forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \text{EXISTS}'(\downarrow(w\circ\downarrow(f,p_w),0),t_w)$.
- $\text{SERIAL}(p)_w = \text{EXISTS}'(w',p_w) \wedge \forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \neg\text{END}(\downarrow(w\circ\downarrow(f,p_w),0),p_w)$
- $\text{REFLEXIVE}(p)_w = \text{EXISTS}'(w',p_w) \wedge$
  $\forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \exists g{:}\text{'}D,W \to W\text{'}\ \downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w),0) = \downarrow(w\circ\downarrow(f,p_w),0)$
- $\text{TRANSITIVE}(p)_w = \text{EXISTS}'(w',p_w) \wedge$
  $\forall f,g{:}\text{'}D,W \to W\text{'}\ \exists h{:}\text{'}D,W \to W\text{'}\ \downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w),0) = \downarrow(w\circ\downarrow(h,p_w),0)$
- $\text{SYMMETRIC}(p)_w = \text{EXISTS}'(w',p_w) \wedge \forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \forall g{:}\text{'}D,W \to W\text{'}\ \exists h{:}\text{'}D,W \to W\text{'}$
  $\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w)\circ\downarrow(h,p_w),0) = \downarrow(w\circ\downarrow(f,p_w),0)$
- $\text{EUCLIDEAN}(p)_w = \text{EXISTS}'(w',p_w) \wedge \forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \forall g,h{:}\text{'}D,W \to W\text{'}\ \exists k{:}\text{'}D,W \to W\text{'}$
  $\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w)\circ\downarrow(k,p_w),0) = \downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(h,p_w),0)$
- $\text{LINEAR}(p)_w = \text{EXISTS}'(w',p_w) \wedge \forall f,g{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \exists h{:}\text{'}D,W \overset{*}{\to} W\text{'}$
  $\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(h,p_w),0) = \downarrow(w\circ\downarrow(g,p_w),0) \vee$
  $\downarrow(w\circ\downarrow(g,p_w)\circ\downarrow(h,p_w),0) = \downarrow(w\circ\downarrow(f,p_w),0)$
- $\text{INCREASING-DOMAIN}(p)_w = \text{EXISTS}'(w',p_w) \wedge \forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \forall g{:}\text{'}D,W \to W\text{'}\ \forall x{:}D$
  $\text{EXISTS}'(\downarrow(w\circ\downarrow(f,p_w),0),x) \Rightarrow \text{EXISTS}'(\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w),0),x)$
- $\text{DECREASING-DOMAIN}(p)_w = \text{EXISTS}'(w',p_w) \wedge \forall f{:}\text{'}D,W \overset{*}{\to} W\text{'}\ \forall g{:}\text{'}D,W \to W\text{'}$
  $\text{EXISTS}'(\downarrow(w\circ\downarrow(f,p_w),0),x) \Leftarrow \text{EXISTS}'(\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w),0),x)$
- $P(t_1,\ldots,t_n)_w = \text{EXISTS}'(w',t_{1w}) \wedge \ldots \wedge \text{EXISTS}'(w',t_{nw}) \wedge P'(w',t_{1w},\ldots,t_{nw})$
  if P is a flexible predicate symbol.
- $(\forall x\ F)_w = \forall x\ \text{EXISTS}'(w',x) \Rightarrow F_w$
- $(\exists x\ F)_w = \exists x\ \text{EXISTS}'(w',x) \wedge F_w$
- $(\Box_p\ F)_w = \text{EXISTS}'(w',p_w) \Rightarrow (\neg\text{END}(w',p_w) \Rightarrow \forall f{:}\text{'}D,W \to W\text{'}\ F_{w\circ\downarrow(f,p_w)})$
- $(\Diamond_p\ F)_w = \text{EXISTS}'(w',p_w) \wedge \neg\text{END}(w',p_w) \wedge \exists f{:}\text{'}D,W \to W\text{'}\ F_{w\circ\downarrow(f,p_w)})$
- For all other cases Φ_F is the usual homomorphic extension.

The top level call of the translation algorithm is: $\Phi_F(\mathcal{F}, id)$. In the translated formulae, terms $\downarrow(id,0)$ are rewritten to '0' and terms 'id$\circ$something' and 'something$\circ$id' are rewritten to something. Hence 'id' disappears completely. ♦

Example:

$\Phi_F(\forall x\ \Box_{k(x)}\ P(x),id)$

$= \ (\forall x\ \Box_{k(x)}\ P(x))_{id}$

$= \ \forall x\ \mathrm{EXISTS}'(\downarrow(id,0),x) \Rightarrow (\Box_{k(x)}\ P(x))_{id}$

$= \ \forall x\ \mathrm{EXISTS}'(\downarrow(id,0),x) \Rightarrow (\mathrm{EXISTS}'(\downarrow(id,0),\ k(x)_{id}) \Rightarrow (\neg\mathrm{END}(\downarrow(id,0),\ k(x)_{id})$
$\qquad \Rightarrow \forall f{:}\text{'D,W}\rightarrow\text{W'}\ (P(x))_{id\circ\downarrow(f,k(x)_{id})}))$

$= \ \forall x\ \mathrm{EXISTS}'(\downarrow(id,0),x) \Rightarrow (\mathrm{EXISTS}'(\downarrow(id,0),\ k(\downarrow(id,0),x)) \Rightarrow (\neg\mathrm{END}(\downarrow(id,0),\ k(\downarrow(id,0),x))$
$\qquad \Rightarrow \forall f{:}\text{'D,W}\rightarrow\text{W'}\ P(\downarrow(id\circ\downarrow(f,k(\downarrow(id,0),x)),0),x))$

$\rightarrow \ \forall x\ \mathrm{EXISTS}'(0,x) \Rightarrow (\mathrm{EXISTS}'(0,\ k(0,x)) \Rightarrow (\neg\mathrm{END}(0,\ k(0,x))$
$\qquad \Rightarrow \forall f{:}\text{'D,W}\rightarrow\text{W'}\ P(\downarrow(\downarrow(f,k(0,x)),0),x))$ ♦

Definition 6.3 The Specification Morphism Φ_S

The specification morphism Φ_S uses Φ_Σ for translating M-Logic signatures into OSPL-signatures and Φ_F for translating M-Logic formulae into OSPL-formulae. Furthermore, it adds the necessary axioms for the application function $\downarrow$ and the composition function $\circ$ (def. 3.10,5) to generate a functional specification (def. 3.10). And finally it adds the axioms which characterize the END predicate and the 'D,W$\xrightarrow{*}$W' sort. (We sometimes use a second order syntax to make the axioms more readable. The first-order version of terms like x(y) is $\downarrow$(x,y)).

Characterization of $\circ$ and $\downarrow$:

① $\forall f,g{:}\text{'W}\xrightarrow{*}\text{W'}\ \forall w{:}W \qquad f(w) = g(w) \Rightarrow f = g$

② $\forall f,g{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall p{:}D\ \forall w{:}W \qquad \downarrow(f,\ p)(w) = \downarrow(g,\ p)(w) \Rightarrow f = g$

③ $\forall f,g{:}\text{'W}\xrightarrow{*}\text{W'}\ \forall w{:}W \qquad \downarrow(f \circ g,\ w) = \downarrow(g,\ \downarrow(f,\ w))$

④ $\forall f,g{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall p{:}D \qquad \downarrow(f \circ g,\ p) = \downarrow(f,\ p) \circ \downarrow(g,\ p)$

Characterization of the END-predicate:

⑤ $\forall f{:}\text{'W}\xrightarrow{*}\text{W'}\ g{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall p{:}D \qquad \mathrm{END}(\downarrow(f\circ\downarrow(g,p),0),\ p) \Rightarrow \neg\mathrm{END}(\downarrow(f,0),p)$

Characterization of the 'D,W$\xrightarrow{*}$W' sort:

⑥ $\exists f{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall p{:}D\ \forall w{:}W \qquad \downarrow(f,p)(w) = w$ (reflexivity)

⑦ $\forall f,g{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \exists h{:}\text{'D,W}\xrightarrow{*}\text{W'} \qquad \downarrow(f,p)\circ\downarrow(g,p) = \downarrow(h,p)$ (transitivity) ♦

It is straightforward to verify that Φ_S is really a specification morphism, i.e. it translates well formed M-Logic specifications into well formed (well sorted) OSPL-specifications.

Definition 6.4 The Interpretation Morphism

Let $\Im_M = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ be an M-Logic interpretation. The interpretation morphism $\Phi_\Im$ constructs a functional predicate logic interpretation $\Im_P = (\mathcal{M}, \mathcal{V})$ as follows:

- The carrier set of $\mathcal{M}$ and the interpretation of the domain sorts is the same as in $\mathcal{P}(w)$ or in all other worlds respectively.

The interpretation of the additional symbols is:

- $W_{\mathcal{M}} = \mathcal{W} \cup \{\perp\}$[9]

[9] The bottom element $\perp$ is necessary to make the context access functions total.

- $\text{'D,W}\rightarrow\text{W'}_{\mathcal{M}} = \{f \in D_{\mathcal{M}} \times W_{\mathcal{M}} \rightarrow W_{\mathcal{M}} \mid \forall u \in W_{\mathcal{M}} \forall p \in D_{\mathcal{M}}$ if $\exists v\, \mathcal{R}_p(u, v)$ then $f(p,u) \neq \bot$ and $f(p,u) \neq \bot \Rightarrow \mathcal{R}_p(u, f(p,u))\}$.
- $\text{'D,W}\xrightarrow{*}\text{W'}_{\mathcal{M}} = \{f \in D_{\mathcal{M}} \times W_{\mathcal{M}} \rightarrow W_{\mathcal{M}} \mid \forall p \in D_{\mathcal{M}}\, f(p) =$ identity or $f(p) = f_1(p) \circ \ldots \circ f_n(p)$ for some $f_i \in \text{'D,W}\rightarrow\text{W'}_{\mathcal{M}}\}$.
- $\text{'W}\rightarrow\text{W'}_{\mathcal{M}} = \{g \in W_{\mathcal{M}} \rightarrow W_{\mathcal{M}} \mid \forall u \in W_{\mathcal{M}} \exists f \in \text{'D,W}\rightarrow\text{W'}_{\mathcal{M}} \exists p \in D_{\mathcal{M}}\, g(u) = f(p,u)\}$.
- $\text{'W}\xrightarrow{*}\text{W'}_{\mathcal{M}} = \{g \in W_{\mathcal{M}} \rightarrow W_{\mathcal{M}} \mid g =$ identity or $g = g_1 \circ \ldots \circ g_n$ for some $g_i \in \text{'W}\rightarrow\text{W'}_{\mathcal{M}}\}$.
- $\text{id}_{\mathcal{M}}$ is the identity function on $W_{\mathcal{M}}$.
- $0_{\mathcal{M}} = w$
- $\downarrow_{\mathcal{M}}$ is the application function, i.e. $\forall p \in D_{\mathcal{M}} \forall f \in \text{'D,W}\xrightarrow{*}\text{W'}_{\mathcal{M}}\ \downarrow_{\mathcal{M}}(f, p) = f(p)$.
- $\circ_{\mathcal{M}}$ is the composition function, i.e.
  $\forall f_1, f_2 \in \text{'W}\xrightarrow{*}\text{W'}_{\mathcal{M}} \forall u \in W_{\mathcal{M}}$ $(f_1 \circ_{\mathcal{M}} f_2)(u) = f_2(f_1(u))$ and
  $\forall f_1, f_2 \in \text{'D,W}\xrightarrow{*}\text{W'}_{\mathcal{M}} \forall u \in W_{\mathcal{M}} \forall p \in D_{\mathcal{M}}$ $(f_1 \circ_{\mathcal{M}} f_2)(p,u) = f_2(p)\,(f_1(p)(u))$.

In the sequel we omit the index $_{\mathcal{M}}$ for $\circ_{\mathcal{M}}$ and $\downarrow_{\mathcal{M}}$.

- The interpretation of the rigid symbols in $\Im_P$ is the same as in $\Im_M$.
- The interpretation of flexible function symbols is:
  $f'_{\mathcal{M}}(u, x_1,\ldots,x_n)) = f_{\mathcal{P}(u)}(x_1,\ldots,x_n)$ where $f' = \Phi_\Sigma(f)$
  and the interpretation of flexible predicate symbols is:
  $(u, x_1,\ldots,x_n) \in P'_{\mathcal{M}}$ iff $(x_1,\ldots,x_n) \in P_{\mathcal{P}(u)}$ where $P' = \Phi_\Sigma(P)$.
- $\text{END}_{\mathcal{M}}(u,p)$ iff there is no world v with $\mathcal{R}_p(u,v)$.

Now let $\Im_P = (\mathcal{M}, \mathcal{V})$ be a predicate logic interpretation for a translated specification. The inverse interpretation morphism $\Psi_\Im^{-1}$ constructs an M-Logic interpretation $\Im_M = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ as follows:

- $\mathcal{W}$ is constructed iteratively:
  $\mathcal{W}_0 = \{0_{\mathcal{M}}\}$
  $\mathcal{W}_i = \{f(p,u) \mid f \in \text{'D,W}\rightarrow\text{W'}_{\mathcal{M}},\ p \in D_{\mathcal{M}},\ u \in \mathcal{W}_{i-1}$ and not $\text{END}_{\mathcal{M}}(u,p)\}$
  $\mathcal{W} = \bigcup_{i=0}^{\infty} \mathcal{W}_i$.
- $\mathcal{P}$ maps worlds to Σ-structures which are like $\mathcal{M}$ except that the interpretation of flexible function symbols in each world u is: $f_{\mathcal{P}(u)}(x_1,\ldots,x_n) = f'_{\mathcal{M}}(u, x_1,\ldots,x_n))$ where $f' = \Phi_\Sigma(f)$,
  and the interpretation of flexible predicate symbols in each world u is:
  $(x_1,\ldots,x_n) \in P_{\mathcal{P}(u)}$ iff $(u, x_1,\ldots,x_n) \in P'_{\mathcal{M}}$ where $P' = \Phi_\Sigma(P)$.
- $\forall u,v \in \mathcal{W}\, \mathcal{R}_p(u,v)$ iff $\exists f \in \text{'D,W}\rightarrow\text{W'}_{\mathcal{M}}$: $v = f(p,u)$
- $w = 0_{\mathcal{M}}$ ♦

Theorem 6.5 Soundness of the Translation
If the M-Logic interpretation $\Im_M$ satisfies the M-Logic specification $S = (\Sigma, G)$ then the translated interpretation $\Phi_\Im(\Im_M)$ satisfies the translated specification $\Phi_S(S)$.

Proof: Let $\Im_M = (\mathcal{W}, \mathcal{R}, \mathcal{P}, w, \mathcal{V})$ and $\Phi_\Im(\Im_M) = \Im_P = (\mathcal{M}, \mathcal{V})$. First of all we have to show that $\Im_P$ is really an OSPL-interpretation, i.e. $\mathcal{M}$ is a Σ-structure. The main thing to check here is that the interpretations of the sorts, in particular of the functional sorts, are not empty, that the sort hierarchy is reflected by corresponding set inclusion relations and that the functions match the corresponding sort declarations. The checks are straightforward and therefore skipped.

In the sequel let again $w' =_{\text{def}} \downarrow(w,0)$ and $t_w =_{\text{def}} \Phi_F(t,w)$.

By induction on the term structure we show exactly in the same way as in theorem 5.5 for a term t the following property: If $\Im_P(w') = w$ then $\Im_M(t) = \Im_P(t_w)$. ①
By induction on the structure of the formula G we show the corresponding property:

If $\mathfrak{I}_P(w') = \mathcal{w}$ and $\mathfrak{I}_M \models_M G$ then $\mathfrak{I}_P \models_P G_w$. ②

Since the initial call to the translation is $\Phi_F(\mathcal{G}, \text{id})$ and $\mathfrak{I}_P(0) = 0_{\mathcal{M}} = \mathcal{w}$, and $\text{id}_{\mathcal{M}}$ = identity i.e. the assumption $\mathfrak{I}_P(w') = \mathcal{w}$ holds, this completes the soundness proof.

The base case of the induction is the atomic level. We have to prove ② for atoms with the user defined predicates and with the special ARP-predicates. Now let $\mathfrak{I}_P(w') = \mathcal{w}$ and $\mathfrak{I}_M \models_M G$.

<u>Case</u> G = EXISTS(t)

$\mathfrak{I}_M \models_M$ EXISTS(t) (assumption)
$\Rightarrow$ $\mathfrak{I}_M(t) \in \text{EXISTS}_{\mathcal{F}(\mathcal{w})}$ (M-Logic semantics)
$\Rightarrow$ $(\mathcal{w}, \mathfrak{I}_M(t)) \in \text{EXISTS}'_{\mathcal{M}}$ (def. of $\Phi_{\mathfrak{I}}$)
$\Rightarrow$ $(\mathfrak{I}_P(w'), \mathfrak{I}_P(t_w)) \in \text{EXISTS}'_{\mathcal{M}}$ (① and assumption)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'(w', t_w) (OSPL semantics)
$\Rightarrow$ $\mathfrak{I}_P \models_P G_w$. (def. of Φ_F)

<u>Case</u> G = PERSISTENT(t)

$\mathfrak{I}_M \models_M$ PERSISTENT(t) (assumption)
$\Rightarrow$ $\mathfrak{I}_M(t) \in \text{EXISTS}_{\mathcal{F}(u)}$ for all all $u \in \mathcal{R}_{\mathfrak{I}(p)}(\mathcal{w})$ (M-Logic semantics)
$\Rightarrow$ $\forall u\, \mathcal{R}^*_{\mathfrak{I}_M(p)}(\mathcal{w}, u) \Rightarrow (u, \mathfrak{I}_M(t)) \in \text{EXISTS}'_{\mathcal{M}}$ (def. of $\Phi_{\mathfrak{I}}$ and $\mathcal{R}^*$)
$\Rightarrow$ $\forall f \in$ 'D,W$\xrightarrow{*}$W'$_{\mathcal{M}}(f(\mathfrak{I}_P(p_w), \mathcal{w}), \mathfrak{I}_P(t_w)) \in \text{EXISTS}'_{\mathcal{M}}$
(①, def. of 'D,W$\xrightarrow{*}$W'$_{\mathcal{M}}$ and assumption)
$\Rightarrow$ $\mathfrak{I}_P \models_P \forall f$:'D,W$\xrightarrow{*}$W' EXISTS'$(\downarrow(w \circ \downarrow(f, p_w), 0), p_w)$ (OSPL semantics)
$\Rightarrow$ $\mathfrak{I}_P \models_P G_w$. (def. of Φ_F)

<u>Case</u> G = P$(t_1, \ldots, t_n)$

$\mathfrak{I}_M \models_M$ P$(t_1, \ldots, t_n)$ (assumption)
$\Rightarrow$ $\mathfrak{I}_M(t_i) \in \text{EXISTS}_{\mathcal{F}(\mathcal{w})}$ for i = 1,…,n and
$(\mathfrak{I}_M(t_1), \ldots, \mathfrak{I}_M(t_n)) \in P_{\mathcal{F}(\mathcal{w})}$ (M-Logic semantics)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'(w', t_{iw}) for i = 1,…,n (previous case)
and $(\mathcal{w}, \mathfrak{I}_M(t_1), \ldots, \mathfrak{I}_M(t_n)) \in P'_{\mathcal{M}}$ (def. of $\Phi_{\mathfrak{I}}$)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'(w', t_{iw}) for i = 1,…,n
and $(\mathfrak{I}_P(w'), \mathfrak{I}_P(t_{1w}), \ldots, \mathfrak{I}_P(t_{nw})) \in P'_{\mathcal{M}}$ (① and assumption)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'$(w', t_{1w}) \wedge \ldots \wedge$ EXISTS'$(w', t_{nw}) \wedge$ P$(w', t_{1w}, \ldots, t_{nw})$ (OSPL semantics)
$\Rightarrow$ $\mathfrak{I}_P \models_P G_w$. (def. of Φ_F)

<u>Case</u> G = SERIAL(p)

$\mathfrak{I}_M \models_M$ SERIAL(p) (assumption)
$\Rightarrow$ $\mathcal{R}_{\mathfrak{I}_M(p)}(\mathcal{w})$ is serial (M-Logic semantics)
$\Rightarrow$ $\mathfrak{I}_M(p)$ exists in $\mathcal{w}$ and (otherwise $\mathcal{R}_{\mathfrak{I}_M(p)}(\mathcal{w})$ is not defined)
$\forall u\, \mathcal{R}^*_{\mathfrak{I}_M(p)}(\mathcal{w}, u) \Rightarrow \exists v\, \mathcal{R}_{\mathfrak{I}_M(p)}(u, v)$ (def. of seriality)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'(w', p_w) and $\forall u\, \mathcal{R}^*_{\mathfrak{I}_M(p)}(\mathcal{w}, u) \Rightarrow$ not $\text{END}_{\mathcal{M}}(u, \mathfrak{I}_M(p))$
(case EXISTS and semantics of END)
$\Rightarrow$ … and $\forall f \in$ 'D,W$\xrightarrow{*}$W'$_{\mathcal{M}}$ not $\text{END}_{\mathcal{M}}(f(\mathfrak{I}_P(p_w), \mathcal{w}), \mathfrak{I}_P(p_w))$ (① and def. of 'D,W$\xrightarrow{*}$W'$_{\mathcal{M}}$)
$\Rightarrow$ … and $\forall f \in$ 'D,W$\xrightarrow{*}$W'$_{\mathcal{M}}$ not $\mathfrak{I}_P[f/f] \models_P$ END$(\downarrow(w \circ \downarrow(f, p_w), 0), p_w)$ (assumption)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'$(w', p_w) \wedge \forall f \in$ 'D,W$\xrightarrow{*}$W' $\neg$END$(\downarrow(w \circ \downarrow(f, p_w), 0), p_w)$ (OSPL semantics)
$\Rightarrow$ $\mathfrak{I}_P \models_P G_w$ (def. of Φ_F)

<u>Case</u> G = REFLEXIVE(p)

$\mathfrak{I}_M \models_M$ REFLEXIVE(p) (assumption)
$\Rightarrow$ $\mathcal{R}_{\mathfrak{I}_M(p)}(\mathcal{w})$ is reflexive (M-Logic semantics)
$\Rightarrow$ $\mathfrak{I}_M(p)$ exists in $\mathcal{w}$ and (otherwise $\mathcal{R}_{\mathfrak{I}_M(p)}(\mathcal{w})$ is not defined)
$\Rightarrow$ $\mathfrak{I}_P \models_P$ EXISTS'(w', p_w) and $\forall u\, \mathcal{R}^*_{\mathfrak{I}_M(p)}(\mathcal{w}, u) \Rightarrow \mathcal{R}_{\mathfrak{I}_M(p)}(u, u)$
(case EXISTS and def. of reflexivity)

➪ … and $\forall u\, \overset{*}{\mathcal{R}}_{\Im_M(p)}(w, u)$ ➪ $\exists g \in$ 'D,W→W'$_{\mathcal{M}}$: $g(\Im_P(p_w))(u) = u$
(① and def. of 'D,W→W'$_{\mathcal{M}}$)

➪ … and $\forall f \in$ 'D,W$\overset{*}{\rightarrow}$W'$_{\mathcal{M}}$ $\exists g \in$ 'D,W→W'$_{\mathcal{M}}$: (assumption and def. of 'D,W$\overset{*}{\rightarrow}$W'$_{\mathcal{M}}$)
$\Im_P[f/f, g/g] \models_P \downarrow(w \circ \downarrow(f,p_w) \circ \downarrow(g,p_w),0) = \downarrow(w \circ \downarrow(f,p_w),0)$

➪ $\Im_P \models_P$ EXISTS'(w',p_w) $\wedge$ (OSPL semantics)
$\forall$f:'D,W$\overset{*}{\rightarrow}$W' $\exists$g:'D,W→W' $\downarrow(w \circ \downarrow(f,p_w) \circ \downarrow(g,p_w),0) = \downarrow(w \circ \downarrow(f,p_w),0)$

➪ $\Im_P \models_P G_w$ (def. of Φ_F)

The proofs for the cases with the other ARP-predicates are similar.

Induction Step: The induction hypothesis is: if $\Im_P(w') = w$ and $\Im_M \models_M F$ then $\Im_P \models_P F_w$.

Cases $G = \forall x{:}S\ F$ and $G = \exists x{:}S\ F$
These cases are exactly like the corresponding cases in theorem 5.5.

Case $G = \Box_p F$
$\Im_M \models_M \Box_p F$ (assumption)

➪ $\Im_M(p)$ exists in w ➪ (M-Logic semantics)
$\forall u\, \mathcal{R}_{\Im_M(p)}(w,u)$ ➪ $\Im_M[u] \models_M F$ (M-Logic semantics)

➪ … ➪ END$_{\mathcal{M}}(w, \Im_M(p))$ or $\forall f \in$ 'D,W→W'$_{\mathcal{M}}$: $\Im_M[f(\Im_M(p), w)] \models_M F$
(semantics of END and def. of 'D,W→W'$_{\mathcal{M}}$)

➪ … ➪ END$_{\mathcal{M}}(w, \Im_P(p_w))$ or $\forall f \in$ 'D,W→W'$_{\mathcal{M}}$: $\Im_M[f(\Im_P(p_w), w)] \models_M F$ (①)

➪ … ➪ $\Im_P \models_P$ END(w',p_w) or $\forall f \in$ 'D,W→W'$_{\mathcal{M}}$: $\Im_P[f/f] \models_P F_{w \circ \downarrow(f,p_w)}$)
(ind.hyp. and assumption)

➪ $\Im_P \models_P$ EXISTS'(w',p_w) $\Rightarrow$ ($\neg$END(w',p) $\Rightarrow$ $\forall$f:'W→W' $F_{w \circ \downarrow(f,p_w)}$))
(OSPL semantics, case EXISTS)

➪ $\Im_P \models_P G_w$ (def. of Φ_F)

Case $G = \Diamond_p F$. This case is analogous to the previous one.

The cases with the normal logical connectives are trivial applications of the induction hypothesis.

To complete the soundness proof, it has to be shown that the characteristic axiom for the END-predicate and the reflexivity and transitivity axioms for the 'D,W$\overset{*}{\rightarrow}$W' sort hold. This is straightforward. ♦

Theorem 6.6 Completeness of the Translation
If the OSPL interpretation $\Im_P$ satisfies a translated specification $\Phi_S((\Sigma,G))$ then the back translated interpretation $\Im_M = \Phi_{\Im}^{-1}(\Im_P)$ satisfies the original specification (Σ,G).

The proof is analogous to the completeness proof of the relational translation. ♦

6.1 General Optimizations of the Functional Translation

On first glance the functional translation seems to generate nothing better than the relational translation. The formulae even look bigger and more complex. Instead of the R-predicate we have the END-predicate and even worse, equations occur already in the translations of propositional modal logic formulae.

It turned out, however, that the functional translation offers many more possibilities to optimize the translation itself and to improve the OSPL resolution and paramodulation calculus by incorporating some of the generated formulae, in particular the translated ARP-predicates, into theory unification [Siekmann 89] and theory resolution rules [Stickel 85]. In the subsequent paragraphs some of the obvious optimizations are discussed.

6.1.1 Removal of Superfluous Axioms

First of all we notice that the two axioms

① $\forall f,g{:}\text{`}W\xrightarrow{*}W\text{'}\ \forall w{:}W \qquad f(w) \neq g(w) \vee f = g$

② $\forall f,g{:}\text{`}D,W\xrightarrow{*}W\text{'}\ \forall p{:}D\ \forall w{:}W \qquad \downarrow(f, p)(w) \neq \downarrow(g, p)(w) \vee f = g$

generated by the specification morphism (def. 6.3) cannot be applied to any of the translated formulae because the inequality does not (theory) unify with any of the other equations. Therefore the first optimization is to drop these two axioms.

The third axiom

③ $\forall f,g{:}\text{`}W\xrightarrow{*}W\text{'}\ \forall w{:}W \qquad \downarrow(f \circ g, w) = \downarrow(g, \downarrow(f, w))$,

applied from right to left, permits the normalization of all terms of sort W as a term $\downarrow(t_1\circ\ldots\circ t_n,0)$. Since this structure is already generated by the translation itself and it is easy to keep it for instantiations caused by resolution and paramodulation, we can drop this axiom too. Furthermore, since all terms of sort W have the above structure, $\downarrow(t_1\circ\ldots\circ t_n,0)$ can be abbreviated by a string $[t_1\ldots t_n]$. (In [Ohlbach 88] this structure, called world-path, is a distinguished syntactic element).

The axioms

④ $\forall f,g{:}\text{`}D,W\xrightarrow{*}W\text{'}\ \forall p{:}D \qquad \downarrow(f \circ g, p) = \downarrow(f, p) \circ \downarrow(g, p)$

⑥ $\exists f{:}\text{`}D,W\xrightarrow{*}W\text{'}\ \forall p{:}D\ \forall w{:}W \qquad \downarrow(f,p)(w) = w$ (reflexivity)

⑦ $\forall f,g{:}\text{`}D,W\xrightarrow{*}W\text{'}\ \exists h{:}\text{`}D,W\xrightarrow{*}W\text{'} \qquad \downarrow(f,p)\circ\downarrow(g,p) = \downarrow(h,p)$ (transitivity)

describe properties of the '$D,W\xrightarrow{*}W$'-elements which have to do with reflexivity and transitivity of the $\mathcal{R}^*$ relation. Therefore they cannot be removed without loosing information. In a resolution and paramodulation calculus, they may however be replaced by a theory unification algorithm which has the following unification rule:

If f is a variable of sort '$D,W\xrightarrow{*}W$' then a unifier for the terms

$\downarrow(f,p)$ and $\downarrow(t_1,p_1)\circ\ldots\circ\downarrow(t_n,p_n)$ is $\tau = \sigma \circ \{f \mapsto t_1\circ\ldots\circ t_n\}$ where $\sigma p = \sigma p_1 = \ldots = \sigma p_n$.

Axiom ④ has to be used as rewrite rule from left to right to normalize instantiated terms such that in the above case $\tau\ (t(f,p)) = \downarrow(t_1\circ\ldots\circ t_n,\sigma p) = \downarrow(t_1,\sigma p)\circ\ldots\circ\downarrow(t_n,\sigma p)$.

This unification rule is sufficient because terms of type '$D,W\xrightarrow{*}W$' occur only in the translation of the ARP-predicates and here they actually stand for nothing else than arbitrary strings $\downarrow(t_1,p)\circ\ldots\circ\downarrow(t_n,p)$.

The only remaining additional axiom is now

⑤ $\forall f{:}\text{`}W\xrightarrow{*}W\text{'}\ g{:}\text{`}D,W\xrightarrow{*}W\text{'}\ \forall p{:}D \qquad \neg\text{END}(\downarrow(f\circ\downarrow(g,p),0), p) \vee \neg\text{END}(\downarrow(f,0),p)$

This axiom can only be removed in the serial case (see below). In the general case, however, it can be used to justify a reduction rule which is able to reduce the number of generated END-literals stemming from the modal operators: If a clause contains the literals

$$\text{END}(\downarrow(t_1\circ\ldots\circ t_n,0), p) \vee \neg\text{END}(\downarrow(t_1\circ\ldots\circ t_n\circ\downarrow(s_1,p)\circ\ldots\circ\downarrow(s_k,p),0), p) \vee D$$

then resolution of the first literal with the second literal in ⑤ yields

$$\neg\text{END}(\downarrow(t_1\circ\ldots\circ t_n\circ\downarrow(g,p),0), p) \vee \neg\text{END}(\downarrow(t_1\circ\ldots\circ t_n\circ\downarrow(s_1,p)\circ\ldots\circ\downarrow(s_k,p),0), p) \vee D$$

where g is a new variable. Since g occurs only once, the factor generated by $\tau = \{g \mapsto s_1\circ\ldots\circ s_k\}$ subsumes the original clause. The net effect is the removal of the literal '$\text{END}(\downarrow(t_1\circ\ldots\circ t_n,0),p)$', and this can be done immediately after the generation of the clause normal form.

6.1.2 Prefix Stability

Functionally translated formulae have a particular syntactic property which can be exploited to simplify certain things in the resolution calculus. This property, called prefix stability, says that the prefix of every variable of sort 'D,W→W' is unique. The prefix of a variable 'f' is the chain of terms '$t_1 \circ \ldots \circ t_n$' standing before 'f' in terms '$\downarrow(t_1 \circ \ldots \circ t_n \circ \downarrow(f,p) \ldots, 0)$'. Since these terms originate from the modal operators, the prefix of a variable 'f' comes from the embracing modal operators of the operator, 'f' comes from. For example '$\Diamond_p \Box_p (Q \vee \Diamond_q R)$' with rigid p and q is translated into

$\text{EXISTS}(0,p) \wedge \neg\text{END}(0,p) \wedge \exists f\ (\text{EXISTS}(\downarrow(\downarrow(f,p),0),p) \Rightarrow \neg\text{END}(\downarrow(\downarrow(f,p),0),p) \Rightarrow$
$\forall g\ (Q(\downarrow(\downarrow(f,p) \circ \downarrow(g,p),0)) \vee (\text{EXISTS}(\downarrow(\downarrow(f,p) \circ \downarrow(g,p),0),p) \wedge \neg\text{END}(\downarrow(\downarrow(f,p) \circ \downarrow(g,p),0),q) \wedge$
$\exists h\ R(\downarrow(\downarrow(f,p) \circ \downarrow(g,p) \circ \downarrow(h,q),0)$.

The prefix of 'f', which comes from '$\Diamond_p$', is empty, the prefix of 'g', which comes from '$\Box_p$', is '$\downarrow(f,p)$' and the prefix of h is '$\downarrow(f,p) \circ \downarrow(g,p)$'.

One place where this nice property can be exploited is the application of the reflexivity axiom. The formula 'REFLEXIVE(p)' where p is again rigid, is translated and Skolemized into '$\text{EXISTS}(p,0) \wedge \forall f\ \downarrow(\downarrow(f,p) \circ \downarrow(g(f),p),0) = \downarrow(\downarrow(f,p),0)$'. Actually '$\downarrow(g(f),p)$' is something like the identity function, however depending on f. Nevertheless we want to use the equation '$\downarrow(\downarrow(f,p) \circ \downarrow(g(f),p),0) = \downarrow(\downarrow(f,p),0)$' to remove a term '$\downarrow(x,p)$' inside a term '$\ldots \circ \downarrow(x,p) \circ \ldots$' where 'x' is a variable, by paramodulation without introducing 'g(f)' by instantiating 'x' somewhere else. If the prefix of 'x' is the same everywhere, for example '$s \circ \downarrow(x,p)$', then paramodulation with the reflexivity axiom at one particular occurrence of 'x' yields the instantiated terms '$s \circ \downarrow(g(s),p)$' everywhere else. Applying the reflexivity axiom as rewrite rule from left to right allows to simplify '$s \circ \downarrow(g(s),p)$' to 's', thus eliminating all occurrences of 'g(...)'. As an example consider the formula '$Q(\downarrow(\downarrow(f,p) \circ \downarrow(x,p),0) \vee R(\downarrow(\downarrow(f,p) \circ \downarrow(x,p),0)$'. Paramodulation with the reflexivity axiom yields '$Q(\downarrow(\downarrow(f,p),0) \vee R(\downarrow(\downarrow(f,p) \circ \downarrow(g(f),p),0)$'. Applying the reflexivity axiom as rewrite rule now yields '$Q(\downarrow(\downarrow(f,p),0) \vee R(\downarrow(\downarrow(f,p),0)$'. The net effect is the removal of '$\downarrow(x,p)$', and this is what is expected in the reflexive case. The same operations are possible in the symmetric, euclidean and linear case. That means prefix stability can be exploited to avoid the introduction of the Skolem functions from the special ARP-formulae into deduced formulae.

In [Ohlbach 88], prefix stability is exploited to develop a finitary unification algorithm which replaces the transitivity axiom for the accessibility relation. Without prefix stability the unification algorithm would be infinitary.

Definition 6.7 Prefix and Prefix Stability
If a term 't' of sort '$D,W \xrightarrow{*} W$' occurs as subterm in a term '$\downarrow(t_1 \circ \ldots \circ t_n \circ \downarrow(t,p) \ldots, 0)$' then '$t_1 \circ \ldots \circ t_n$' is called a **prefix** of 't' and '$t_1 \circ \ldots \circ t_n \circ \downarrow(t,p)$' is called prefix* of 't'. For a given set F of formulae 'prefix(t,F)' is the set of all prefixes of 't' in F and 'prefix*(t,F)' is the set of all prefix*s of 't' in F. A set F of formulae is called **prefix stable** if prefix(f,F) is a singleton for all '$D,W \xrightarrow{*} W$'-variables in f. ♦

There is a useful property of prefix stable terms which can be exploited at various places: In prefix stable formulae variables do not occur in their own prefix.

Lemma 6.8 If the prefix of a variable f is unique then f does not occur in its own prefix.
Proof: Assume the prefix of f is $\downarrow(t_1,p_1) \circ \ldots \circ \downarrow(t_k,p_k)$ and f occurs in some t_i or p_i. Since f's prefix is unique, $\downarrow(t_1,p_1) \circ \ldots \circ \downarrow(t_k,p_k)$ must occur in t_i or p_i respectively, which is impossible for finite terms. ♦

An immediate consequence of the previous lemma is a kind of top level linearity for prefix stable terms: '$D,W\xrightarrow{*}W'$'-variables can occur at most once on top level of the terms $\downarrow(t_1,p_1)\circ\ldots\circ\downarrow(t_k,p_k)$. That means for example a term like $\downarrow(\downarrow(f,p)\circ\downarrow(f,q))$ never occurs in translated M-Logic formulae.

Proposition 6.9 Φ_F produces prefix stable formulae.
The formula morphism Φ_F produces prefix stable formulae if all quantified variables are different.

Proof: We prove by induction on the structure of terms and formulae: $F_w =_{def} \Phi_F(F,w)$ is prefix stable and for all variables f occurring in w, prefix(f, F_w) = prefix(f,w).
Almost all cases are straightforward. We therefore show only one typical case in the induction step:

Case $F = F_1 \wedge F_2$:
$F_w = F_{1w} \wedge F_{2w}$ (def. of Φ_F)
Let f be a '$D,W\rightarrow W'$'-variable.
Case 1: f occurs in w:
⇨ prefix(f,F_{1w}) = prefix(f,w) = prefix(f,F_{2w}) (induction hypothesis)
⇨ prefix(f,F_w) = prefix(f,$F_{1w} \wedge F_{2w}$) = prefix(f,w) (def. of Φ_F)
and prefix(f,F_w) is a singleton (induction hypothesis).
Case 2: f does not occur in w:
⇨ Since all quantified variables are different, f occurs either in F_{1w} or in F_{2w} but not in both; w.l.o.g assume f occurs in F_{1w}.
⇨ prefix(f,F_w) = prefix(f,$F_{1w} \wedge F_{2w}$) (def. of Φ_F)
= prefix(f,F_{1w}) $\cup$ { }
⇨ prefix(f,F_w) is a singleton (induction hypothesis). ♦

Prefix stability can be exploited only if it is preserved by all operations on the translated formulae, i.e. by Skolemization, generation of clauses, resolution and paramodulation. For all these operations, either prefix stability invariance has to be shown or the operation has to modified to preserve prefix stability.

Skolemization

Standard Skolemization does not preserve prefix stability as the following example shows: The Skolemization of '$\forall f\,\forall g\,\exists h\; P(\downarrow(\downarrow(f,p)\circ\downarrow(g,p)\circ\downarrow(h,p),0)$' yields '$\forall f\,\forall g\; P(\downarrow(\downarrow(f,p)\circ\downarrow(g,p)\circ\downarrow(h(f,g),p),0)$' where 'g' occurs once in '$\downarrow(g,p)$' with prefix '$\downarrow(f,p)$' and once in h(f,g) where the definition of a prefix makes not much sense. To restore prefix stability, we introduce an extended Skolemization (c.f. [Herzig 89]) which in the above case yields '$\forall f\,\forall g\; P(\downarrow(\downarrow(f,p)\circ\downarrow(g,p)\circ\downarrow(h(\downarrow(f,p),\downarrow(f,p)\circ\downarrow(g,p)),p),0)$'. Although this looks more complicated, it helps simplify the proof search.

Lemma 6.10 Extended Skolemization
A formula $\forall x_1\ldots x_n\,\exists y\; F[t_1(x_1),\ldots,t_n(x_n),y]$ can be Skolemized to $\forall x_1\ldots x_n\; F[t_1(x_1),\ldots t_n(x_n), g(t_1(x_1),\ldots,t_n(x_n)]$ where $F[t_1(x_1),\ldots,t_n(x_n),y]$ means that all occurrences of x_i are within a term $t_i(x_i)$.
Proof:
$\forall x_1\ldots x_n\,\exists y\; F[t_1(x_1),\ldots,t_n(x_n),y]$
$\Leftrightarrow \forall x_1'\ldots x_n'\,\exists y\,\forall x_1\ldots x_n\; x_1' = t_1(x_1) \wedge\ldots\wedge x_n' = t_n(x_n) \Rightarrow F[x_1',\ldots,x_n',y]$

is unsatisfiable iff

$\forall x_1'\ldots x_n' \forall x_1\ldots x_n\; x_1' = t_1(x_1) \wedge\ldots\wedge x_n' = t_n(x_n) \Rightarrow F[x_1',\ldots,x_n',g(x_1'\ldots x_n')]$

$\Leftrightarrow \forall x_1\ldots x_n\; F[x_1',\ldots,x_n',g(t_1(x_1),\ldots,t_n(x_n)))]$ ♦

This lemma, which holds for arbitrary predicate logic formulae, can be used to Skolemize translated M-Logic formulae by first generating a prefix form, which is a standard transformation in predicate logic and then Skolemizing as follows:

$\forall f_1\ldots f_n$:'D,W→W' $\forall x_1\ldots x_m \exists y\; F[t_1(x_1),\ldots,t_n(x_n),y]$

$\rightarrow \forall f_1\ldots f_n$:'D,W→W' $\forall x_1\ldots x_n\; F[t_1(x_1),\ldots,t_n(x_n),f(s_1,\ldots,s_n, x_1\ldots x_m)]$

where $s_i = \text{prefix}^*(f_i,F)$.

Since the translated formulae are prefix stable (proposition 6.9), lemma 6.10 is applicable and the extended Skolemization yields again prefix stable formulae.

Generation of Clauses

Clause generation consists of manipulations on the formula level followed by a variable renaming to make the clauses variable disjoint. Since the manipulations on the formula level do not modify the structure of terms, prefix stability is guaranteed. The variable renaming operation replaces all variables consistently by new variables. It is easy to see that this operation also preserves prefix stability.

Instantiation of Clauses and Resolution

Both deduction rules, resolution and paramodulation consist of an instantiation operation followed by the corresponding clause forming operations. As the following example shows, instantiation of clauses with arbitrary substitutions does not preserve prefix stability: $\{x \mapsto a(\downarrow(\downarrow(f,p)\circ\downarrow(g,p),0))\}$ $(P(x) \vee Q(\downarrow(\downarrow(h,p)\circ\downarrow(g,p),0))) = P(a(\downarrow(\downarrow(f,p)\circ\downarrow(g,p),0)) \vee Q(\downarrow(\downarrow(h,p)\circ\downarrow(g,p),0))$ which is no longer prefix stable because there are two different prefixes for 'g'. The hope is therefore that at least the instantiation with most general unifiers preserves the prefix stability. The next lemma gives a sufficient condition for substitutions such that instantiation preserves prefix stability.

Lemma 6.11

If C is a prefix stable clause set and $\sigma = \{x \mapsto s\}$ is an idempotent substitution where either s is a new variable or s occurs in C or $s = s_1\circ\ldots\circ s_n$ and $\downarrow(s_1,p)\circ\ldots\circ\downarrow(s_n,p)$ occur in C and prefix(s_1,C) = prefix(x,C) then σC is prefix stable.

Proof: A consequence of lemma 6.8 is that for the $s = s_1\circ\ldots\circ s_n$ case the variable x is not contained in prefix(s_1,C) = prefix(x,C) ①

Let f be a 'D,W$\xrightarrow{*}$W'-variable in σC.

Case 1: f does not occur in s.

In this case f occurs in C. Let q be f's (unique) prefix in C. Since f does not occur in s, f also not occurs in σq which is the prefix of $\sigma f = f$. Hence f's prefix in σC is again unique.

Case 2: f occurs in s.

If f occurs at a deeper nesting level in s, its prefix in C must be the same as its prefix in σC because s occurs in C. Therefore let $s = s_1\circ\ldots\circ s_n$ and $f = s_i$ for some i. (The case that s is a new variable is trivial). The situation is as follows:

Occurrences of $f = s_i$ in C: prefix(s_1,C)$\circ\downarrow(s_2,p)\circ\ldots\circ\downarrow(s_i,p)$.

Since prefix(s_1,C) = prefix(x,C) and because of ①, x does not occur in prefix(s_1,C). Furthermore because σ is idempotent, x does not occur in $\downarrow(s_2,p)\circ\ldots\circ\downarrow(s_i,p)$. Therefore for these occurrences in

C we have that the prefix in σC is again prefix(s_1,C)$\circ\downarrow(s_2,p)\circ\ldots\circ\downarrow(s_i,p)$.
Now we have to consider the occurrence of x in C which is prefix(x,C)$\circ\downarrow$(x,p) = prefix(s_1,C)$\circ\downarrow$(x,p). Instantiated with σ and normalized, prefix(s_1,C)$\circ\downarrow(s_2,p)\circ\ldots\circ\downarrow(s_i,p)\circ\ldots$ is obtained, and this is the same term as above. Therefore we conclude in this case again that prefix(f,σC) is unique. ♦

Definition 6.12 Standard Unification
A unification algorithm generating unifiers σ for some terms t in the following way is called a **standard unification algorithm:** σ can be written as a composition $\sigma_1\circ\ldots\circ\sigma_n$ where for i = 1,...,n, $\sigma_i = \{x \mapsto s\}$ is an idempotent substitution such that

- either s is a new variable or
- s occurs in the partially unified terms $t_{i-1} =_{def} (\sigma_1\circ\ldots\circ\sigma_{i-1})t$ or
- or $s = s_1\circ\ldots\circ s_n$ and $\downarrow(s_1,p)\circ\ldots\circ\downarrow(s_n,p)$ occur in t_{i-1} and prefix(s_1,t_{i-1}) = prefix(x,t_{i-1}). ♦

It is noted that the normal unification algorithm for OSPL [Schmidt-Schauß 89], enhanced by the unification rule for 'D,W$\overset{*}{\rightarrow}$W'-terms is of this type, even with arbitrary term declarations. In the sequel we consider only standard unification.

Lemma 6.13 Instantiation of a clause set C with a unifier for some of its terms which is generated by a standard unification algorithm preserves prefix stability.
Proof: We exploit that the unification algorithm generates substitutions σ which can be written as a composition $\sigma_1\circ\ldots\circ\sigma_n$ with properties as listed in lemma 6.11. Induction on n and application of lemma 6.11 yields the desired result. ♦

Lemma 6.14 Resolution with a unifier computed by a standard unification algorithm preserves prefix stability.
The proof is an immediate consequence of the previous lemma. ♦

Paramodulation

It is quite natural that resolution preserves prefix stability because instantiation of whole clauses with most general unifiers does not seriously change the structure of terms. This is different for the paramodulation operation which replaces a single subterm by another term. In particular, a paramodulation may modify one occurrence of a prefix of a 'D,W$\overset{*}{\rightarrow}$W'-variable and leave another occurrence unchanged, thus destroying prefix stability. For example consider the prefix stable clause 'P($\downarrow(\downarrow$(f,p(a))$\circ\downarrow$(g,p(a)),0) $\vee$ Q($\downarrow(\downarrow$(f,p(a))$\circ\downarrow$(g,p(a)),0)'. Paramodulation with 'a = c' yields 'P($\downarrow(\downarrow$(f,p(c))$\circ\downarrow$(g,p(a)), 0) $\vee$ Q($\downarrow(\downarrow$(f,p(a))$\circ\downarrow$(g,p(a)), 0)' which is no longer prefix stable. The idea to restore prefix stability is to change the paramodulation strategy such that all occurrences of a particular term in the prefix of a world variable of a clause are paramodulated simultaneously. In the above example, the second occurrence of 'a' would also be paramodulation such that the twice paramodulated clause is again prefix stable.

A general example for simultaneous paramodulation:

$$\frac{\begin{array}{l}P(k(x)) \vee Q(k(a),x) \\ k(a) = c \vee D \qquad\qquad \sigma = \{x \mapsto a\}, \text{paramodulation into } k(x)\end{array}}{P(c) \vee Q(c,a) \vee D}$$ ♦

Dan Benanav has shown that this kind of simultaneous paramodulation, together with resolution and factoring is complete for unsorted predicate logic [Benanav 90], i.e. every unsatisfiable clause set can be refuted with simultaneous paramodulation, resolution and factoring. For sorted logic, simultaneous paramodulation is in general not complete as the following counter example shows: Suppose we have the two sorts A and B, the subsort declaration $A \sqsubseteq B$, a constant symbol a:B, a predicate symbol P: B×A and a function symbol k: B→A. The three clauses {P(a,k(a))}, {¬P(k(a),k(a))} and {a = k(a)} can be refuted by normal paramodulation and resolution but not by simultaneous paramodulation. The reason is that P(a,a) is not a well sorted atom and therefore these paramodulations are not possible. The possible simultaneous paramodulation into the first clause yields P(k(a),k(k(a)) which does not help.

For preserving prefix stability, it is however not necessary to apply simultaneous paramodulation only. Only paramodulations into different occurrences of a world variable's prefix should be paramodulated simultaneously. All other cases are treated as before. Since the sort of a world variable's prefix is always '$W \rightarrow W$' or '$W \xrightarrow{*} W$' and never changes through paramodulation, this guarantees that whenever a paramodulation into one occurrence of a world variable's prefix is possible, simultaneous paramodulation into all occurrences is possible too. Benanav's completeness proof now carries over to this restricted version of simultaneous paramodulation.

Lemma 6.15 Restricted simultaneous paramodulation preserves prefix stability.
Proof: Let C[s'] and s = t, D be the variable disjoint, prefix stable parent clauses of a paramodulation yielding (σC)[σs' → σt], σD as simultaneous paramodulant. According to lemma 6.13, σ(C, s = t, D) is prefix stable. Let 'f' be a world variable in the paramodulant. We consider the critical case where σs occurs in f's prefix in σ(C, s = t, D). If f occurs in σC, all occurrences of σs in σC are replaced by σt. Therefore f's prefix in (σC)[σs' → σt] is still unique.. f must also occur in C with s' in its prefix, because otherwise there would be another variable g which is instantiated with f and s would be in the prefix of g. Since g cannot occur in its own prefix (lemma 6.8), and since the parent clauses are variable disjoint, g can not be instantiated by the unifier for s' and s. Therefore f occurs already in C. Since the clauses are variable disjoint and since f is not instantiated by σ, f cannot occur in σD. Thus, f's prefix is unique in the whole paramodulant. ♦

6.2 Specific Optimizations for the Functional Translation

For particular cases, the translation rules can further be simplified. We consider some of the more frequently occurring cases.

6.2.1 Unparametrized Modal Operators

Since modal operators without parameters are an important subcase, we list the simplified translation rules for this case:

- SERIAL_W $= \forall f{:}\text{‘}W \xrightarrow{*} W\text{’}\ \neg\text{END}(\downarrow(w \circ f),0))$
- REFLEXIVE_W $= \forall f{:}\text{‘}W \xrightarrow{*} W\text{’} \exists g{:}\text{‘}W \to W\text{’}\ \downarrow(w \circ f \circ g,0) = \downarrow(w \circ f,0)$
- TRANSITIVE_W $= \forall f,g{:}\text{‘}W \to W\text{’} \exists h{:}\text{‘}W \to W\text{’}\ \downarrow(w \circ f \circ g,0) = \downarrow(w \circ h,0)$
- SYMMETRIC_W $= \forall f{:}\text{‘}W \xrightarrow{*} W\text{’}\ \forall g{:}\text{‘}W \to W\text{’}\ \exists h{:}\text{‘}W \to W\text{’}\ \downarrow(w \circ f \circ g \circ h,0) = \downarrow(w \circ f,0)$
- EUCLIDEAN_W $= \forall f{:}\text{‘}W \xrightarrow{*} W\text{’}\ \forall g,h{:}\text{‘}W \to W\text{’}\ \exists k{:}\text{‘}W \to W\text{’}\ \downarrow(w \circ f \circ g \circ k,0) = \downarrow(w \circ f \circ h,0)$
- LINEAR_W $= \forall f,g{:}\text{‘}W \xrightarrow{*} W\text{’}\ \exists h{:}\text{‘}W \xrightarrow{*} W\text{’}$
  $\downarrow(w \circ f \circ h,0) = \downarrow(w \circ g,0) \vee \downarrow(w \circ g \circ h,0) = \downarrow(w \circ f,0)$
- $\text{INCREASING-DOMAIN}_W$ $= \forall f{:}\text{‘}W \xrightarrow{*} W\text{’}\ \forall g{:}\text{‘}W \to W\text{’}\ \forall x{:}D$
  $\text{EXISTS'}(\downarrow(w \circ f,0),x) \Rightarrow \text{EXISTS'}(\downarrow(w \circ f \circ g,0),x)$
- $\text{DECREASING-DOMAIN}_W$ $= \forall f{:}\text{‘}W \xrightarrow{*} W\text{’}\ \forall g{:}\text{‘}W \to W\text{’}\ \forall x{:}D$
  $\text{EXISTS'}(\downarrow(w \circ f,0),x) \Leftarrow \text{EXISTS'}(\downarrow(w \circ f \circ g,0),x)$
- $(\Box F)_W$ $= \neg\text{END}(\downarrow(w,0)) \Rightarrow \forall f{:}\text{‘}W \to W\text{’}\ F_{w \circ f}$
- $(\Diamond F)_W$ $= \neg\text{END}(\downarrow(w,0)) \wedge \exists f{:}\text{‘}W \to W\text{’}\ F_{w \circ f}$
- All other cases are as before.

As mentioned before the only axiom to be generated by the specification morphism is ⑤. The simplified version for the unparametrized case is:

⑤' $\forall f{:}\text{‘}W \xrightarrow{*} W\text{’}\ g{:}\text{‘}W \xrightarrow{*} W\text{’}\qquad \text{END}(\downarrow(f \circ g,0)) \Rightarrow \neg\text{END}(\downarrow(f,0))$

6.2.2 Serial Accessibility Relation

The formula ‘$\forall p\ \text{SERIAL}(p)$’ is translated into ‘$\forall p\ \forall f{:}\text{‘}D,W \xrightarrow{*} W\text{’}\ \neg\text{END}(\downarrow(\downarrow(f,p),0),p)$’. This clause subsumes all formulae containing a literal ‘$\neg\text{END}(\ldots)$’ conjunctively and allows to “resolve away” literals ‘$\text{END}(\ldots)$’ contained disjunctively in a formula. Such literals occur in the translation of the two modal operators. The combined effect of generating these literals during the translation and subsuming or resolving them away with the seriality clause can be achieved by simply not generating them. The optimized translation rules for this case are therefore:

- If a specification contains a formula ‘$\forall p\ \text{SERIAL}(p)$’ then remove axiom ⑤ and use the following optimized translation rules:
  - $(\Box_p F)_W$ $= \text{EXISTS'}(w',p_W) \Rightarrow \forall f{:}\text{‘}W \to W\text{’}\ F_{w \circ \downarrow(f,p_w)}$
  - $(\Diamond_p\, F)_W$ $= \text{EXISTS'}(w',p_W) \wedge \exists f{:}\text{‘}W \to W\text{’}\ F_{w \circ \downarrow(f,p_w)}$.

6.2.3 Constant Domain Interpretations

In case '$\forall x \forall p$ PERSISTENT(x,p)' holds, i.e the domain does not change, the following clause is generated by Φ_F: '$\forall p,x{:}D\ \forall f{:}$'D,W$\xrightarrow{*}$W' EXISTS'($\downarrow(\downarrow(f,p),0),x)$'. This formula subsumes or resolves away respectively all EXISTS' literals generated by the translation. Therefore we can formulate optimized translation rules for the constant domain case too:

- If a specification contains the formula '$\forall x \forall p$ PERSISTENT(x,p)' then use the following optimized translation rules:
  - $P(t_1,\ldots,t_n)_w = P'(w',t_{1w},\ldots,t_{nw})$ if P is a flexible predicate symbol.
  - $\text{SERIAL}(p)_w = \forall f{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \neg\text{END}(\downarrow(w\circ\downarrow(f,p_w),0),p_w)$
  - $\text{REFLEXIVE}(p)_w = \forall f{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \exists g{:}\text{'D,W}\rightarrow\text{W'}\ \downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w),0) = \downarrow(w\circ\downarrow(f,p_w),0)$
  - $\text{TRANSITIVE}(p)_w = \forall f,g{:}\text{'D,W}\rightarrow\text{W'}\exists h{:}\text{'D,W}\rightarrow\text{W'}\ \downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w),0) = \downarrow(w\circ\downarrow(h,p_w),0)$
  - $\text{SYMMETRIC}(p)_w = \forall f{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall g{:}\text{'D,W}\rightarrow\text{W'}\ \exists h{:}\text{'D,W}\rightarrow\text{W'}$
    $\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w)\circ\downarrow(h,p_w),0) = \downarrow(w\circ\downarrow(f,p_w),0)$
  - $\text{EUCLIDEAN}(p)_w = \forall f{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall g,h{:}\text{'D,W}\rightarrow\text{W'}\ \exists k{:}\text{'D,W}\rightarrow\text{W'}$
    $\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(g,p_w)\circ\downarrow(k,p_w),0) = \downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(h,p_w),0)$
  - $\text{LINEAR}(p)_w = \forall f,g{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \exists h{:}\text{'D,W}\xrightarrow{*}\text{W'}$
    $\downarrow(w\circ\downarrow(f,p_w)\circ\downarrow(h,p_w),0) = \downarrow(w\circ\downarrow(g,p_w),0)\ \vee$
    $\downarrow(w\circ\downarrow(g,p_w)\circ\downarrow(h,p_w),0) = \downarrow(w\circ\downarrow(f,p_w),0)$
  - $(\forall x\ F)_w = \forall x\ F_w$
  - $(\exists x\ F)_w = \exists x\ F_w$
  - $(\Box_p F)_w = \neg\text{END}(w',p_w) \Rightarrow \forall f{:}\text{'D,W}\rightarrow\text{W'}\ F_{w\circ\downarrow(f,p_w)}$
  - $(\Diamond_p F)_w = \neg\text{END}(w',p_w) \wedge \exists f{:}\text{'D,W}\rightarrow\text{W'}\ F_{w\circ\downarrow(f,p_w)}$.

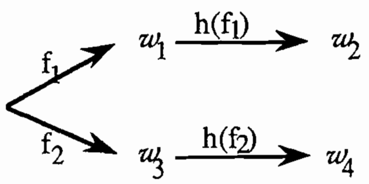
That means no EXISTS'-literal at all occur in the translated formulae.

6.3 Further Optimizations

The optimizations presented in this section are still subject to ongoing investigations.

6.3.1 Optimized Skolemization of Modal Variables

In the serial case the formula $\Box\Diamond Q$ is translated, into $\forall f\ \exists g\ Q(\downarrow(f\circ g,0)$ which, Skolemized in the usual way, yields $\forall f\ Q(\downarrow(f\circ h(f),0))$. The Skolem function h depends on f. Since the term $\downarrow(f\circ h(f),0)$ is nothing else than $h(f)(f(0)) = h(f,f(0))$ in second order syntax, this term depends twice on f, which seems to be redundant. A graphical picture of the situation supports this conjecture.



h needs not depend on f and may nevertheless map $w1$ to $w2$ and $w3$ to $w4$. That means an optimized Skolemization could generate $\forall f\ Q(\downarrow(f\circ h,0))$ with a constant symbol h.

Unfortunately this view is too simplistic as the following counter example shows, which was discovered by Patrice Enjalbert: Assuming constant domains, the translated and optimized

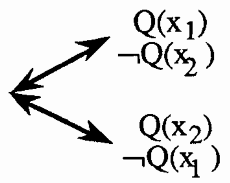
Skolemized version of the modal formula $\Box(\exists x(Q(x) \wedge \Box\Diamond\neg Q(x))$ is
$$\forall f\ Q(\downarrow(f,0),a(\downarrow(f,0)) \wedge \forall f,g\ \neg Q(\downarrow(f\circ g\circ h,0),a(\downarrow(f,0)).$$

If the accessibility relation is symmetric, i.e. the equation

$\forall f{:}\text{'D,W}\xrightarrow{*}\text{W'}\ \forall g{:}\text{'D,W}\rightarrow\text{W'}\ \exists h{:}\text{'D,W}\rightarrow\text{W'}\ \downarrow(f\circ g\circ h,0) = \downarrow(f,0)$

or, not 100% correct, but working in practice: $\forall g{:}\text{'D,W}\rightarrow\text{W'}\ g\circ g^{-1} = id$

holds, this formula is refutable. (A corresponding unification algorithm with this axiom built in, generates a unifier $\sigma = \{f \mapsto h,\ g \mapsto h^{-1}\}$ for $Q(\downarrow(f,0),a(\downarrow(f,0))$ and $Q(\downarrow(f\circ g\circ h,0),a(\downarrow(f,0)))$. The modal logic version, however, is satisfiable. A model is:



That means at least in the symmetric case the normal or extended Skolemization is necessary. Andreas Herzig has shown that the optimized Skolemization is sound at least for the propositional modal logic case [Herzig 89]. There is a strong conjecture that it also works in the first order case when the properties of the accessibility relation do not permit movements arbitrarily forward and backward in the possible worlds structure. For certain properties this conjecture has been proved in [Auffray 89].

6.3.2 Theory Unification

The translation of the ARP predicates yields equations in most cases. To tune the resolution and paramodulation calculus, it is necessary to turn these equations into theory unification algorithms. I have done this for the case of classical modal logics with unparametrized operators and for logics with reflexive, symmetric and transitive accessibility relation. The algorithm is presented here without soundness and completeness proofs. They can be found in [Ohlbach 88]. The algorithm is not yet extended to the general case.

The process of unification is considered 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 [Herbrand 30], [Martelli & Montanari 82] and others. We 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) 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 transformation rules are:
(The letters written outlined denote - possibly empty - strings $s_1\circ\ldots\circ s_k$ of world terms.)

General rules:

$f(s_1,\ldots,s_n) = f(t_1,\ldots t_n)$	$\to s_1 = t_1 \,\&\ldots\&\, s_n = t_n$	(Decomposition)
$s \circ \mathbb{s} = t \circ \mathbb{t}$	$\to s = t \,\&\, \mathbb{s} = \mathbb{t}$	(Separation).

Rule for reflexive accessibility relation: (f is a world variable)

$\mathbb{s} \circ f \circ \mathbb{s}' = t$	$\to f = id \,\&\, \mathbb{s} \circ \mathbb{s}' = t$	(Identity).

Rule for symmetric accessibility relation:

$\mathbb{s} \circ s \circ f \circ \mathbb{s}' = t$	$\to f = s^{-1} \,\&\, \mathbb{s} \circ \mathbb{s}' = t$	(Inverse)

Rules for transitive accessibility relation:

$f \circ \mathbb{s} = \mathbb{t} \circ \mathbb{t}'$	$\to f = \mathbb{t} \;\&\; \mathbb{s} = \mathbb{t}'$	(Path-Separation)
$f \circ s \circ \mathbb{s} = \mathbb{t} \circ t \circ g \circ \mathbb{t}'$	$\to g = g_1 \circ g_2 \;\& f = \mathbb{t} \circ t \circ g_1 \;\& s \circ \mathbb{s} = g_2 \circ \mathbb{t}'$	
	if s and t exist. g_1 and g_2 are new world variables.	(Splitting) ♦

Examples: Unification of the terms $a\circ f\circ g$ and h, where f,g,h are variables, assuming symmetry and reflexivity of the accessibility relation yields the unifiers $\{f \mapsto id, g \mapsto id, h \mapsto a\}$, $\{f \mapsto a^{-1}, g \mapsto h\}$ and $\{g \mapsto f^{-1}, h \mapsto a\}$. Unification of the terms $f\circ b\circ c\circ g$ and $a\circ h\circ c$, where f,g,h are again variables, assuming transitivity of the accessibility relation yields the unifiers $\{f \mapsto a, h \mapsto b\circ c, g \mapsto c\}$ and $\{f \mapsto a, g \mapsto g_1\circ c, h \mapsto b\circ c\circ g_1\}$.

6.4 A Final Example

There is a famous example from McCarthy, the *wise men puzzle*, that has been used to test the representation ability of formalisms for knowledge and belief. As a last example we give an axiomatization of the wise men puzzle in M-Logic and a proof by functional translation and resolution refutation. Its 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 color 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 men 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 ≠-predicate.

At least one of them has a white spot and everybody knows that everybody else knows that his colleagues know this. (ws(S) means S has a white spot.)

C4: $\forall S, S', S''\text{: } \Box_S \Box_{S'} \Box_{S''}\ ws(A) \vee ws(B) \vee ws(C)$

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: $\forall S,S'\text{: } S \neq S' \Rightarrow \Box_S\ (\neg ws(S) \Rightarrow \Box_{S'}\neg ws(S))$
C6: $\forall S,S',S''\ S \neq S' \wedge S \neq S'' \wedge S' \neq S''\text{: } \Rightarrow \Box_S \Box_{S'}\ (\neg ws(S) \Rightarrow \Box_{S''}\neg ws(S))$
C7: $\forall S,S',S''\ S \neq S' \wedge S \neq S'' \wedge S' \neq S''\text{: } \Rightarrow \Box_S \Box_{S'}\ (\neg ws(S') \Rightarrow \Box_{S''}\neg ws(S'))$

(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: $\Box_C \neg \Box_B\ ws(B) \qquad (\Leftrightarrow \Box_C \Diamond_B \neg ws(B))$

C knows that B knows that A does not know the colour of his spot.

C9: $\Box_C \Box_B \neg \Box_A\ ws(A) \quad (\Leftrightarrow \Box_C \Box_B \Diamond_A \neg ws(A))$.

We translate the formulae into OSPL syntax assuming seriality of the accessibility relation:

The sort of the variables in lowercase symbols is 'W→W'. To make the formula more readable we use second order syntax and drop the ∘ and ↓-function and the 0 sign writing terms ↓(x∘...∘z, 0) in simple brackets [x...z].

C1: $A \neq B$ C2: $A \neq C$ C3: $B \neq C$
C4: $\forall S,u,S',u',S'',u''$:
$ws([u(S)u'(S')u''(S'')],A) \vee ws([u(S)u'(S')u''(S'')],B) \vee ws([u(S)u'(S')u''(S'')],C)$
C5: $\forall S,u, S',u'\text{: } S = S' \vee ws([u(S)], S) \vee \neg ws([u(S)u'(S')],S)$
C6: $\forall S,u, S',u', S'',u''$:
$S = S' \vee S = S'' \vee S' = S'' \vee ws([u(S)u'(S')], S) \vee \neg ws([u(S)u'(S')u''(S'')], S))$
C7: $\forall S,u, S',u', S'',u''$:
$S = S' \vee S = S'' \vee S' = S'' \vee ws([u(S)u'(S')],S') \vee \neg ws([u(S)u'(S')u''(S'')],S'))$
C8: $\forall u\ \neg ws([u(C)g(u)(B)],B)$
C9: $\forall u,v\ \neg ws([u(C)v(B)\ h(u,v)(A)],A)$ (g and h are Skolem functions)

A deduction of the fact that C knows the colour of his own spot, i.e. $\Box_C ws(C)$ is now a trivial exercise for any resolution theorem prover. The following UR-proof was found by our system [Ohlbach &Siekmann 89]:

C1,C2,C3,C7,C8 → R1: $\forall u,u''\ \neg ws([u(C)\ g(u)(B)\ u''(A)], B)$ $(\Leftrightarrow \Box_C \Diamond_B \Box_A \neg ws(B))$
C9, R1,C4 → R2: $\forall u\ ws([u(C)\ g(u)(B)\ h(u,g(u))(A)], C)$ $(\Leftrightarrow \Box_C \Diamond_B \Diamond_A ws(C))$

C1,C2,C3,R2,C6 $\rightarrow$ R3: $\forall u$ ws([u(C) g(u)(B)], C) $(\Leftrightarrow \Box_C \Diamond_B \text{ws}(C))$
C3,R3,C5 $\rightarrow$ R4: $\forall u$ ws([u(C)], C) $(\Leftrightarrow \Box_C \text{ws}(C))$ ♦

7 RELATED WORK

There are well known but very specific translation methods for logics. For example Skolemization translates formulae from general predicate logic into the fragment without existential quantifier. Translation of formulae from sorted logics to unsorted logics, usually called relativization, is sometimes used to clarify certain aspects of the sorted logic and to support completeness proofs for sorted calculi. Most of these procedures, however, were not seen as a translation process. Some of the authors even call the deductions in the translated syntax "meta reasoning".

The relational translation for modal logics is quite natural and should therefore be as old as the Kripke semantics itself. To my knowledge, the first one who has used it for a particular application, namely in natural language processing, is [Moore 80]. I have extended this method to the more expressive M-Logic with the ARP-predicates, mainly to illustrate the translation methodology itself with a not too complicated example.

The basic idea for the functional translation is to represent paths in the possible worlds structure by strings of terms. The only meaningful interpretation of these strings is as composition of functions mapping worlds to accessible worlds. Some authors have used these terms and the strings as additional labels of the modal operators without exploiting this interpretation. Graham Wrightson has used them in a tableaux calculus as a syntactical basis for an early version of a unification algorithm for modal contexts [Wrightson 83]. In [Chan 87] a resolution method for propositional S4 modal logic is defined which works on the original syntax, but with labelled operators. Lincoln Wallen has extended the matrix method of [Andrews 81] and [Bibel 81,82] to modal logics. He still works on the original syntax, but introduces also labelled operators. Wallen was the first who applied the special unification algorithm to these labels [Wallen 87].

The earliest work in the spirit of the translation idea seems to be Nakamatsu and Suzuki's method for translating modal formulae into two-sorted predicate logic. They considered mainly the S4 and S5 cases [Nakamatsu & Suzuki 82,84]. Recently some research groups independently of each other came up with almost the same idea for the functional translation. These are Luis Fariñas del Cerro and Andreas Herzig from Toulouse, Yves Auffray from Saint-cloud together with Patrice Enjalbert from Caen, and Peter Jackson and Han Reichgelt. For the classical first order modal logic case without equations and for serial accessibility relation their systems do not differ seriously. Jackson and Reichgelt use a sequent calculus on the target logic side with a special unification algorithm for modal terms (which I do not understand) [Jackson & Reichgelt 87]. Auffray and Enjalbert have characterized the properties of the accessibility relation with equations in a very similar way as I did. They do not insist on a particular calculus for the translated formulae [Auffray & Enjalbert 88, Auffray 89]. Herzig's system is - for the serial case - almost the same as I have presented in [Ohlbach 88], i.e. a resolution calculus with the special theory unification algorithms which have built in the properties of the accessibility relation. For the nonserial case Herzig's system does not generate END-literals during the translation, but treats the nonseriality dynamically during the resolution operation. To my knowledge this is the most compact and efficient treatment of nonseriality in a resolution calculus. Its integration in a resolution theorem prover, however, requires some modifications beyond the exchange of the unification algorithm.

Inspired by the first publication of my translation method, which did not take into account equality reasoning and which considered only reflexivity, symmetry and transitivity of the accessibility relation, Arild Waaler has extended it to the euclidean case and considered some aspects of paramodulation [Waaler 89].

For the treatment of logics with more complex operators, for example temporal UNTIL, EVENTUALLY - etc. operators, it turned out that the one step translation is too complicated. Therefore I developed an intermediate logic, called Context Logic, with predicate logic syntax, but possible worlds semantics. This logic allows to break up the translation into two steps where the second step, translation into OSPL, is independent of the actual source logic [Ohlbach 89]. The two step translation via Context Logic has been used to translate a quite complex temporal logic with a whole bunch of operators into predicate logic [Ohlbach 89]. Also Fagin and Halpern's Logic of Local Reasoning [Fagin & Halpern 88] has been translated into predicate logic via Context Logic [Waagbø 90].

Brand new is Dov Gabbay's "labelled deduction systems" approach to generate optimized calculi not from the semantics of the source logic, but from Hilbert calculi [Gabbay 90]. He marks formulae with labels which, for the modal logic case, correspond to the possible worlds, and in other logics to something else. The calculus permits the manipulation of the labels and the formulae independently and in parallel such that it is easy to switch to another logic by exchanging the labelling discipline.

8 CONCLUSION

We have presented a general framework for translating logical formulae from one logic into another logic. The methodology has been illustrated with two different approaches to translating a rather expressive modal logic into predicate logic. The modal logic we considered is a normal logic in the sense of [Chellas 80]. It is first order, many sorted with built in equality. It has the two modal operators $\Box$ and $\Diamond$ parametrized with arbitrary terms. Particular properties of the accessibility relation, namely seriality, reflexivity, symmetry, transitivity, euclideanness and linearity can be specified for the whole possible worlds structure or for parts of it by special built in predicates within the logic itself. Constant and function symbols may be rigid or flexible, i.e. change their meaning from world to world. We also allow varying domain interpretations where terms may denote objects existing in one world and not existing in another world.

The first translation method we presented is the well known "relational" translation which makes the modal logic's possible worlds structure explicit by introducing a distinguished predicate symbol to represent the accessibility relation. In the second approach, the "functional" translation method, paths in the possible worlds structure are represented by compositions of functions which map worlds to accessible worlds. This allows reasoning about larger parts of the possible worlds structure within a single call of the unification algorithm in a resolution calculus.

Both translation methods yield formulae which can be processed with a standard many sorted resolution and paramodulation calculus. That means predicate logic theorem provers or logic programming systems are now immediately applicable to modal logic.

Several optimizations of the functional translation have been sketched. The most effective optimization, however, the translation of the axioms describing the properties of the accessibility relation into theory unification and theory resolution rules has been worked out so far only for the fragment of the modal logic that corresponds to the classical modal systems K, D, T, DB, K4, D4, B, S4 and S5. The extension to the full logic used in this paper is in some aspects quite straightforward, in some aspects, in particular for linearity, really nontrivial. A first version has been developed by Andreas Nonnengart and will be published soon.

Acknowledgement

I wish to thank all my colleagues in Jörg Siekmann's research group and in the MEDLAR team who contributed to this work. Lincoln Wallen's talk about his matrix method, given in spring 1987 in Munich, originally inspired this work. During many discussions with Andreas Nonnengart the basic principles were clarified and the first versions of the unification algorithms were developed. Jörg Siekmann, my PhD supervisor, insisted on clear and understandable formalisms and formulations. His criticism forced me to change and simplify, and therefore to improve, considerable parts of the classical modal logic version of the formalism and proofs. I had many helpful and inspiring discussions with Andreas Herzig and Luis Fariñas del Cerro from Toulouse and Arild Waaler from Oslo. Finally I wish to thank Michael McRobbie from the Australian National University at Canberra who invited me for a research stay at ANU and whom I own the three most productive months in my life. Some parts of the translation method have been developed there.

References

Anderson & Bledsoe 70	R.Anderson, W.W. Bledsoe. *A linear format for resolution with merging and a new technique for establishing completeness*. Journal of ACM, vol. 17, pp. 525-534, 1970.
Andrews 81	P.B. Andrews. *Theorem-Proving via General Matings*. Journal of the Association for Computer Machinery, 28,2, pp. 193-214, 1981.
Auffray 89	Y. Auffray. *Résolution modal et logique des chemins*. Thèse de doctorat de l'université de Caen, Caen, 1989.
Auffray&Enjalbert 88	Y. Auffray, P. Enjalbert. *Démonstration de théorèmes en logiques modale - Un point de vue equationnel*. Journées Européennes Logique et Intelligence Artificielle, Roscoff, 1988.
Benanav 90	D. Benanav. *Simultaneous Paramodulation*. Proc. of 10th CADE, Springer Lecture Notes in Artificial Intelligence 449, pp. 442-455, 1990.
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.
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.
Chellas 80	B.F. Chellas. *Modal logic - an introduction*. Cambridge University Press, 1980.
Fagin & Halpern 88	R. Fagin, J.Y. Halpern. Belief, Awareness and Limited Reasoning. Artificial Intelligence 34, pp. 39-76, 1988.

Fitting 83	M.C. Fitting. *Proof methods for modal and intuitionistic logics.* Vol. 169 of Synthese Library, D. Reidel Publishing Company, 1983.
Gabbay 90	D. Gabbay. *Labelled Deduction Systems.* Imperial College, London, to be published.
Grätzer 79	G. Grätzer. *Universal Algebra.* Springer Verlag, 1979.
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.
Herzig 89	A. Herzig. *Raisonnement automatique en logique modale et algorithmes d'unification.* Thèse de doctorat de l'université Paul-Sabatier de Toulouse, 1989.
Jackson&Reichgelt 87	P. Jackson, H. Reichgelt. *A general proof method for first-order modal logic.* Proc. of Int. Joint Conference on Artificial Intelligence (IJCAI), pp. 942-944, 1987.
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.
Loecks&Sieber 84	J. Loecks, 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	A. Martelli, U. Montanari. *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.
Ohlbach 88	H.J. Ohlbach. *A Resolution Calculus for Modal Logics.* Proc. of 9th CADE, Argonne, Springer LNCS 310, pp. 500-516, 1988. Full version: SEKI Report SR-88-08, FB Informatik, Univ. of Kaiserslautern, Germany, 1988.
Ohlbach 89	H.J. Ohlbach. *Context Logic.* SEKI Report SR-89-08, FB Informatik, Univ. of Kaiserslautern, Germany, 1989. see also: H.J. Ohlbach. *Context Logic - An Introduction.*

	Proc. of GWAI-89, Geseke, Springer Informatik Fachberichte 216, pp. 27-36, 1989.
Ohlbach&Siekmann 89	H.J. Ohlbach, J.H. Siekmann. *The Markgraf Karl Refutation Procedure.* SEKI Report SR-89-19, FB Informatik, Univ. of Kaiserslautern, Germany, 1989.
Robinson 65	J.A. Robinson. *A Machine Oriented Logic Based on the Resolution Principle.* J.ACM, Vol. 12, No 1, pp. 23-41, 1965.
Robinson & Wos 69	Robinson, G., Wos, L. *Paramodulation and theorem proving in first order theories with equality.* Machine Intelligence 4, American Elsevier, New York, pp. 135-150, 1969.
Schmidt-Schauß 89	M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* Springer Lecture Notes in Artificial Intelligence 395, 1989.
Siekmann 89	J. Siekmann. *Unification Theory.* J. of Symbolic Computation 8, 1989.
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, pp. 333-356, 1985.
Waaler 89	A. Waaler. *Resolusion i modallogiske systemer for bruk i kunnskaps-teknologi.* Diploma thesis, Institutt for Datateknikk og Telematikk. Norges Tekniske Høykole, Trondheim, Norway, 1989.
Wagbø 89	G. Wagbø, *Logics and Semantics for Knowledge and Belief.* Diploma thesis. Norwegian Institute of Technology, Trondheim, 1989.
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.
Wrightson 83	G. Wrightson. On some tableau proof procedures for modal logic. Dissertation, Fak. f. Informatik, University of Karlsruhe. Also published by VDI-Verlag 1983.

INDEX