A Sorted Logic Using Dynamic Sorts

Christoph Weidenbach

MPI–I–91–218 December 1991

Author's Address

Christoph Weidenbach
Max-Planck-Institut für Informatik
Im Stadtwald
6600 Saarbrücken 11
Germany
weidenb@mpi-sb.mpg.de

Publication Notes

This report is a more detailed and elaborated version of a number of papers submitted to several conferences.

Acknowledgements

I would like to thank D. Fehrer, P. Graf, M. Keil, A. Nonnengart and H.J Ohlbach for many discussions on sorted deduction as well as for many helpful remarks directly concerning this report.

Abstract

We present a sound and complete calculus $\mathcal{CL}_{\mathcal{S}}$ for the first-order logic $\mathcal{L}_{\mathcal{S}}$ without equality but dynamic sorts. By dynamic sorts we understand a sort structure which is exploited during the deduction process and is not fixed from the beginning. Therefore the logic allows arbitrary sort declarations and every first-order logic formula can be compiled into our logic. Deductions in the calculus turn out to be very efficient as will be shown by examples.

Thus the new logic significantly improves the efficiency of automated deductions if one-place (predicates) sorts are involved in the reasoning process. Furthermore the calculus $\mathcal{CL}_{\mathcal{S}}$ is conservative extension of the existing approaches to sorted logic. That means if the sorted information could be represented in the known approaches the more general calculus $\mathcal{CL}_{\mathcal{S}}$ has the same behaviour as the existing calculi.

Chapter 1

Introduction

It is widely accepted that the addition of “sorts” to first-order logic results in a more efficient resolution calculus and a more natural representation of problems, e.g. see [Wal87, SS89, WO90, Coh87, Fri89, BHPS89]. Although the “naturalness” of a representation may depend on the personal taste, there are many examples in the literature, where a sorted formalization of a problem leads to shorter proofs and a lesser branching rate, i.e. a smaller search space (e.g. the first solution to Schubert's Steamroller [Wal84]).

Sorts correspond to unary predicates in unsorted first-order logic. For example in the logic of [WO90] the sort “declaration” $f(x_{Male}) \lessdot Male$ is logically equivalent to the clause $Male(x) \Rightarrow Male(f(x))$ (where f denotes the father function). This simple example shows the advantages of a sorted formalization: less literals and hence less resolution possibilities. In particular, for this simple example there are infinitely many (self) resolution possibilities in the unsorted version, whereas there are no possible resolution steps in the sorted version.

Most of the work done so far separates the information about sorts from the rest [Wal87, SS89, Coh87, Fri89, BHPS89]. The description of the sort structure, e.g. subsort relationships or how functions operate on sorts, is contained in what is called the SBOX (Sort Box) and this remains static during the deduction process. The rest of the formulas is put in an FBOX (Formula Box) and here standard deduction rules (e.g. resolution) are applied. The SBOX is either built up using a special language (e.g. the declaration $x_S \lessdot T$ expresses that S is a subsort of T, using the language of [WO90]) or a first-order language is restricted to sort symbols as predicates (e.g. $\forall x\ S(x) \Rightarrow T(x)$ [Fri89] also expressing the subsort relationship between S and T). In the FBOX no sort symbols are allowed as predicates. The connection between the SBOX and the FBOX is established by affixing a sort symbol to the variables. Variables are restricted to range over the denotation of their sort which is semantically viewed as a subset of the universe of discourse. Deduction rules (e.g. resolution) are modified by replacing the unsorted unification algorithm by a sorted version. The sorted unification algorithm checks the assignments to variables with respect to the information in the SBOX. In order so illustrate sorted resolution consider the following example (for the SBOX we use the syntax of [WO90]):

	SBOX
(1)	$peter \lessdot Male$
(2)	$mary \lessdot Woman$
(3)	$x_{Man} \lessdot Male$
(4)	$f(x_{Male}) \lessdot Male$

	FBOX
(5)	$Love(x_{Male}, mary)$
(6)	$\neg Love(f(f(peter)), y_{Woman})$

which is logically equivalent to the unsorted formalization

	UNSORTED
(1)	$Male(peter)$
(2)	$Woman(mary)$
(3)	$Man(x) \Rightarrow Male(x)$
(4)	$Male(x) \Rightarrow Male(f(x))$
(5)	$Male(x) \Rightarrow Love(x, mary)$
(6)	$Woman(y) \Rightarrow \neg Love(f(f(peter)), y)$

All variables are assumed to be universally quantified and the function f denotes the father of a person. There is only one resolution step between the clauses (5) and (6) in the sorted formalization leading to the unification problem [Sie89] $\Gamma = \{x_{Male} = f(f(peter)),\ y_{Woman} = mary\}$. The unification problem is trivially solved already, if we disregard the sort unification. However the sorted unification algorithm must also check the well sortedness of the assignments. It has to be proved that in every model satisfying the SBOX, $f(f(peter))$ is of sort *Male* and *mary* of sort *Woman*. In this example this is clearly the case. In general the proof is done by the sorted unification algorithm. In our case Γ is well sortedly solved and one resolution step yields the empty clause.

In the unsorted version of the example there are several possible resolution steps. Resolution between (5) and (6) results in the clause

$$(7) \quad \neg Male(f(f(peter))) \ \vee \ \neg Woman(mary)$$

which can be refuted using twice clause (4) and then the clauses (1) and (2). This simple example demonstrates the advantages of sorted deduction:

- Compact Representations
  In the sorted version we needed six literals (including the declarations in the SBOX) in the unsorted version ten.

- Short Deductions
  In the sorted version the clauses are refuted in one step, in the unsorted version we needed five steps. Of course we have also to take the steps of the sorted unification algorithm into account. But in practise the amount spent on the sorted unification can be neglected in comparison to the rest of the work. This topic will be discussed in more detail later on.

- Small Search Space
  In the sorted version there is one unique resolution step yielding the empty clause, in the unsorted version there are infinitely many possible steps. For example there are infinitely many (self) resolvents of clause (4).

In this report we present a new approach to sorted deduction which generalizes the existing ones: Arbitrary formulas are allowed in the SBOX and there is no longer a distinction between the SBOX and the FBOX. Sort declarations of the form $t \lessdot S$ (or sort predicates known from other sorted logics) can be used as ordinary predicates. Nevertheless the new sorted calculus which follows the ideas of [Wal87, SS89, WO90] significantly outperforms unsorted calculi. In addition the new approach is a conservative extension of the existing ones. That means if the information could be separated and represented in the known approaches (e.g. [Wal87, SS89]) the more general calculus has the same behaviour as the existing calculi.

Compared to the other approaches [Wal87, SS89, Fri89] strictly follow the SBOX, FBOX paradigm and [Coh87, BHPS89] allow sort information to occur in the FBOX, but they only consider the static SBOX information for the sorted reasoning. In addition all these approaches impose strong restrictions on the formulas in the SBOX in order to obtain a sound and complete calculus. This topic will be discussed in more detail in Section 1.1.1. In [WO90] the SBOX, FBOX scheme is still abolished but the calculus presented there is rather weak compared to the $\mathcal{CL_S}$ calculus which is established in this report. This topic will be discussed in more detail in Chapter 6.

The following sections give a an informal overview of the subsequent chapters. Results will be motivated and discussed but without any technical details. In Section 1.1 the new logic will be motivated and introduced. Section 1.2 explains the new calculus. The following sections are informal versions of the chapters 2-6. Chapter 7 introduces some refinements and further improvements on the calculus $\mathcal{CL_S}$. In Chapter 8 we solve some well known examples which are introduced in Section 3.3 and discuss efficiency aspects. The report is finished by the conclusions presented in Chapter 9.

1.1 The Sorted Logic $\mathcal{L_S}$

1.1.1 Motivation

The main difference between the new sorted logic $\mathcal{L_S}$ presented in this paper and existing approaches is that we abolish the SBOX, FBOX paradigm. There are at least three reasons to pull down the barrier between SBOX and FBOX: (i) expressivity, (ii) Skolemization and (iii) disjunctions in the SBOX.

In order to exemplify the first problem, namely expressivity consider the phrase which says that a male is a man if he has written a book, begotten a child and planted a tree. Coding this example gives the formula:

$$\begin{gathered} \mathit{Write}(x_{\mathit{Male}}, y_{\mathit{Book}}) \;\wedge\; \mathit{Plant}(x_{\mathit{Male}}, z_{\mathit{Tree}}) \;\wedge\; \mathit{Beget}(x_{\mathit{Male}}, u_{\mathit{Child}}) \\ \Rightarrow x_{\mathit{Male}} \lessdot \mathit{Man} \end{gathered}$$

This formula does not fit into the SBOX, FBOX scheme, because ordinary two-place predicates are used to define a sort. Thus the formula can not be represented adequately in existing approaches. Formulas of the above kind are not artificial, e.g. they frequently occur in knowledge bases of the KL-ONE [BS85] family.

The next problem is Skolemization, the elimination of existential quantifiers in refutation based deduction systems. The proper Skolemization (see Chapter 4) of the FBOX formula

$$\exists y_{\mathit{Woman}}\; \mathit{Love}(\mathit{peter}, y_{\mathit{Woman}}) \;\vee\; \mathit{Love}(\mathit{peter}, \mathit{Bill})$$

is

$$(\mathit{Love}(\mathit{peter}, a) \;\wedge\; a \lessdot \mathit{Woman}) \;\vee\; \mathit{Love}(\mathit{peter}, \mathit{Bill})$$

where a is a new Skolem constant. Again the SBOX, FBOX framework is violated because declarations (or sort predicates) are not allowed in the FBOX. Putting the declaration $a \lessdot \mathit{Woman}$ outside the FBOX and into the SBOX results in a Skolemization which does not preserve satisfiability, because $a \lessdot \mathit{Woman}$ in the SBOX forces the sort Woman to be non empty in every model whereas there are models of the input formula which assign the empty set to the sort Woman. However empty sorts are not the only reason that disallow declarations resulting from skolemization in the SBOX and in general we need strong restrictions (e.g. see [Wal87, SS89]) to be imposed on the SBOX in order to place these declarations into the SBOX.

The third problem arises from disjunctions in the SBOX. If there are formulas like

$$\mathit{peter} \lessdot \mathit{Man} \;\vee\; \mathit{peter} \lessdot \mathit{Woman}$$

in the SBOX then there is no initial model for the SBOX. A least model either includes $\mathit{peter} \lessdot \mathit{Man}$ or $\mathit{peter} \lessdot \mathit{Woman}$. From the formula

$$\forall x_{\mathit{Man}}\; \mathit{Love}(x_{\mathit{Man}}, \mathit{mary})$$

we must be able to deduce

$$\mathit{Love}(\mathit{peter}, \mathit{mary}) \;\vee\; \mathit{peter} \lessdot \mathit{Woman}$$

in order to obtain a complete calculus. Again this clause does not fit into the SBOX, FBOX paradigm.

All these problems can be overcome if the artificial barrier between SBOX an FBOX is eliminated: SBOX expressions are also FBOX expressions and formulas denoting sort relationships can be combined with ordinary formulas. This logic is called $\mathcal{L}_{\mathcal{S}}$.

1.1.2 Syntax of $\mathcal{L}_S$

A signature $\mathbf{\Sigma}{:}= (\mathbf{S}, \mathbf{V}, \mathbf{F}, \mathbf{P})$ of the sorted logic $\mathcal{L}_S$ contains a set of sort symbols in addition to the usual sets $\mathbf{V}$ of variable symbols, $\mathbf{F}$ of function symbols and $\mathbf{P}$ of predicate symbols. The fixed 2-place predicate $\lessdot$ is always in $\mathbf{P}$. Variables are indexed with their sort, e.g. x_S, y_V.

Terms and atoms are built in the usual unsorted way with the addition that if t is a term and S a sort symbol, then $t{\lessdot}S$ is an atom. Atoms of the form $t{\lessdot}S$ are called *declarations*. From atoms we construct literals and formulas using the logical connectives. A clause is a set of literals which is interpreted as the universal closure of the disjunction of the literals.

The set of all terms is denoted by $\mathbf{T_\Sigma}$. Substitutions are total functions $\sigma{:}\,\mathbf{V} \to \mathbf{T_\Sigma}$ such that the set $DOM(\sigma){:}= \{x_S \mid \sigma(x_S) \neq x_S\}$ is finite. The application of substitutions can be extended to terms, formulas, and clauses in the usual way.

1.1.3 Semantics of $\mathcal{L}_S$

In order to define the semantics for the new logic, a set of objects from the non-empty universe has to be assigned to every sort symbol and a total function to every function symbol. The symbol $\lessdot$ is interpreted as the membership relation, whereas predicates and the logical connectives are interpreted as in unsorted first-order logic.

Let $\mathbf{\Sigma}$ be a signature. A $\mathbf{\Sigma}$-*algebra* $\mathcal{A}$ consists of a non-empty carrier set $\mathbf{A}$, a total function $f_\mathcal{A}{:}\,\mathbf{A}^n \to \mathbf{A}$ for every function symbol $f{\in}\mathbf{F}_n$, a set $S_\mathcal{A} \subseteq \mathbf{A}$ for every sort S.

An interpretation $\Im$ [1] assigns an n-ary relation $P_\Im \subseteq \mathbf{A}^n$ to every n-place predicate symbol P and the membership relation to $\lessdot$, such that for every formula $\mathcal{F}$, $t{\in}\mathbf{T_\Sigma}$, $S{\in}\mathbf{S}$, and $P{\in}\mathbf{P}_n$:

- $\Im \models P(t_1, \ldots, t_n)$ iff $(\Im_h(t_1), \ldots, \Im_h(t_n)){\in}P_\Im$.
- $\Im \models t{\lessdot}S$ iff $\Im_h(t){\in}S_\mathcal{A}$
- $\Im \models \forall x_S\ \mathcal{F}$ iff for all $a{\in}S_\mathcal{A}, \Im\{x_S/a\} \models \mathcal{F}$
- $\Im \models \exists x_S\ \mathcal{F}$ iff there exists an $a{\in}S_\mathcal{A}, \Im\{x_S/a\} \models \mathcal{F}$
- The logical connectives are interpreted as usual.

[1]An interpretation $\Im\{x_S/a\}$ is like $\Im$ except that it maps x_S to a

1.2 Calculi for Sorted Logics

1.2.1 Existing Calculi

Logics following the SBOX, FBOX paradigm [Wal87, SS89, Fri89, Coh87, BHPS89] construct a free model from the declarations in the SBOX[2]. In order to guarantee the existence of a free model only unit declarations of the form $t{<}S$ are allowed in the SBOX. The free model is the set $\mathbf{T}_{\mathbf{L}_{<}}$ of well sorted terms where $\mathbf{L}_{<}$ is the set of all declarations occurring in the SBOX.

Definition 1.2.1 (Well Sorted Terms) Let $\mathbf{L}_{<}$ be a set of declarations. Then the set of well sorted terms $\mathbf{T}_{\mathbf{L}_{<},S}$ of sort S is recursively defined by:

- for every variable x_S, $x_S{\in}\mathbf{T}_{\mathbf{L}_{<},S}$
- for every declaration $(t{<}S){\in}\mathbf{L}_{<}$ with $V(t) = \{x_{S_1}, \ldots, x_{S_n}\}$ and terms $t_i{\in}\mathbf{T}_{\mathbf{L}_{<},S_i}$, $\sigma{:}= \{x_{S_1}{\mapsto}t_1, \ldots, x_{S_n}{\mapsto}t_n\}$, $\sigma(t){\in}\mathbf{T}_{\mathbf{L}_{<},S}$

The function V maps terms to their variables, substitutions σ are enumerated by the variable term pairs (e.g. $x_{S_1}{\mapsto}t_1$) of their finite domain ($DOM(\sigma)$). Computations in the free model are very efficient (this is one serious reason why sorted unification is always more efficient than the corresponding unsorted resolution). It is decidable in polynomial time and space [SS89] if $t{\in}\mathbf{T}_{\mathbf{L}_{<},S}$ for an arbitrary term t, sort S, and set of declarations $\mathbf{L}_{<}$.

Applying the definition of well sorted terms to the initial example, we get $\mathbf{L}_{<} = \{\mathit{peter}{<}\mathit{Male},\ \mathit{mary}{<}\mathit{Woman},\ x_{\mathit{Man}}{<}\mathit{Male},\ f(x_{\mathit{Male}}){<}\mathit{Male}\}$ and $\mathbf{T}_{\mathbf{L}_{<},\mathit{Male}} = \{\mathit{peter},\ f(\mathit{peter}),\ f(f(\mathit{peter})),\ \ldots,\ x_{\mathit{Male}},\ y_{\mathit{Male}},\ \ldots,\ f(x_{\mathit{Male}}),\ \ldots\}$

The sorted unification algorithm is an extension of the known unsorted unification algorithm. In addition to the unsorted algorithm variables are weakened until for every assignment $x_S \mapsto t$ we have $t{\in}\mathbf{T}_{\mathbf{L}_{<},S}$. The sorted resolution rule is an extension of the unsorted resolution rule by sorted unification.

1.2.2 The New Sorted Calculus $\mathcal{CL}_{\mathcal{S}}$

The starting point for the new calculus is a set CS of clauses (in Chapter 4 we show the transformation into clause normal form). Clearly the traditional means of sorted reasoning do not apply to this expressive sorted logic $\mathcal{L}_{\mathcal{S}}$ since in general there exist no free model for the declarations occurring in a clause set. Even more, as declarations can be combined with arbitrary first-order formulas well sortedness can not be decided globally. Thus it is not clear how sorted unification can be established. Nevertheless the information about sorts for the unification algorithm is contained in the declarations occurring in the clause set. The open question is which declarations must be considered for the unification algorithm. Clearly

[2]Frisch [Fri89] does not give a syntactic characterization of well sortedness and hence the question how to check well sortedness remains open

all declarations occurring in unit clauses must be chosen (e.g. $\mathit{peter} \lessdot \mathit{Man}$ from the initial example) because every model satisfying the clause set satisfies these declarations. Additionally a translation of the sort structures of [Wal87, SS89] into $\mathcal{L}_{\mathcal{S}}$ results in unit clauses that are declarations.

But if we choose only declarations that occur in unit clauses, this would result in an incomplete calculus as the following example shows.

	SBOX-FBOX
(1)	$\mathit{peter} \lessdot \mathit{Man} \;\vee\; \mathit{peter} \lessdot \mathit{Woman}$
(2)	$\mathit{Love}(x_{\mathit{Man}}, \mathit{paul})$
(3)	$\mathit{Love}(y_{\mathit{Woman}}, \mathit{mary})$
(4)	$\neg\mathit{Love}(\mathit{peter}, \mathit{paul})$
(5)	$\neg\mathit{Love}(\mathit{peter}, \mathit{mary})$

The clause set is unsatisfiable. There is no unit declaration because the only declarations $\mathit{peter} \lessdot \mathit{Man}$ and $\mathit{peter} \lessdot \mathit{Woman}$ occur in the same clause. Thus if we only consider unit declarations, there is no well sorted resolution step. The sorted unification algorithm has no information about sorts. Thus neither the assignment $x_{\mathit{Man}} \mapsto \mathit{peter}$ nor $y_{\mathit{Woman}} \mapsto \mathit{peter}$ is well sorted. But to obtain a complete calculus we must consider at least one of the declarations for unification. This can not be done in the standard way (see Section 1.2.1) because not every model satisfying the clause (1) satisfies the chosen declaration[3]. The other literal can not be discarded, it must be taken as a residue literal for the chosen declaration similar to a partial theory resolution step [Sti85]. A declaration occurring in a non unit clause is called a *conditioned* declaration and denoted as a pair $(t \lessdot S, C)$ where C is a set of literals. The idea is to use the declaration as for sorted unification and to collect the conditions during the computation of well sortedness.

Selecting the set of declarations $\mathbf{L}_{\lessdot} = \{(\mathit{peter} \lessdot \mathit{Man}, \{\mathit{peter} \lessdot \mathit{Woman}\})\}$ for unification, a resolution step between (2) and (4) is possible as the assignment $x_{\mathit{Man}} \mapsto \mathit{peter}$ is well sorted with respect to the chosen declaration $(\mathit{peter} \lessdot \mathit{Man}, \{\mathit{peter} \lessdot \mathit{Woman}\})$. But as not every model satisfies the chosen declaration the result is the only condition

$$(6) \quad \mathit{peter} \lessdot \mathit{Woman}$$

Intuitively clause (6) means that if we assume *peter* to be a *Man* we get a contradiction between (2) and (4). Therefore we conclude *peter* to be of sort *Woman*. This new clause is a unit declaration and subsumes clause (1). We must change the information of the unification algorithm into $\mathbf{L}_{\lessdot} = \{(\mathit{peter} \lessdot \mathit{Woman}, \emptyset)\}$. Now a resolution step between clause (3) and clause (5) is possible, yielding the empty clause.

If we would have started with the set of declarations $\mathbf{L}_{\lessdot} = \{(\mathit{peter} \lessdot \mathit{Woman}, \{\mathit{peter} \lessdot \mathit{Man}\})\}$ we would have made the same resolution steps in reverse order.

[3]An alternative approach was presented in [WO90] where residue literals were added for each assignment in order to get sound inference rules.

The example shows that for the $\mathcal{CL_S}$ calculus, the sort information must be changed dynamically during the deduction process and non unit declarations have to be considered.

The conditioned declarations are needed for a generalized notion of well sortedness where a term is of a specific sort under certain conditions. Thus conditioned declaration cause conditioned terms (pairs of the kind (t, C) where t is a term and C a set of literals) and conditioned substitutions (pairs of the kind (σ, C) where σ is a substitution and C a set of literals).

Definition 1.2.2 (Conditioned Well Sorted Terms) Let $\mathbf{L}_{\lessdot}$ be a set of conditioned declarations. Then the set of conditioned well sorted terms $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ of sort S is recursively defined by:

- for every variable x_S, $(x_S, \emptyset) \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$
- for every conditioned declaration $(t \lessdot S, C) \in \mathbf{L}_{\lessdot}$ with $V(t) = \{x_{S_1}, \ldots, x_{S_n}\}$ and conditioned terms $(t_i, C_i) \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S_i}$, $\sigma := \{x_{S_1} \mapsto t_1, \ldots, x_{S_n} \mapsto t_n\}$, $(\sigma(t), \sigma(C) \cup C_1 \ldots \cup C_n) \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$

Questions of the form $t \lessdot \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ are again decidable in polynomial time and space with respect to the number of symbols in t and the number of declarations in $\mathbf{L}_{\lessdot}$. The new sorted unification algorithm is an extension of the known sorted unification algorithms (for an overview see [SS89]) by conditioned well sorted terms. A presentation of the algorithm can be found in Section 6.1. The result of the algorithm is a conditioned well sorted unifier including the set of conditions that are needed on order to establish the conditioned well sortedness. The resolution rule is then:

Definition 1.2.3 (Resolution Rule) Let C_1 and C_2 be two clauses with no variables in common, $L \in C_1$ and $K \in C_2$. If there exists a conditioned well sorted mgu $\sigma^{\mathbf{c}} = (\sigma, D)$ of L and K, such that $\sigma(L)$ and $\sigma(K)$ are two complementary literals, then

$$\sigma((C_1 \setminus \{L\}) \cup (C_2 \setminus \{K\})) \cup D$$

is a *resolvent* of C_1 and C_2.

A conditioned substitution $\sigma^{\mathbf{c}} = (\sigma, D)$ consists of the substitution σ and the set of conditions D. The sorted factorization rule is defined accordingly. The correctness of the inference rules follows immediately from the correctness of conditioned well sorted terms and the form of the rules. The proof of the following completeness theorem is given in Section 6.3:

Theorem 1.2.4 (Completeness Theorem) Let CS be a clause set of $\mathcal{L_S}$ and let $\mathbf{L}_{\lessdot} := \{(t_i \lessdot S_i, C_i')\}$ such that for each clause $C_i \in CS$ consisting of declarations we choose exactly one declaration $t_i \lessdot S_i$ with $C_i = \{t_i \lessdot S_i\} \cup C_i'$.

If CS is unsatisfiable, there exists a derivation of the empty clause using resolution and factorization.

Now the refutation procedure works as follows: a set $\mathbf{L}_{\leqslant}$ of declarations (see Theorem 1.2.4) from the given clause set CS is chosen and the inference rules are applied. If a new clause consisting of declarations only is derived, the set $\mathbf{L}_{\leqslant}$ of declarations is updated. This process is repeated until the empty clause is obtained.

Chapter 2

Syntax of $\mathcal{L}_{\mathcal{S}}$

The Syntax of $\mathcal{L}_{\mathcal{S}}$ is very similar to the syntax of unsorted first-order logic, because we have to allow for "ill-sorted" terms. Thus we generate terms, atoms, and formulas without considering sorts. Only the special relation $\lessdot$ is added and variables are indexed with the sort they range over.

2.1 Signature

Definition 2.1.1 (Signature) A signature $\mathbf{\Sigma} := (\mathbf{S}, \mathbf{V}, \mathbf{F}, \mathbf{P})$ consists of the following disjoint sets:

- $\mathbf{S}$ is a finite set of sort symbols, the fixed symbol $\top$ for "any" is always in $\mathbf{S}$. Sort symbols are S, T.
- $\mathbf{V}$ is an infinite set of countably many variable symbols. The symbols x, y, z represent variables.
- $\mathbf{F}$ is a countably infinite set of function symbols. $\mathbf{F}$ is divided into the sets of n–place function symbols $\mathbf{F}_n$ ($n \in \mathbb{N}_0$). We use the f, g, h as function symbols and a, b, c as constant symbols (0-place function symbols).
- $\mathbf{P}$ is a finite set of predicate symbols divided into the sets of n–place predicate symbols $\mathbf{P}_n$. Predicates are named by P, Q.

Additionally, we assume a function *sort*: $\mathbf{V} \to \mathbf{S}$ such that for every sort $T \in \mathbf{S}$, there exist countably infinite many variables $x \in \mathbf{V}$, with $\mathit{sort}(x) = T$. In this case we say x has sort T which is denoted by x_T.

In order to simplify the notation for variables, we allow to write x_S and x_T in the same formula even if they are supposed to be different. Using this writing convention two variables are equal if they consist of the same variable symbol and have the same sort.

Definition 2.1.2 (Special Symbols) The following symbols are available:

- The logical connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists$
- The auxiliary symbols "(", ")", ", "
- The special relation $\lessdot$ which is used to represent sort information.

Assumption 2.1.3 (Non-empty Universe) We always assume that there is at least one constant symbol in $\mathbf{\Sigma}$. This assumption implies that the universe of discourse and the sort $\top$ are not empty. This appears natural to us, because we have the same assumption in unsorted first-order logic.

2.2 Terms, Atoms, and Formulas

Definition 2.2.1 (Terms) The set of all terms $\mathbf{T}_{\mathbf{\Sigma}}$ is inductively constructed by the following two rules:

- $x \in \mathbf{T}_{\mathbf{\Sigma}}$, if $x \in \mathbf{V}$
- $f(t_1, \ldots, t_n) \in \mathbf{T}_{\mathbf{\Sigma}}$, if $f \in \mathbf{F}_n$ and $t_i \in \mathbf{T}_{\mathbf{\Sigma}}$ for every i

Let $V(t)$ denote the set of variables occurring in a term t, i. e. $V(t) := \{t\}$, if t is a variable and $V(t) := \bigcup_{1 \le i \le n} V(t_i)$, if $t = f(t_1, \ldots, t_n)$. We can naturally extend V to atoms, literals, clauses, and other objects. An object t having $V(t) = \emptyset$ is called *ground*.

Definition 2.2.2 (Atoms) If $P \in \mathbf{P}_n$ and $t_1, \ldots, t_n$ are terms then $P(t_1, \ldots, t_n)$ is an *atom*. If S is a sort symbol and t is a term, then $t \lessdot S$ is an *atom*. An atom of the form $t \lessdot S$ is called a *declaration*.

Definition 2.2.3 (Literals) Atoms and their negations are *literals*. A literal is called *negative* if it consists of an atom and a negation symbol. Otherwise it is called *positive*. The meta level function *atom* maps literals to their atoms. For literals we write L, K.

Definition 2.2.4 (Clauses) A *clause* is a finite set of literals. It is interpreted as the disjunction of its literals where the whole clause is universally quantified over all occurring variables. For clauses we write $C = \{L_1, L_2\}$. The empty clause is denoted by $\square$.

If C is a clause then $\forall(C)$ denotes the formula obtained by building the disjunction of all literals in C and closing all variables universally. For example if $C = \{P(x_S), \neg Q(x_S, f(y_T))\}$ then $\forall(C) = \forall x_S\, \forall y_T\, (P(x_S) \vee \neg Q(x_S, f(y_T)))$. If we want a variable in C not to be universally closed, we write $\forall[x_S]\,(C)$, i.e. for the example $\forall[x_S]\,(C) = \forall y_T\, (P(x_S) \vee \neg Q(x_S, f(y_T)))$.

Definition 2.2.5 (Formulas) Every literal is a *formula.* If $\mathcal{F}$ and $\mathcal{G}$ are formulas then $(\neg\mathcal{F}), (\mathcal{F}\wedge\mathcal{G}), (\mathcal{F}\vee\mathcal{G}), (\mathcal{F}\Rightarrow\mathcal{G})$ and $(\mathcal{F}\Leftrightarrow\mathcal{G})$ are *formulas.* If $\mathcal{F}$ is a formula and x_T a variable then $(\forall x_T\ \mathcal{F})$ and $(\exists x_T\ \mathcal{F})$ are *formulas.* We omit parenthesis whenever possible using the following precedence starting from the logical connective with the highest precedence to the lowest one: $\neg, \forall, \exists, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

Definition 2.2.6 (Substitutions) A *substitution* σ is a total function $\sigma\colon \mathbf{V}\to\mathbf{T}_{\mathbf{\Sigma}}$, such that the set $\{\, x_S{\in}\mathbf{V} \mid \sigma(x_S)\neq x_S \,\}$ is finite.

Let $DOM(\sigma)\colon= \{\, x_S{\in}\mathbf{V} \mid \sigma(x_S)\neq x_S \,\}$. Since every substitution σ is uniquely determined by its action on the variables of $DOM(\sigma)$, it can be represented as a finite set of variable–term pairs $\{x_{S_1}\mapsto t_1,\ldots,x_{S_n}\mapsto t_n\}$, where $DOM(\sigma) = \{x_{S_1},\ldots,x_{S_n}\}$. We can extend the application of σ to $\mathbf{T}_{\mathbf{\Sigma}}$ by $\sigma(t)\colon=\sigma(t)$, if $t{\in}\mathbf{V}$ and $\sigma(t)\colon= f(\sigma(t_1),\ldots,\sigma(t_n))$, if $t = f(t_1,\, \ldots,\, t_n)$. Similarly, the application of σ can be extended to other objects, e.g. literals, clauses or sets of such objects. Let $COD(\sigma)\colon=\sigma(DOM(\sigma))$ and $I(\sigma)\colon= V(COD(\sigma))$. A substitution σ is called *ground* if $I(\sigma)=\emptyset$.

The composition of two substitutions $\sigma = \{x_{S_1}\mapsto t_1,\ldots,x_{S_n}\mapsto t_n\}$ and $\tau = \{y_{T_1}\mapsto s_1,\ldots,y_{T_m}\mapsto s_m\}$ can be computed by $\sigma\tau = \{y_{T_1}\mapsto\sigma(s_1),\ldots,y_{T_m}\mapsto\sigma(s_m)\}\cup\{\, x_{S_i}\mapsto t_i \mid x_i{\in}(DOM(\sigma)-DOM(\tau)) \,\}$. A substitution σ is called *idempotent* if $\sigma\sigma=\sigma$. Note that a substitution σ is idempotent iff $DOM(\sigma)\cap I(\sigma)=\emptyset$. With $\sigma[x_S]$ we denote a substitution that is equal to σ, except that it maps x_S to x_S.

Definition 2.2.7 (Subformula, Scope, Positive Component) Let $\mathcal{F},\mathcal{G},\mathcal{H}$ be formulas. $\mathcal{F}$ is a *subformula* of $\mathcal{G}$, if $\mathcal{F}$ is $\mathcal{G}$ or a formula that occurs within $\mathcal{G}$. $\mathcal{F}$ (or a variable x_T) is *in the scope* of a universal quantifier $\forall$ (an existential quantifier $\exists$, a conjunction $\wedge$,...) iff $\mathcal{F}$ is a subformula of $\mathcal{G}$ (or $\mathcal{H}$) in $\forall x_S\ \mathcal{G}$ ($\exists x_S\ \mathcal{G}, \mathcal{G}\wedge\mathcal{H},\ldots$). An occurrence of a subformula $\mathcal{F}$ in a formula $\mathcal{G}$ is a *positive component* of $\mathcal{G}$ iff $\mathcal{G}$ contains no implication or equivalence symbols and the occurrence of $\mathcal{F}$ is not in the scope of a negation symbol.

Definition 2.2.8 (Free, Bound Variables) For a formula $\mathcal{F}$, the set $FV(\mathcal{F})$ of free variables of $\mathcal{F}$ [Gal86] is defined by:

- $FV(P(t_1,\, \ldots,\, t_n)) = V(t_1)\cup\ldots\cup V(t_n)$ for an n–place predicate P
- $FV(t{<}S) = V(t)$ for the special relation $<$
- $FV(\neg\mathcal{F}) = FV(\mathcal{F})$
- $FV(\mathcal{F}\star\mathcal{G}) = FV(\mathcal{F})\cup FV(\mathcal{G})$, where $\star{\in}\{\wedge,\vee,\Rightarrow,\Leftrightarrow\}$
- $FV(\forall x_T\ \mathcal{F}) = FV(\mathcal{F})\setminus\{x_T\}$
- $FV(\exists x_T\ \mathcal{F}) = FV(\mathcal{F})\setminus\{x_T\}$

Given a formula $\mathcal{F}$, the set $BV(\mathcal{F})$ of bound variables of $\mathcal{F}$ is given by:

- $BV(P(t_1,\, \ldots,\, t_n)) = \emptyset$ for an n–place predicate P

- $BV(t \lessdot S) = \emptyset$ for the special relation $\lessdot$
- $BV(\neg\mathcal{F}) = BV(\mathcal{F})$
- $BV(\mathcal{F} \star \mathcal{G}) = BV(\mathcal{F}) \cup BV(\mathcal{G})$, where $\star \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
- $BV(\forall x_T\ \mathcal{F}) = BV(\mathcal{F}) \cup \{x_T\}$
- $BV(\exists x_T\ \mathcal{F}) = BV(\mathcal{F}) \cup \{x_T\}$

A variable x_T is *free* within a formula $\mathcal{F}$ if $x_T \in FV(\mathcal{F})$ and x_T is called *bound* within a formula $\mathcal{F}$ if $x_T \in BV(\mathcal{F})$. Note that a variable can occur both free and bound in a formula. For example the variable x_S in the formula $\forall x_S\ P(x_S)\ \vee\ Q(x_S)$ has this property.

Assumption 2.2.9 (Considering only Sentences) We only consider formulas without free variables. Such formulas are called *sentences.*

Chapter 3

Semantics

In order to define the semantics for $\mathcal{L}_{\mathcal{S}}$, we have to assign a set of objects of the non-empty universe to every sort symbol, a total function to every function symbol, $\lessdot$ is interpreted as the membership relation and we interpret predicate symbols and the logical connectives in the usual way.

3.1 Algebras

Definition 3.1.1 ($\boldsymbol{\Sigma}$-algebra) Let $\boldsymbol{\Sigma}$ be a signature. A $\boldsymbol{\Sigma}$-*algebra* $\mathcal{A}$ consists of a non-empty carrier set $\mathbf{A}$, a total function $f_{\mathcal{A}}: \mathbf{A}^n \rightarrow \mathbf{A}$ for every function symbol $f \in \mathbf{F}_n$, a set $S_{\mathcal{A}} \subseteq \mathbf{A}$ for every sort S and $\top_{\mathcal{A}} := \mathbf{A}$.

Definition 3.1.2 ($\boldsymbol{\Sigma}$-assignment) Let $\mathcal{A}$ be a $\boldsymbol{\Sigma}$-algebra. A $\boldsymbol{\Sigma}$-*assignment* is a partial mapping $\varphi: \mathbf{V} \rightarrow \mathbf{A}$ with $\varphi(x) \in \mathit{sort}(x)_{\mathcal{A}}$ if $x \in \mathcal{D}(\varphi)$[1]. The *homomorphic extension* $\varphi_h: \mathbf{T}_{\boldsymbol{\Sigma}} \rightarrow \mathbf{A}$ of a $\boldsymbol{\Sigma}$-assignment φ is defined as follows:

- $\varphi_h(x) := \varphi(x)$ for all $\boldsymbol{\Sigma}$-variables x
- $\varphi_h(f(t_1, \ldots, t_n)) := f_{\mathcal{A}}(\varphi_h(t_1), \ldots, \varphi_h(t_n))$ if $t_i \in \mathcal{D}(\varphi_h)$ and we define $f(t_1, \ldots, t_n) \in \mathcal{D}(\varphi_h)$

From now on we don't distinguish between φ and φ_h. Let φ be a $\boldsymbol{\Sigma}$-assignment, $a \in S_{\mathcal{A}}$, $x \in \mathbf{V}$ with $\mathit{sort}(x) = S$ and $t \in \mathbf{T}_{\boldsymbol{\Sigma}}$ then

- $\varphi[\{x/a\}](t) := a$ if $t \in \mathbf{V}$ and $t = x$.
- $\varphi[\{x/a\}](t) := \varphi(t)$ if $t \in \mathbf{V}$ and $t \neq x$.
- $\varphi[\{x/a\}](t) := f_{\mathcal{A}}(\varphi[\{x/a\}](t_1), \ldots, \varphi[\{x/a\}](t_n))$ if $t = f(t_1, \ldots, t_n)$.

That means $\varphi[\{x/a\}]$ is like φ except that it maps x to a.

[1] $\mathcal{D}(\varphi)$ denotes the domain of the function φ

3.2 Structures and Interpretations

Definition 3.2.1 (Σ-structure) Let $\mathbf{\Sigma}$ be a signature. A $\mathbf{\Sigma}$-*structure* $\mathcal{M}$ is a $\mathbf{\Sigma}$-algebra $\mathcal{A}$ which has additional denotations $P_{\mathcal{M}}$ for every predicate symbol $P{\in}\mathbf{P}_n$ and the special symbol $<$, such that:

- $P_{\mathcal{M}}$ is a relation with $P_{\mathcal{M}} \subseteq \mathbf{A}^n$
- $<_{\mathcal{M}}$ is the membership relation, i.e. $<_{\mathcal{M}} = \in$

Definition 3.2.2 (Interpretation) Let $\mathcal{M}$ be a $\mathbf{\Sigma}$-structure and $\varphi\colon \mathbf{T_\Sigma} \to \mathbf{A}$ a $\mathbf{\Sigma}$-assignment. The pair $\Im = (\mathcal{M}, \varphi)$ is called an *interpretation* and we define the satisfiability relation $\models$ between $\Im$ and formulas as follows: let $\mathcal{F}$ be a formula, $s, t{\in}\mathbf{T_\Sigma}, T{\in}\mathbf{S}$ and $P{\in}\mathbf{P}_n$:

- $\Im(t) = \varphi(t)$
- $\Im \models P(t_1, \ldots, t_n)$ iff $t_i{\in}\mathcal{D}(\varphi)$ and $(\varphi(t_1), \ldots, \varphi(t_n)){\in}P_{\mathcal{M}}$.
- $\Im \models s{<}T$ iff $s{\in}\mathcal{D}(\varphi)$ and $\varphi(s){\in}T_{\mathcal{M}}$.
- $\Im \models \forall x_S\ \mathcal{F}$ iff for all $a{\in}S_{\mathcal{A}}, (\mathcal{M}, \varphi[\{x_S/a\}]) \models \mathcal{F}$
- $\Im \models \exists x_S\ \mathcal{F}$ iff there exists an $a{\in}S_{\mathcal{A}}, (\mathcal{M}, \varphi[\{x_S/a\}]) \models \mathcal{F}$
- The logical connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ are interpreted in the usual way.

Note that the $\mathbf{\Sigma}$-assignment φ is defined for every term occurring in a sentence.

Definition 3.2.3 (Σ-model, satisfiable, unsatisfiable, well-sorted) Let $\mathcal{F}$ be a formula. An interpretation $\Im$ is a $\mathbf{\Sigma}$-*model* for $\mathcal{F}$ if $\Im \models \mathcal{F}$. A set F of formulas has a $\mathbf{\Sigma}$-model $\Im$ if for every $\mathcal{F}{\in}F, \Im \models \mathcal{F}$. A formula is called *satisfiable* if it has a $\mathbf{\Sigma}$-model. We call a formula *unsatisfiable* if it has no $\mathbf{\Sigma}$-model.

3.3 Examples

Example 3.3.1 (Schubert's Steamroller) The first example is the standard example for the first complete order-sorted calculus given by Walther [Wal87].

> Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars, but not snails. Caterpillars and snails like to eat some plants. Therefore there is an animal that likes to eat a grain-eating animal.

This problem has been coded in first-order logic in many different ways. We adopt a formulation proposed by Mark Stickel in [Sti86] where "grain-eating animal" refers to an animal that eats some grain. We introduce two binary relations E and M, where $E(x, y)$ means "x likes to eat y" and $M(u, v)$ means "u is much smaller than v" and the sort symbols G, P, C, B, S, F, W, A for grains, plants, caterpillars, birds, snails, foxes, wolves, and animals respectively. In order to have only non-empty sorts we introduce a constant for each of them. Then the following formulas represent the steamroller problem:

1) $\forall x_W\ x_W \lessdot A \quad\wedge\quad \forall x_F\ x_F \lessdot A \quad\wedge\quad \forall x_B\ x_B \lessdot A \quad\wedge\quad \forall x_C\ x_C \lessdot A \quad\wedge\quad \forall x_S\ x_S \lessdot A$
2) $\mathit{lupo} \lessdot W \quad\wedge\quad \mathit{foxy} \lessdot F \quad\wedge\quad \mathit{tweety} \lessdot B \quad\wedge\quad \mathit{raupy} \lessdot C \quad\wedge\quad \mathit{schnecky} \lessdot S$
3) $\mathit{muesli} \lessdot G \quad\wedge\quad \forall x_G\ x_G \lessdot P$
4) $\forall x_A\ (\forall y_P\ E(x_A, y_P)\ \vee\ \forall z_A\ ((M(z_A, x_A)\ \wedge\ \exists v_P\ E(z_A, v_P)) \Rightarrow E(x_A, z_A)))$
5) $\forall x_C, y_S, z_B, x'_F, y'_W\ (M(x_C, z_B)\ \wedge\ M(y_S, z_B)\ \wedge\ M(z_B, x'_F)\ \wedge\ M(x'_F, y'_W))$
6) $\forall x_G, y_F, z_W\ (\neg E(z_W, y_F)\ \wedge\ \neg E(z_W, x_G))$
7) $\forall x_B, y_C, z_S\ (E(x_B, y_C)\ \wedge\ \neg E(x_B, z_S))$
8) $\forall x_C\ \exists y_P\ E(x_C, y_P)\ \wedge\ \forall x_S\ \exists y_P\ E(x_S, y_P)$
9) $\exists x_A, y_A, z_G\ (E(x_A, y_A)\ \wedge\ E(y_A, z_G))$

In [Sti86] different formalizations of the theorem are discussed. In order to demonstrate our CNF-Algorithm (see Chapter 4, Example 4.2.3) we will use the theorem formula $\exists x_A\ \exists y_A\ (E(x_A, y_A)\ \wedge\ \forall x_G\ E(y_A, x_G))$. Choosing this representation of the theorem results in a more complicated resolution proof.

Example 3.3.2 (The Lion and the Unicorn) The second example is a puzzle named "The Lion and the Unicorn", that can be found in [Smu78]. It reads as follows:

> When Alice entered the forest of forgetfulness, she did not forget everything, only certain things. She often forgot her name, and the most likely thing for her to forget was the day of the week. Now, the lion and the unicorn were frequent visitors to this forest. These two are strange creatures. The lion lies on Mondays, Tuesdays and Wednesdays and tells the truth on the other days of the week. The unicorn, on the other hand, lies on Thursdays, Fridays and Saturdays, but tells the truth on the other days of the week.
> One day Alice met the lion and the unicorn resting under a tree. They made the following statements:

Lion:	Yesterday was one of my lying days.
Unicorn:	Yesterday was one of my lying days.

> From this statements, Alice, who was a bright girl, was able to deduce the day of the week. What was it ?

In our formalization of the problem we follow mainly the suggestions in [OSS85]. Thus our signature consists of the sorts *Mo*, *Tu*, *We*, *Th*, *Fr*, *Sa* and *Su* which stand for the days of the week, the sorts LL and LU which denote the lying days of the lion and the unicorn, respectively, a sort D for all days and a sort C for the two creatures. The constants *mon*, *tue*, *wed*, *thu*, *fri*, *sat* and *sun*, the constants *lion* and *uni* and a 1-place function f are needed. Furthermore we need a 3-place predicate $Lies(x_C, y_D, z_D)$ which is true if the creature x_C says at day y_D that it lies at day z_D. Now we can give a complete set of formulas for this problem:

$$
\begin{array}{rllllll}
1) & mon \in Mo & \wedge & tue \in Tu & \wedge & wed \in We & \wedge \\
 & thu \in Th & \wedge & fri \in Fr & \wedge & sat \in Sa & \wedge \\
 & sun \in Su & & & & & \\
2) & \forall x_{Mo}\ x_{Mo} \in LL & \wedge & \forall x_{Tu}\ x_{Tu} \in LL & \wedge & \forall x_{We}\ x_{We} \in LL & \wedge \\
 & \forall x_{Th}\ x_{Th} \notin LL & \wedge & \forall x_{Fr}\ x_{Fr} \notin LL & \wedge & \forall x_{Sa}\ x_{Sa} \notin LL & \wedge \\
 & \forall x_{Su}\ x_{Su} \notin LL & & & & & \\
3) & \forall x_{Mo}\ x_{Mo} \notin LU & \wedge & \forall x_{Tu}\ x_{Tu} \notin LU & \wedge & \forall x_{We}\ x_{We} \notin LU & \wedge \\
 & \forall x_{Th}\ x_{Th} \in LU & \wedge & \forall x_{Fr}\ x_{Fr} \in LU & \wedge & \forall x_{Sa}\ x_{Sa} \in LU & \wedge \\
 & \forall x_{Su}\ x_{Su} \notin LU & & & & & \\
4) & \forall x_{Mo}\ x_{Mo} \in D & \wedge & \forall x_{Tu}\ x_{Tu} \in D & \wedge & \forall x_{We}\ x_{We} \in D & \wedge \\
 & \forall x_{Th}\ x_{Th} \in D & \wedge & \forall x_{Fr}\ x_{Fr} \in D & \wedge & \forall x_{Sa}\ x_{Sa} \in D & \wedge \\
 & \forall x_{Su}\ x_{Su} \in D & & & & & \\
5) & \forall x_{LL}\ x_{LL} \in D & \wedge & \forall x_{LU}\ x_{LU} \in D & & & \\
6) & lion \in C & \wedge & uni \in C & & & \\
7) & \forall x_{Mo}\ f(x_{Mo}) \in Su & \wedge & \forall x_{Tu}\ f(x_{Tu}) \in Mo & \wedge & \forall x_{We}\ f(x_{We}) \in Tu & \wedge \\
 & \forall x_{Th}\ f(x_{Th}) \in We & \wedge & \forall x_{Fr}\ f(x_{Fr}) \in Th & \wedge & \forall x_{Sa}\ f(x_{Sa}) \in Fr & \wedge \\
 & \forall x_{Su}\ f(x_{Su}) \in Sa & & & & & \\
8) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \notin LL\ \wedge\ Lies(lion, x_D, y_D) \Rightarrow y_D \in LL)} \\
9) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \notin LL\ \wedge\ \neg Lies(lion, x_D, y_D) \Rightarrow y_D \notin LL)} \\
10) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \in LL\ \wedge\ Lies(lion, x_D, y_D) \Rightarrow y_D \notin LL)} \\
11) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \in LL\ \wedge\ \neg Lies(lion, x_D, y_D) \Rightarrow y_D \in LL)} \\
12) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \notin LU\ \wedge\ Lies(uni, x_D, y_D) \Rightarrow y_D \in LU)} \\
13) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \notin LU\ \wedge\ \neg Lies(uni, x_D, y_D) \Rightarrow y_D \notin LU)} \\
14) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \in LU\ \wedge\ Lies(uni, x_D, y_D) \Rightarrow y_D \notin LU)} \\
15) & \multicolumn{6}{l}{\forall x_D\ \forall y_D\ (x_D \in LU\ \wedge\ \neg Lies(uni, x_D, y_D) \Rightarrow y_D \in LU)} \\
16) & \multicolumn{6}{l}{\exists x_D\ (Lies(lion, x_D, f(x_D))\ \wedge\ Lies(uni, x_D, f(x_D)))}
\end{array}
$$

Comparing our formalization given so far with the formalization in [OSS85], we have replaced the *MEMBER* predicate by the $\in$ relation and we have tried to express as much as possible in terms of sort.

Example 3.3.3 (Condensed Detachment) Condensed detachment is an inference rule that combines modus ponens and instantiation [Mer77]. The general technique is to code various problems into the term structure of first-order logic. Thus we only need one predicate symbol P expressing "is a theorem". Solving examples of this form with an automated theorem prover can be arbitrarily hard [MW92]. The example shown here is from the two-valued sentential calculus, where the 2-place function i denotes implication and the 1-place function n negation. We give an unsorted formalization:

1) $\forall x, y\ (P(i(x,y)) \ \wedge\ P(x) \Rightarrow P(y))$
2) $\forall x, y, z\ P(i(i(x,y), i(i(y,z), i(x,z))))$
3) $\forall x\ P(i(i(n(x),x),x))$
4) $\forall x, y\ P(i(x, i(n(x), y)))$
5) $\forall x\ P(i(x,x))$

We will prove that (1 $\wedge$ 2 $\wedge$ 3 $\wedge$ 4) $\Rightarrow$ 5.

3.4 Relativization

In this subsection we show how to transform formulas of $\mathcal{L}_{\mathcal{S}}$ into unsorted first-order logic. This is useful for two reasons. Firstly, the aim of our approach is not to develop a more expressive logic than unsorted first-order logic, but a sorted first-order logic which can deal with taxonomic information in a more efficient way. Secondly, if we can prove that the semantic and syntactic notations of both logics are equivalent well known results from unsorted first-order logic can be taken over. Therefore, we prove the model theoretic part of the sort theorem which states the equivalence of the logics.

The transformation of the formulas will be done in two steps. The first step of the transformation eliminates variables which have not sort $\top$. Then we use the standard method [Wal87] to provide a unary predicate for each sort symbol and to add the formula $\forall x\ \top(x)$ to axiomatize the top sort.

Lemma 3.4.1 (Simple Properties) Let $\mathcal{F}$ be a formula, $\Im$ be an interpretation and S a sort symbol, then:

- $\Im \models \forall x_S\ \mathcal{F}$ iff $\Im \models \forall x_\top\ (x_\top \lessdot S \Rightarrow \{x_S \mapsto x_\top\}\mathcal{F})$ $\quad x_\top \notin FV(\mathcal{F})$
- $\Im \models \exists x_S\ \mathcal{F}$ iff $\Im \models \exists x_\top\ (x_\top \lessdot S \ \wedge\ \{x_S \mapsto x_\top\}\mathcal{F})$ $\quad x_\top \notin FV(\mathcal{F})$

Proof: Can easily be seen by exploiting the semantics. □

Algorithm 3.4.2 (Ψ: Eliminating variables with sorts different from $\top$) We will now present an algorithm Ψ, which uses the equivalences of Lemma 3.4.1 to eliminate variables with sorts different from $\top$. We present our algorithms in terms of nondeterministic rules, because this gives a better understanding of their properties and the fundamental ideas.

- $\Psi(\forall x_S\ \mathcal{F})$ $\quad\rightarrow\ \forall x_\top\ (x_\top \lessdot S \Rightarrow \{x_S \mapsto x_\top\}\Psi(\mathcal{F}))\qquad x_\top \notin FV(\mathcal{F})$
- $\Psi(\exists x_S\ \mathcal{F})$ $\quad\rightarrow\ \exists x_\top\ (x_\top \lessdot S\ \wedge\ \{x_S \mapsto x_\top\}\Psi(\mathcal{F}))\qquad x_\top \notin FV(\mathcal{F})$
- $\Psi(\neg\mathcal{F})$ $\quad\rightarrow\ \neg\Psi(\mathcal{F})$
- $\Psi(\mathcal{F} \star \mathcal{G})$ $\quad\rightarrow\ \Psi(\mathcal{F}) \star \Psi(\mathcal{G})$ with $\quad \star \in \{\Leftrightarrow, \Rightarrow, \wedge, \vee\}$
- $\Psi(P(t_1, \ldots, t_n))$ $\quad\rightarrow\ P(t_1, \ldots, t_n)$
- $\Psi(t \lessdot S)$ $\quad\rightarrow\ t \lessdot S$

Using Lemma 3.4.1 we find that $\Psi(\mathcal{F})$ is logically equivalent to $\mathcal{F}$ for every formula $\mathcal{F}$. Obviously, Ψ terminates and $\Psi(\mathcal{F})$ contains only variables of sort $\top$.

Before we show the transformation of our logic into first-order logic we have to give a brief introduction. Mainly the syntax and semantics of the logics are the same. The only differences are the sort symbols and the special relation $\lessdot$. Therefore we use similar syntactic and semantic notions by adding $'$ to predicate and function symbols of our logic. The sort symbols attached to variables are omitted. All this leads to the corresponding symbols of unsorted first-order logic.

Definition 3.4.3 (Unsorted First-Order Logic) The *first-order signature* $\mathbf{\Sigma}'$ is given by $\mathbf{\Sigma}' := (\mathbf{V}', \mathbf{F}', \mathbf{P}')$. Terms, atoms, literals and formulas are built in the usual way. An *interpretation* $\Im' = (\mathcal{M}', \varphi')$ consists of a $\mathbf{\Sigma}'$*-structure* $\mathcal{M}'$ and a total $\mathbf{\Sigma}'$*-assignment* $\varphi' \colon \mathbf{V}' \rightarrow \mathbf{U}$ which can be naturally extended to terms. The $\mathbf{\Sigma}'$-structure $\mathcal{M}'$ consists of a non empty universe $\mathbf{U}$, a total function $f'_{\mathbf{U}} \colon \mathbf{U}^n \rightarrow \mathbf{U}$ for each n-place function symbol $f' \in \mathbf{F}'_n$ and a n-place relation $P'_{\mathcal{M}'} \subseteq \mathbf{U}^n$ for each predicate symbol $P' \in \mathbf{P}'_n$. The evaluation of terms, atoms, literals, and formulas is straightforward.

Algorithm 3.4.4 (Φ: Transformation into Unsorted First-Order Logic) Let $\mathbf{\Sigma} := (\mathbf{S}, \mathbf{V}, \mathbf{F}, \mathbf{P})$ be a sorted signature and $\mathcal{F}$ be a $\mathbf{\Sigma}$-sentence. Then the corresponding $\mathbf{\Sigma}'$-signature is given by $\mathbf{\Sigma}' := (\mathbf{V}', \mathbf{F}', \mathbf{P}')$ where $\mathbf{V}'$ is a countably infinite set of variable symbols, $\mathbf{F}' := \{\, f' \mid f'$ is a new n-place function symbol for every $f \in \mathbf{F}_n \,\}$ and $\mathbf{P}' := \{\, P' \mid P'$ is a new n-place predicate symbol for every $P \in \mathbf{P}_n \,\} \cup \{\, S' \mid S'$ is a new 1-place predicate for every sort $S \in \mathbf{S} \,\}$. The algorithm Π translates $\mathbf{\Sigma}$-expressions in $\mathbf{\Sigma}'$-expressions:

- $\Pi(f(t_1, \ldots, t_n))$ $\quad\rightarrow\ f'(\Pi(t_1), \ldots, \Pi(t_n))$
- $\Pi(\mathcal{Q}x_\top\ \mathcal{F})$ $\quad\rightarrow\ \mathcal{Q}x\ \Pi(\mathcal{G})$
  where $\mathcal{Q} \in \{\forall, \exists\}$ and $\mathcal{G}$ is obtained by replacing every free occurrence of $x_\top$ in $\mathcal{F}$ by x
- $\Pi(\neg\mathcal{F})$ $\quad\rightarrow\ \neg\Pi(\mathcal{F})$
- $\Pi(\mathcal{F} \star \mathcal{G})$ $\quad\rightarrow\ \Pi(\mathcal{F}) \star \Pi(\mathcal{G})$ with $\star \in \{\Leftrightarrow, \Rightarrow, \wedge, \vee\}$
- $\Pi(P(t_1, \ldots, t_n))$ $\quad\rightarrow\ P'(t'_1, \ldots, t'_n)$

- $\Pi(t{<}S) \qquad\qquad\qquad \rightarrow\ S'(t')$

Then the corresponding set of $\mathbf{\Sigma}'$-sentences is:

$$\Phi(\mathcal{F}) = \Pi(\Psi(\mathcal{F})) \cup \{\forall x\ \top'(x)\}$$

Theorem 3.4.5 (Sort Theorem) Let $\mathcal{F}$ be a sentence of $\mathcal{L}_{\mathcal{S}}$. Then $\mathcal{F}$ has a $\mathbf{\Sigma}$-model iff $\Phi(\mathcal{F})$ has a $\mathbf{\Sigma}'$-model.
Proof: It suffices to show that $\Psi(\mathcal{F})$ has a $\mathbf{\Sigma}$-model iff $\Phi(\mathcal{F})$ has a $\mathbf{\Sigma}'$-model, because $\Psi(\mathcal{F})$ is equivalent to $\mathcal{F}$ according to Lemma 3.4.1

"$\Rightarrow$" Let $\Im = (\mathcal{M}, \varphi)$ be a $\mathbf{\Sigma}$-model for $\Psi(\mathcal{F})$. Then we construct a $\mathbf{\Sigma}'$-model $\Im'$ as follows: $\mathbf{U} := \mathbf{A}$, functions and predicates are interpreted in $\mathcal{M}'$ as they are interpreted in $\mathcal{M}$ and the added unary predicates S_i' for the sort symbols are given by $(S_i')_{\mathcal{M}'} := (S_i)_{\mathcal{A}}$. Especially $(\top')_{\mathcal{M}'} := \mathbf{A}$. Obviously $\mathbf{U}$ is not empty and $\Im' \models \forall x\ \top'(x)$. Thus we only have to show by induction on the structure of formulas, that $\Im \models \Psi(\mathcal{F})$ implies $\Im' \models \Pi(\Psi(\mathcal{F}))$. We prove the non trivial parts only.

- $\Psi(\mathcal{F}) = P(t_1, \ldots, t_n)$ for a n-place predicate symbol $P{\in}\mathbf{P}_n$, then $\Im \models P(t_1, \ldots, t_n)$ implies $\Im' \models P'(t_1', \ldots, t_n')$, because $\Im(P) = \Im'(P')$ and $\Im(t_i) = \Im'(t_i')$ for every i by construction and by defining $\varphi'(x) := \varphi(x)$ for all $x{\in}V(P(t_1, \ldots, t_n))$.
- $\Psi(\mathcal{F}) = \forall x_\top\ (x_\top{<}S \Rightarrow \mathcal{G})$, then $\Im \models \forall x_\top\ (x_\top{<}S \Rightarrow \mathcal{G})$ implies $\Im' \models \forall x\ (S'(x) \Rightarrow \mathcal{G}')$, because $\mathbf{U} = \top_{\mathcal{A}}$ and we have $(\mathcal{M}, \varphi[\{x_\top/a\}]) \models x_\top{<}S \Rightarrow \mathcal{G}$ for all $a{\in}\top_{\mathcal{A}}$ implies $(\mathcal{M}', \varphi'[\{x/a\}]) \models S'(x) \Rightarrow \mathcal{G}'$ by induction hypothesis. The case $\Psi(\mathcal{F}) = \exists x_\top\ (x_\top{<}S\ \wedge\ \mathcal{G})$ is similar to this one.

"$\Leftarrow$" Let $\Im' = (\mathcal{M}', \varphi')$ be a $\mathbf{\Sigma}'$-model for $\Phi(\mathcal{F})$. Then we construct a $\mathbf{\Sigma}$-model $\Im$ as follows: the $\mathbf{\Sigma}$-algebra $\mathcal{A}$ is defined with carrier $\mathbf{A} = \top'_{\mathcal{M}'} = \mathbf{U}$. $\mathbf{A}$ is not empty because $\mathbf{U}$ is not empty. For every n-place function symbol $f{\in}\mathbf{F}_n$, $(u_1, \ldots, u_n){\in}\mathbf{A}^n$ we define $f_{\mathcal{A}}(u_1, \ldots, u_n) = f'_{\mathcal{M}'}(u_1, \ldots, u_n)$. For every sort symbol $S{\in}\mathbf{S}$ the set $S'_{\mathcal{M}'}$ is assigned. $S_{\mathcal{A}} \subseteq \mathbf{A}$ because of $\Im' \models \forall x\ \top'(x)$. For every n-place predicate symbol $P{\in}\mathbf{P}_n$ we assign the relation $P_{\mathcal{M}} := P'_{\mathcal{M}'}$. Finally we show by induction on the structure of formulas, that $\Im' \models \Pi(\Psi(\mathcal{F}))$ implies $\Im \models \Psi(\mathcal{F})$.

- $\Pi(\Psi(\mathcal{F})) = P'(t_1', \ldots, t_n')$ for a n-place predicate symbol $P'{\in}\mathbf{P}_n'$, then $\Im' \models P'(t_1', \ldots, t_n')$ implies $\Im \models P(t_1, \ldots, t_n)$, because the fact that $\Pi(\Psi(\mathcal{F}))$ is a sentence of a special form ensures that $\varphi'(x){\in}(\mathit{sort}(x))_{\mathcal{A}}$ for all $x{\in}V(P'(t_1', \ldots, t_n'))$
- $\Pi(\Psi(\mathcal{F})) = S'(t')$ then $\Im' \models S'(t')$ implies $\Im \models t{<}S$ by construction and the above argumentation for the interpretation of t'.
- $\Pi(\Psi(\mathcal{F})) = \forall x\ (S'(x) \Rightarrow \mathcal{G}')$ then $\Im' \models \forall x\ (S'(x) \Rightarrow \mathcal{G}')$ implies $\Im \models \forall x_\top\ (x_\top{<}S \Rightarrow \mathcal{G})$ because $a{\in}S'_{\mathcal{M}'}$ implies $a{\in}S_{\mathcal{A}}$ and the induction hypothesis for $\mathcal{G}'$ and $S'(x)$ holds. The case $\Pi(\Psi(\mathcal{F})) = \exists x\ (S'(x)\ \wedge\ \mathcal{G}')$ is similar to this case.

□

Theorem 3.4.6 (Compactness Theorem) A set F of $\mathcal{L}_{\mathcal{S}}$ sentences is satisfiable iff every finite subset F' of F is satisfiable.

Proof: Follows from Theorem 3.4.5 and the fact that the Compactness Theorem holds for first-order logic. □

Chapter 4

Clause Normal Form

4.1 Introduction

Resolution based deduction systems usually require a certain normal form of formulas, called “clause normal form”. In clause normal form negation symbols do only occur in front of predicate symbols, all variables are universally quantified and the whole formula is a conjunction of disjunctions, called clauses. In most sorted logics, the process of transforming a formula into clause normal form is very similar to the unsorted case, because they do not allow for empty sorts and declarations inside the formula. In $\mathcal{L}_S$, this is all possible. Therefore, transformations have to be realized more carefully.

The main differences between the standard algorithms [CL73, Lov78, SS89] and our algorithm are as follows. Firstly, sets may be empty and therefore some standard transformation rules don't hold, e.g. the rule for extending the scope of a universal quantifier. $\forall x_S\ \mathcal{F} \ \wedge\ \mathcal{G}$ is not equivalent to $\forall x_S\ (\mathcal{F} \ \wedge\ \mathcal{G})$, where x_S occurs free in $\mathcal{F}$. Consider an interpretation $\Im$ where $S_{\mathcal{A}} = \emptyset$ and $\Im \not\models \mathcal{G}$. Then $\Im \models \forall x_S\ (\mathcal{F} \ \wedge\ \mathcal{G})$ but $\Im \not\models \forall x_S\ \mathcal{F} \ \wedge\ \mathcal{G}$. Secondly, Skolemization doesn't work in the usual way, because information about the domain and range sorts of a Skolem function has to be locally provided. In other sorted logics [SS89] the sort information of a Skolem function can be globally declared, but the example below shows that this does lead to an incorrect transformation in our logic. The formula $(\exists x_S\ P(x_S) \ \vee\ Q(a)) \ \wedge\ \forall x_S\ x_S \not\lessdot S$ has a model, but the naive global skolemization $(P(b) \ \vee\ Q(a)) \ \wedge\ \forall x_S\ x_S \not\lessdot S \ \wedge\ b \lessdot S$ has no model, whereas the local Skolemization $((P(b) \ \wedge\ b \lessdot S) \ \vee\ Q(a)) \ \wedge\ \forall x_S\ x_S \not\lessdot S$ preserves satisfiability. The necessity for doing local Skolemization results from the combination of allowing empty sorts on the semantic level and negative sort information on the syntactic one.

4.2 The CNF Algorithm

Definition 4.2.1 (Normal Forms) A formula $\mathcal{F}$ is said to be in *negation normal* *form*, if every negation symbol occurring in $\mathcal{F}$ stands directly in front of a predicate symbol. $\mathcal{F}$ is in *clause normal form*, if $\mathcal{F}$ is in negation normal form and $\mathcal{F} = \mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n$, where the $\mathcal{F}_i = \forall y_{S_1}, \ldots, y_{S_n} \ (L_1 \vee \ldots \vee L_n)$, the L_i are literals and $V(\{L_1, \ldots, L_n\}) = \{y_{S_1}, \ldots, y_{S_n}\}$

Algorithm 4.2.2 (Δ: Transformation in Clause Normal Form) The input of the algorithm Δ is a sorted logic sentence $\mathcal{F}$, which is transformed into a set of clauses $\Delta(\mathcal{F})$. Δ includes seven steps which are applied incrementally. A step is finished if none of its rules is applicable.

Step1: Use the rules

- $\mathcal{F} \Leftrightarrow \mathcal{G} \rightarrow (\mathcal{F} \Rightarrow \mathcal{G}) \wedge (\mathcal{G} \Rightarrow \mathcal{F})$
- $\mathcal{F} \Rightarrow \mathcal{G} \rightarrow \neg\mathcal{F} \vee \mathcal{G}$

to eliminate implications and equivalences. Note that the two rules establish a naive equivalence elimination, because formulas may grow exponentially in size and are highly redundant after performing the elimination. Here standard techniques as described in [Soc90, Soc91] can be applied.

Step2: Use the rules

- $\neg\forall x_S \ \mathcal{F} \rightarrow \exists x_S \ \neg\mathcal{F}$
- $\neg\exists x_S \ \mathcal{F} \rightarrow \forall x_S \ \neg\mathcal{F}$
- $\neg(\mathcal{F} \wedge \mathcal{G}) \rightarrow \neg\mathcal{F} \vee \neg\mathcal{G}$
- $\neg(\mathcal{F} \vee \mathcal{G}) \rightarrow \neg\mathcal{F} \wedge \neg\mathcal{G}$
- $\neg(\neg\mathcal{F}) \rightarrow \mathcal{F}$

to move the negation sign inwards.

Step3: Use the rules

- $\forall x_S \ (\mathcal{F} \vee \mathcal{G}) \rightarrow \forall x_S \ \mathcal{F} \vee \mathcal{G}$
  $x_S \notin FV(\mathcal{G})$
- $\exists x_S \ (\mathcal{F} \wedge \mathcal{G}) \rightarrow \exists x_S \ \mathcal{F} \wedge \mathcal{G}$
  $x_S \notin FV(\mathcal{G})$
- $\forall x_S \ (\mathcal{F} \wedge \mathcal{G}) \rightarrow \forall x_S \ \mathcal{F} \wedge \forall x_S \ \mathcal{G}$
  $x_S \in FV(\mathcal{F})$ and $x_S \in FV(\mathcal{G})$
- $\exists x_S \ (\mathcal{F} \vee \mathcal{G}) \rightarrow \exists x_S \ \mathcal{F} \vee \exists x_S \ \mathcal{G}$
  $x_S \in FV(\mathcal{F})$ and $x_S \in FV(\mathcal{G})$
- $\forall x_S \ \forall y_T \ (\mathcal{F} \vee \mathcal{G}) \rightarrow \forall y_T \ \forall x_S \ (\mathcal{F} \vee \mathcal{G})$
  $y_T \in FV(\mathcal{F})$ and $y_T \in FV(\mathcal{G})$, but either $x_S \notin FV(\mathcal{F})$ or $x_S \notin FV(\mathcal{G})$

- $\exists x_S \, \exists y_T \, (\mathcal{F} \ \wedge \ \mathcal{G}) \rightarrow \exists y_T \, \exists x_S \, (\mathcal{F} \ \wedge \ \mathcal{G})$
  $y_T{\in}FV(\mathcal{F})$ and $y_T{\in}FV(\mathcal{G})$, but either $x_S{\notin}FV(\mathcal{F})$ or $x_S{\notin}FV(\mathcal{G})$

to move the quantifiers inwards. These rules are not necessary for Skolemization, but are useful to get small Skolem functions, which increase efficiency of the refutation process.

Step4: Rename all variables, such that different quantifiers have different variables.

Step5: From left to right remove all existential quantifiers. Let $\mathcal{F} = \exists x_S \, \mathcal{G}$ be the formula of the current existential quantifier. Let $\forall y_{S_1} \ldots \forall y_{S_n}$ be the universal quantifiers which have $\mathcal{F}$ in their scope. Then replace $\mathcal{F}$ by $\Big(\{x_S {\mapsto} f(y_{S_1}, \ldots, y_{S_n})\}\mathcal{G} \ \wedge \ f(y_{S_1}, \ldots, y_{S_n}){<}S\Big)$ where f is a new n-place Skolem function. If we have $S = \top$, we can skip the declaration $f(y_{S_1}, \ldots, y_{S_n}){<}S$.

Step6: Use the rules

- $\mathcal{H} \ \vee \ (\mathcal{F} \ \wedge \ \mathcal{G}) \rightarrow (\mathcal{H} \ \vee \ \mathcal{F}) \ \wedge \ (\mathcal{H} \ \vee \ \mathcal{G})$
- $\forall x_S \, (\mathcal{F} \ \wedge \ \mathcal{G}) \rightarrow \forall x_S \, \mathcal{F} \ \wedge \ \forall x_S \, \mathcal{G}$
- $(\forall x_S \, \mathcal{F}) \ \vee \ \mathcal{G} \rightarrow \forall x_S \, (\mathcal{F} \ \vee \ \mathcal{G})$

and variable renaming to transform the formula into the form $\mathcal{F} = \mathcal{F}_1 \ \wedge \ \ldots \ \wedge \ \mathcal{F}_n$, where the $\mathcal{F}_i = \forall x_{S_1} \ldots \forall x_{S_n} \, L_1 \ \vee \ \ldots \ \vee \ L_m$ and the L_i are literals. Again the rule may grow the formula exponentially in size. The growing can be prevented by introducing new predicates regarded as abbreviations for subformulas.

Step7: Use the rule

- $\forall x_{S_1} \ldots \forall x_{S_n} \, (L_1 \ \vee \ \ldots \ \vee \ L_m) \rightarrow \forall x_{S_1} \ldots \forall x_{S_{i-1}} \, \forall x_\top \, \forall x_{S_{i+1}} \, \forall x_{S_n} \, (L_1 \ \vee \ \ldots \ \vee \ L_m \ \vee \ x_\top {\not<} S_i)$
  where $x_{S_i}{\notin}FV(L_1 \ \vee \ \ldots \ \vee \ L_m)$ and $x_\top$ is a new variable

to ensure that a variable occurs in a disjunction $L_1 \ \vee \ \ldots \ \vee \ L_m$ iff it occurs in the universal quantifiers in front of the disjunction. After having applied the above rule, we reach the clause normal form of the input formula.

The rule can be further improved. We only need to add the additional literals of the form $x_\top {\not<} S$ for variables x_S, if no different variable of sort S is bound by the universal quantifier. For example the formula $\forall x_S, y_S \, P(x_S)$ can be transformed into $\forall x_S \, P(x_S)$.

Now we can drop the quantifiers and switch to the clause oriented notation by transforming all disjunctions into clauses and the whole conjunction in a set of clauses.

Note that dropping the quantifiers and assuming all variables to be universally quantified is not a correct operation if the disjunctions of *Step6* contain free variables or there are variables which do not occur in the literals of a disjunction but in the universal quantifiers. For example the formula $\forall x_S\ P(a)\ \wedge\ \neg P(a)$ has a model, but $P(a)\ \wedge\ \neg P(a)$ is unsatisfiable. Also $\forall x_S\ (P(x_S)\ \vee\ Q(y_T))\ \wedge\ \neg P(a)\ \wedge\ \neg Q(b)\ \wedge\ a \lessdot S\ \wedge\ b \lessdot T$ is satisfiable, but dropping the quantifiers and assuming all variables to be universally quantified leads to an unsatisfiable formula. Therefore we demand sentences to be the input of Δ.

Example 4.2.3 (Applying Δ) The following example demonstrates the transformation of the complicated theorem formula of Example 3.3.1 in clause normal form.

$\neg(\exists x_A\ \exists y_A\ (E(x_A, y_A)\ \wedge\ \forall x_G\ E(y_A, x_G)))$

$\overset{\star}{\rightarrow}$ $\forall x_A\ \forall y_A\ (\neg E(x_A, y_A)\ \vee\ \exists x_G\ \neg E(y_A, x_G))$
using the rules of *Step2*

$\rightarrow$ $\forall y_A\ \forall x_A\ (\neg E(x_A, y_A)\ \vee\ \exists x_G\ \neg E(y_A, x_G))$
using *Step3*, fifth rule

$\rightarrow$ $\forall y_A\ (\forall x_A\ \neg E(x_A, y_A)\ \vee\ \exists x_G\ \neg E(y_A, x_G))$
using *Step3*, first rule

$\rightarrow$ $\forall y_A\ (\forall x_A\ \neg E(x_A, y_A)\ \vee\ (\neg E(y_A, f(y_A))\ \wedge\ f(y_A) \lessdot G))$
using *Step5* where f is a new 1-place Skolem function. Without having applied the rules of *Step3* we would have got a 2-place Skolem function

$\rightarrow$ $\forall y_A\ ((\forall x_A\ \neg E(x_A, y_A)\ \vee\ \neg E(y_A, f(y_A)))\ \wedge\ (\forall z_A\ \neg E(z_A, y_A)\ \vee\ f(y_A) \lessdot G))$
using *Step6*, first rule and variable renaming

$\rightarrow$ $\forall y_A\ (\forall x_A\ \neg E(x_A, y_A) \vee \neg E(y_A, f(y_A))) \wedge \forall u_A\ (\forall z_A\ \neg E(z_A, u_A) \vee f(u_A) \lessdot G)$
using *Step6*, second rule and variable renaming

$\overset{\star}{\rightarrow}$ $\forall y_A\ \forall x_A\ (\neg E(x_A, y_A) \vee \neg E(y_A, f(y_A))) \wedge \forall u_A\ \forall z_A\ (\neg E(z_A, u_A) \vee f(u_A) \lessdot G)$
using *Step6*, third rule

$\rightarrow$ $\{\{\neg E(x_A, y_A), \neg E(y_A, f(y_A))\}, \{\neg E(z_A, u_A), f(u_A) \lessdot G\}\}$
using *Step7*

Example 4.2.4 (Schubert's Steamroller) We present the set of clauses obtained from applying the algorithm to the formulas presented in Example 3.3.1. The theorem was negated.

(1)	$\{x_W \lessdot A\}$	(2)	$\{x_F \lessdot A\}$	(3)	$\{x_B \lessdot A\}$
(4)	$\{x_C \lessdot A\}$	(5)	$\{x_S \lessdot A\}$	(6)	$\{lupo \lessdot W\}$
(7)	$\{foxy \lessdot F\}$	(8)	$\{tweety \lessdot B\}$	(9)	$\{raupy \lessdot C\}$
(10)	$\{schnecky \lessdot S\}$	(11)	$\{muesli \lessdot G\}$	(12)	$\{x_G \lessdot P\}$
(13)	$\{E(x_A, y_P), \neg M(z_A, x_A), \neg E(z_A, v_P), E(x_A, z_A)\}$				
(14)	$\{M(x_C, z_B)\}$	(15)	$\{M(y_S, z_B)\}$	(16)	$\{M(z_B, x_F)\}$
(17)	$\{M(x_F, y_W)\}$	(18)	$\{\neg E(z_W, y_F)\}$	(19)	$\{\neg E(z_W, x_G)\}$
(20)	$\{E(x_B, y_C)\}$	(21)	$\{\neg E(x_B, z_S)\}$	(22)	$\{E(x_C, f(x_C))\}$
(23)	$\{f(x_C) \lessdot P\}$	(24)	$\{E(x_S, g(x_S))\}$	(25)	$\{g(x_S) \lessdot P\}$
(26)	$\{\neg E(x_A, y_A), \neg E(y_A, z_G)\}$				

Example 4.2.5 (The Lion and the Unicorn) We present the set of clauses obtained by applying the algorithm to the formulas presented in Example 3.3.2. The theorem was negated.

(1)	$\{mon \in Mo\}$	(2)	$\{tue \in Tu\}$	(3)	$\{wed \in We\}$
(4)	$\{thu \in Th\}$	(5)	$\{fri \in Fr\}$	(6)	$\{sat \in Sa\}$
(7)	$\{sun \in Su\}$	(8)	$\{x_{Mo} \in LL\}$	(9)	$\{x_{Tu} \in LL\}$
(10)	$\{x_{We} \in LL\}$	(11)	$\{x_{Th} \notin LL\}$	(12)	$\{x_{Fr} \notin LL\}$
(13)	$\{x_{Sa} \notin LL\}$	(14)	$\{x_{Su} \notin LL\}$	(15)	$\{x_{Th} \in LU\}$
(16)	$\{x_{Fr} \in LU\}$	(17)	$\{x_{Sa} \in LU\}$	(18)	$\{x_{Su} \notin LU\}$
(19)	$\{x_{Mo} \notin LU\}$	(20)	$\{x_{Tu} \notin LU\}$	(21)	$\{x_{We} \notin LU\}$
(22)	$\{x_{Mo} \in D\}$	(23)	$\{x_{Tu} \in D\}$	(24)	$\{x_{We} \in D\}$
(25)	$\{x_{Th} \in D\}$	(26)	$\{x_{Fr} \in D\}$	(27)	$\{x_{Sa} \in D\}$
(28)	$\{x_{Su} \in D\}$	(29)	$\{x_{LL} \in D\}$	(30)	$\{x_{LU} \in D\}$
(31)	$\{lion \in C\}$	(32)	$\{uni \in C\}$	(33)	$\{f(x_{Mo}) \in Su\}$
(34)	$\{f(x_{Tu}) \in Mo\}$	(35)	$\{f(x_{We}) \in Tu\}$	(36)	$\{f(x_{Th}) \in We\}$
(37)	$\{f(x_{Fr}) \in Th\}$	(38)	$\{f(x_{Sa}) \in Fr\}$	(39)	$\{f(x_{Su}) \in Sa\}$

(40)	$\{x_D \in LL, \neg Lies(lion, x_D, y_D), y_D \in LL\}$
(41)	$\{x_D \in LL, Lies(lion, x_D, y_D), y_D \notin LL\}$
(42)	$\{x_D \notin LL, \neg Lies(lion, x_D, y_D), y_D \notin LL\}$
(43)	$\{x_D \notin LL, Lies(lion, x_D, y_D), y_D \in LL\}$
(44)	$\{x_D \in LU, \neg Lies(uni, x_D, y_D), y_D \in LU\}$
(45)	$\{x_D \in LU, Lies(uni, x_D, y_D), y_D \notin LU\}$
(46)	$\{x_D \notin LU, \neg Lies(uni, x_D, y_D), y_D \notin LU\}$
(47)	$\{x_D \notin LU, Lies(uni, x_D, y_D), y_D \in LU\}$
(48)	$\{\neg Lies(lion, x_D, f(x_D)), \neg Lies(uni, x_D, f(x_D))\}$

Example 4.2.6 (Condensed Detachment) We present the set of clauses obtained by applying the algorithm to the formulas presented in Example 3.3.3. The theorem is negated.

(1)	$\{\neg P(i(x, y)), \neg P(x), P(y)\}$
(2)	$\{P(i(i(x, y), i(i(y, z), i(x, z))))\}$
(3)	$\{P(i(i(n(x), x), x))\}$
(4)	$\{P(i(x, i(n(x), y)))\}$
(5)	$\{\neg P(i(a, a))\}$

We can use the CNF-algorithm in order to transform unsorted formulas also, if the top sort $\top$ is attached to all variables of the problem.

4.3 Properties of Δ

We prove that the usual properties of CNF-algorithms also hold for Δ. The termination of the algorithm can be shown by assigning an appropriate measure to every step, which decreases with every rule application. We can easily show by contradiction that $\Delta(\mathcal{F})$ is in clause normal form for every sorted logic sentence $\mathcal{F}$. Now we focus on the remaining logical properties of Δ.

Lemma 4.3.1 (Correctness of Replacing Positive Components) If $\mathcal{F}$ is a positive component of a formula $\mathcal{G}$ and $\mathcal{G}'$ is obtained from $\mathcal{G}$ by replacing $\mathcal{F}$ in $\mathcal{G}$ by a formula $\mathcal{F}'$ and if there is a model $\Im$ for the set of formulas $\{\mathcal{G}, \mathcal{F} \Rightarrow \mathcal{F}'\}$, then $\mathcal{G}'$ has model $\Im$.

Proof: The proof is a direct extension of the proof in [Lov78]. □

Theorem 4.3.2 (Soundness and Completeness of Δ) Let $\mathcal{F}$ be a sentence. Then $\mathcal{F}$ has a model $\Im$ iff $\Delta(\mathcal{F})$ has a model $\Im'$.

Proof: Except for *Step5*, both parts of each conversion rule are equivalent, so for these rules there is nothing to show. It remains to consider the elimination of the existential quantifiers. Wlog. we can assume that there is only one Skolemization step. Let $\mathcal{G} = \exists x_S\ \mathcal{H}$ be the formula with the existential quantifier being eliminated, $\forall y_{S_1} \ldots \forall y_{S_n}$ be the universal quantifiers which have $\mathcal{G}$ in their scope and $\mathcal{G}' = \{x_S \mapsto f(y_{S_1}, \ldots, y_{S_n})\}\mathcal{H} \ \wedge \ f(y_{S_1}, \ldots, y_{S_n}) \lessdot S$ be the formula obtained from $\mathcal{G}$ after elimination of the existential quantifier.

"$\Rightarrow$" Let $\Im = (\mathcal{M}, \varphi)$ be a model for $\mathcal{F}$, $(S_i)_\mathcal{A} \neq \emptyset$ for every i and $(\mathcal{M}, \varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]) \models \exists x_S\ \mathcal{H}$ for an arbitrary $\mathbf{\Sigma}$-assignment $\varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]$ with $a_i \in (S_i)_\mathcal{A}$ for all i. We will now extend $\Im$ to a model $\Im'$ which satisfies $\Delta(\mathcal{F})$. If $\exists x_S\ \mathcal{H}$ holds, there must be an $a \in S_\mathcal{A}$, such that $(\mathcal{M}, \varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n, x_S/a\}]) \models \mathcal{H}$. Now define $\Im'$ like $\Im$ except that $f_\mathcal{A}(a_1, \ldots, a_n) = a$. Thus we have $\Im' \models \{\mathcal{F}, \mathcal{G} \Rightarrow \mathcal{G}'\}$ and together with Lemma 4.3.1 we conclude $\Im' \models \Delta(\mathcal{F})$.

"$\Leftarrow$" Let $\Im' = (\mathcal{M}', \varphi')$ be a model for $\Delta(\mathcal{F})$, $(S_i)_\mathcal{A} \neq \emptyset$ for every i and $(\mathcal{M}', \varphi'[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]) \models \mathcal{G}'$ for a $\mathbf{\Sigma}$-assignment $\varphi'[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]$ with $a_i \in (S_i)_\mathcal{A}$ for all i. Then $f_\mathcal{A}(a_1, \ldots, a_n) \in S_\mathcal{A}$. Therefore $\Im' \models \exists x_S\ \mathcal{H}$ and application of Lemma 4.3.1 proves the theorem. □

Chapter 5

The Basic Resolution Calculus

In this section we will develop a sound and complete resolution calculus for sorted logic. From the computational point of view the question is how the sort information can be used to reduce the number of possible resolution steps and how to deal with taxonomic information in an efficient way. There are lots of answers to this question for standard sorted logics [SS89]. At first sight these results don't apply here, because in $\mathcal{L}_{\mathcal{S}}$ the sort information is part of the formulas itself and therefore any question concerning the structure of the sort information is undecidable and can e.g. not be attacked by efficient sorted unification algorithms. So our first answer to the question how the sort information can be exploited is a basic calculus which uses unsorted unification.

In the next chapter we will show how sorted unification can be established in this dynamic environment.

5.1 The Herbrand Theorem

The development of the calculus will be done by the usual two steps. First we investigate the ground case and then lift this result to the general case.

Before defining the ground term structure we should make a remark on the sort $\top$. The clause set $\{\{a \not\in \top\}\}$ is unsatisfiable, because every object is of sort $\top$. This implies that we should be able to refute this clause set. As the standard resolution rule does not apply to one literal we introduce the $\top$-reduction rule which allows to delete literals of the form $t \not\in \top$.

As well sortedness is not decidable in our logic, we have to change the usual instantiation rule. The idea is to add a literal of the form $t \not\in S$ for every variable x_S which is replaced by a term t. A ground instance of the clause $\{P(x_S)\}$ is $\{P(a), a \not\in S\}$ applying the ground substitution $\{x_S \mapsto a\}$. The intuitive semantics of the clause is "if a is an element of sort S, then $P(a)$ holds". From an unsorted point of view we just delay the introduction of the sort literals. The unsorted formalization of $\{P(x_S)\}$ is $\{P(x), \neg S'(x)\}$ which yields $\{P(a), \neg S'(a)\}$ using the unsorted substitution $\{x \mapsto a\}$ and the unsorted instantiation rule.

Definition 5.1.1 (Conditioned Literals) Let σ be a substitution. Then the set of *conditioned literals* $\mathit{Cond}(\sigma)$ of σ is defined as $\mathit{Cond}(\sigma) := \{\, (t \not\in T) \mid \text{there is an } x_T \in \mathit{DOM}(\sigma) \text{ with } \sigma(x_T) = t \,\}$.

Definition 5.1.2 (Conditioned Instantiation of Clauses) Let $C = \{L_1, \ldots, L_n\}$ be a clause and σ a substitution. Then $\sigma{\downarrow}C := \{\sigma(L_1), \ldots, \sigma(L_n)\} \cup \mathit{Cond}(\sigma)$ is called a *conditioned instance* of C.

The conditional literals of the form $t \not\in T$ handle the case that σ is not well sorted, i.e. in some interpretation $\Im$, $\Im(t) \notin \Im(T)$. So either for every variable $x_T \in \mathit{DOM}(\sigma)$, $\Im(\sigma(x_T)) \in \Im(T)$ or some of the added literals become true.

Lemma 5.1.3 (Soundness of Conditioned Instantiation) For every interpretation $\Im$, substitution σ and clause C, $\Im \models \forall\,(C)$ implies $\Im \models \forall\,(\sigma{\downarrow}C)$.

Proof: We prove the lemma by induction on the number of variables in $\mathit{DOM}(\sigma)$:

$|\mathit{DOM}(\sigma)| = 0$: $\sigma{\downarrow}C = C$ and therefore the lemma holds trivially.

$|\mathit{DOM}(\sigma)| = n+1$: let $x_S \in \mathit{DOM}(\sigma)$ with $\sigma(x_S) = t$. If $x_S \notin V(C)$ then $\sigma[x_S]{\downarrow} C \subseteq \sigma{\downarrow}C$ and the lemma is proven by induction hypothesis. So assume $x_S \in V(C)$, $V(\sigma{\downarrow}C) = \{y_{S_1}, \ldots, y_{S_n}\}$ and $\Im = (\mathcal{M}, \varphi)$ is a model for $\forall\,(C)$. If $S_{\mathcal{A}} = \emptyset$, then $\Im \models (t \not\in S)$ and because $(t \not\in S) \in (\sigma{\downarrow}C)$ we conclude $\Im \models \sigma{\downarrow}C$. If $S_{\mathcal{A}} \neq \emptyset$, let $\varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]$ be an arbitrary assignment for the variables of $\sigma{\downarrow}C$. If $\varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}](t) \notin S_{\mathcal{A}}$ we again have $(\mathcal{M}, \varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]) \models t \not\in S$ and thus $(\mathcal{M}, \varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}]) \models \sigma{\downarrow}C$. If $\varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}](t) \in S_{\mathcal{A}}$ with $\varphi[\{y_{S_1}/a_1, \ldots, y_{S_n}/a_n\}](t) = a$, then we conclude $\Im \models \forall\,(\sigma{\downarrow}C)$, because $\Im \models \forall\,(\sigma[x_S]{\downarrow}C)$ by induction hypothesis and $a \in S_{\mathcal{A}}$. . $\square$

Lemma 5.1.4 (Associativity of Conditioned Instantiation) Let σ, τ be two substitutions and $C = \{L_1, \ldots, L_n\}$ be a clause. If $\mathit{DOM}(\sigma) \cap \mathit{DOM}(\tau) = \emptyset$, then $\tau{\downarrow}(\sigma{\downarrow}C) = (\tau\sigma){\downarrow}C$.

Proof:

$(\tau\sigma){\downarrow}C$

$= \{\, (\tau\sigma)(L_1), \ldots, (\tau\sigma)(L_n)\} \cup \{(t \not\in T) \mid \text{there is an } x_T \in \mathit{DOM}(\tau\sigma) \text{ with } (\tau\sigma)(x_T) = t\,\}$
according Definition 5.1.2

$= \{\, (\tau\sigma)(L_1), \ldots, (\tau\sigma)(L_n)\} \cup \{\tau(t \not\in T) \mid \text{there is an } x_T \in \mathit{DOM}(\sigma) \text{ with } \sigma(x_T) = t\,\} \cup \{\, (s \not\in S) \mid \text{there is an } x_S \in (\mathit{DOM}(\tau) \setminus \mathit{DOM}(\sigma)) \text{ with } \tau(x_S) = s\,\}$
according Definition 2.2.6

$= \{(\tau\sigma)(L_1), \ldots, (\tau\sigma)(L_n)\} \cup \{\, \tau(t \not\in T) \mid \text{there is an } x_T \in \mathit{DOM}(\sigma) \text{ with } \sigma(x_T) = t\,\} \cup \{\, (s \not\in S) \mid \text{there is an } x_S \in \mathit{DOM}(\tau) \text{ with } \tau(x_S) = s\,\}$
because $\mathit{DOM}(\sigma) \cap \mathit{DOM}(\tau) = \emptyset$

$= \tau{\downarrow}\Big(\{\sigma(L_1), \ldots, \sigma(L_n)\} \cup \{\, t{\not<} T \mid \text{there is an } x_T{\in}DOM(\sigma) \text{ with } \sigma(x_T) = t\,\}\Big)$
according Definition 5.1.2

$= \tau{\downarrow}(\sigma{\downarrow}C)$

□

Definition 5.1.5 (Herbrand Set of Clauses) Let $\mathcal{F}$ be a sentence, then the *Herbrand clause set* $\Delta(\mathcal{F})_H$ is defined as follows: $\Delta(\mathcal{F})_H{:}= \{\, \sigma{\downarrow}C \mid C{\in}\Delta(\mathcal{F})$ and σ is a ground substitution with $DOM(\sigma) = V(C)\,\}$

Definition 5.1.6 (Herbrand Interpretation) A *Herbrand interpretation* $\Im_H$ is a set $\Im_H$ of ground literals satisfying the following conditions:

- There are no literals L_1 and L_2 in $\Im_H$ which are equal as atoms but L_1 is positive and L_2 is negative or vice versa
- For every ground term t, $t{<}\top$ is in $\Im_H$

A Herbrand model of a Herbrand clause set $\Delta(\mathcal{F})_H$, is a Herbrand interpretation $\Im_H$ such that for every clause $C_H{\in}\Delta(\mathcal{F})_H{:}\, C_H \cap \Im_H \neq \emptyset$. If $\Im_H$ is a Herbrand model for a Herbrand clause set CS_H, we write $\Im_H \models CS_H$ or $\Im_H \models C_H$ for a clause $C_H{\in}CS_H$.

Lemma 5.1.7 (Compatibility of $\mathbf{\Sigma}$-models and Herbrand models) Let $\mathcal{F}$ be a sentence, $CS{:}= \Delta(\mathcal{F})$ and $CS_H{:}= \Delta(\mathcal{F})_H$. Then CS has a $\mathbf{\Sigma}$-model $\Im$ iff CS_H has a Herbrand model $\Im_H$.

Proof:

"⇒" Let $\Im = (\mathcal{M}, \varphi)$ be a $\mathbf{\Sigma}$-model for CS. Then $\Im_H{:}= \{\, L \mid L$ is a ground literal and $\Im \models L\,\}$. $\Im_H$ is a Herbrand interpretation. Now we prove by contradiction that $\Im_H$ is a Herbrand model for CS_H. Suppose there is a clause $C_H{\in}CS_H$ with $C_H \cap \Im_H = \emptyset$. We know that C_H is an instantiation of a clause $C{\in}CS$. That means $C = \{L_1, \ldots, L_n\}$ and there exists a ground substitution $\sigma = \{y_{S_1} \mapsto t_1, \ldots, y_{S_n} \mapsto t_n\}$ with $DOM(\sigma) = V(C)$ and $\sigma{\downarrow}C = C_H$. If $\Im(t_i){\notin}(S_i)_\mathcal{A}$ for some i, then $\Im \models t_i{\not<} S_i$ and this contradicts $C_H \cap \Im_H = \emptyset$. So $\Im(t_i){\in}(S_i)_\mathcal{A}$ for all i. Therefore $(\mathcal{M}, \varphi[\{y_{S_1}/\Im(t_1), \ldots, y_{S_1}/\Im(t_1)\}]) {\not\models} C$. But C is universally closed over the y_{S_i}, hence this contradicts the assumption that $\Im$ is a $\mathbf{\Sigma}$-model for CS.

"⇐" Let $\Im_H$ be a Herbrand model for CS_H. We construct a $\mathbf{\Sigma}$-model as follows: The $\mathbf{\Sigma}$-algebra $\mathcal{A}$ is defined with carrier $\mathbf{A}{:}= \{\, t \mid t$ is a ground term $\}$, for every sort $S{\in}\mathbf{S}$ a set $S_\mathcal{A}{:}= \{\, t \mid (t{<}S){\in}\Im_H\,\}$ is assigned and for every $f{\in}\mathbf{F}_n$, $f_\mathcal{A}(t_1, \ldots, t_n){:}= f(t_1, \ldots, t_n)$. Note that we have $\top_\mathcal{A} = \mathbf{A}$.

Next we define a $\mathbf{\Sigma}$-structure $\mathcal{M}$ by adding $P_\mathcal{M}{:}= \{\, (t_1, \ldots, t_n) \mid P(t_1, \ldots, t_n){\in}\Im_H\,\}$ for every $P{\in}\mathbf{P}_n$.What we have defined so far is a kind of initial model for CS which is based on a ground term algebra. So from now on we can drop the differences between terms and their interpretations. To finish the proof we show by contradiction that $\Im = (\mathcal{M}, \varphi)$ is a $\mathbf{\Sigma}$-model for CS.

Assume $\Im$ is not a $\mathbf{\Sigma}$-model for CS. Then there is a clause $C{\in}CS$ with $\Im{\not\models}\ \forall\ (C)$ Let $V(C) = \{y_{S_1}, \ldots, y_{S_n}\}$ be the variables occurring in C. As $\Im$ is not a model for C, we have $(S_i)_{\mathcal{A}} \neq \emptyset$ for every i and there is a $\mathbf{\Sigma}$-assignment $\varphi[\{y_{S_1}/t_1, \ldots, y_{S_n}/t_n\}]$ with $t_i{\in}(S_i)_{\mathcal{A}}$ and $(\mathcal{M}, \varphi[\{y_{S_1}/t_1, \ldots, y_{S_n}/t_n\}]){\not\models}\ C$. But there is a clause $C_H{\in}CS_H$ with $C_H = \{y_{S_1} \mapsto t_1, \ldots, y_{S_n} \mapsto t_n\}{\downarrow}C$ and $\Im_H \models C_H$. Thus there is at least one literal L which is both in C_H and $\Im_H$. If L is a literal introduced by conditioned instantiation, we have a contradiction against Definition 5.1.6. If L is an instance of a literal occurring in C we have $(\mathcal{M}, \varphi[\{y_{S_1}/t_1, \ldots, y_{S_n}/t_n\}]) \models L$ and hence $(\mathcal{M}, \varphi[\{y_{S_1}/t_1, \ldots, y_{S_n}/t_n\}]) \models C$ contradicting the assumption that $(\mathcal{M}, \varphi[\{y_{S_1}/t_1, \ldots, y_{S_n}/t_n\}]){\not\models}\ C$. □

Theorem 5.1.8 (Herbrand Theorem) Let $\mathcal{F}$ be a sentence, CS and CS_H be the above defined clause sets. Then CS is unsatisfiable iff there is a finite subset of CS_H which is unsatisfiable.

Proof:

"$\Rightarrow$" We show this part by contradiction. Assume CS is unsatisfiable and every finite subset of CS_H is satisfiable. Then because of Theorem 3.4.6, CS_H is satisfiable and with Lemma 5.1.7 CS is satisfiable, which contradicts our assumption.

"$\Leftarrow$" If CS_H has a finite unsatisfiable subset, CS_H has no Herbrand model and with Lemma 5.1.7 CS has no $\mathbf{\Sigma}$-model. □

5.2 Ground Resolution

The next step towards a sound and complete calculus is to define a resolution rule on ground level. For this rule we have to prove the soundness and completeness, i.e. we prove that a set of ground clauses is unsatisfiable iff there is a derivation of the empty clause using ground resolution. The only difference between the resolution calculus for the sorted logic and the resolution rule for unsorted first-order logic is a rule eliminating literals of the form $t{\not\in}\top$.

Definition 5.2.1 (Ground Resolution Rule) Let C_1 and C_2 be ground clauses, $L_1{\in}C_1$ and $L_2{\in}C_2$. If L_1 and L_2 are two complementary literals, then $R_g(C_1, C_2, L_1, L_2) := (C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{L_2\})$ is a *ground resolvent* of C_1 and C_2.

Definition 5.2.2 ($\top$-Reduction Rule) Let C be a ground clause containing a literal $t{\not\in}\top$, then $E_g(C, t{\not\in}\top) := C \setminus \{t{\not\in}\top\}$ is an $\top$*-reduction* of C.

Lemma 5.2.3 (Ground Resolution and $\top$-Reduction are Sound) Let C_1 and C_2 be ground clauses, $L_1{\in}C_1$ and $L_2{\in}C_2$, L_1 and L_2 be complementary literals, then for every interpretation $\Im$ having $\Im \models C_1$ and $\Im \models C_2$, $\Im \models R_g(C_1, C_2, L_1, L_2)$ holds. If $(t{\not\in}\top){\in}C_1$ then for every interpretation $\Im$ with $\Im \models C_1$, $\Im \models E_g(C_1, t{\not\in}\top)$ holds.

Proof: The proofs are obvious. □

Lemma 5.2.4 (Unsatisfiability of Ground Unit Clauses) Let CS be an unsatisfiable set of ground unit clauses. Then CS is unsatisfiable iff CS contains two unit clauses with complementary literals or a clause $\{t \not\leq \top\}$.

Proof: We will prove the following equivalent conjecture: a set CS of ground unit clauses has a $\mathbf{\Sigma}$-model iff CS does not contain complementary literals and no literal $t \not\leq \top$.

"$\Rightarrow$" If CS has a $\mathbf{\Sigma}$-model, then by Lemma 5.1.7, CS has a Herbrand model and therefore CS does not contain complementary literals and no literal $t \not\leq \top$.

"$\Leftarrow$" If CS does not contain complementary literals and no literal $t \not\leq \top$, then CS can be turned into a Herbrand model and using Lemma 5.1.7 we conclude that CS has a $\mathbf{\Sigma}$-model. $\square$

Lemma 5.2.5 (Ground Resolution and $\top$-Reduction are Complete) Let CS be an unsatisfiable set of ground clauses. Then there exists a derivation of the empty clause using ground resolution and $\top$-reduction.

Proof: The lemma is proved by induction on the k-parameter [AB70], $k(CS) := \sum \{|C| - 1 \mid C \in CS\}$, where $|C|$ is the number of literals in the clause C.

If $k(CS) = 0$, then from Lemma 5.2.4 we follow that CS contains two clauses with complementary literals or a clause $\{t \not\leq \top\}$. Hence one ground resolution step or one $\top$-reduction step yields the empty clause.

If $k(CS) > 0$, then there exists a non-unit clause C. Doing a case analysis, we separate C into two parts C_1 and C_2 and obtain two unsatisfiable clause sets CS_1 and CS_2 by replacing C by C_1 or C_2, respectively. Since $k(CS_i) < k(CS)$, there are refutations of CS_1 and CS_2 by ground resolution. As all clauses are ground, these resolution proofs can be combined to a resolution proof of the empty clause in CS. $\square$

5.3 Unsorted Unification

After proving the soundness and completeness on the ground level, we will now show how to lift these results on the general level where we also deal with variables. One problem that arises is that if some terms are syntactically equal on ground level they need not to be equal on the general level. For example the ground literals $P(f(a)), \neg P(f(a))$ are syntactically equal if we disregard their sign. Suppose they stem from two literals $P(x_S), \neg P(f(x_T))$ after applying the ground substitution $\sigma = \{x_S \mapsto f(a), x_T \mapsto a\}$. Clearly $P(x_S)$ and $\neg P(f(x_T))$ are not syntactically equal and the unification task is now to find a more general substitution which equals the literals.

Since unification works as in unsorted first-order logic, we give only a brief introduction to unification. A lot of efficient unification procedures are known, e.g. [PW78, MM82]. For reasons of simplicity we will present a rule based version of the Robinson [Rob65] unification procedure following [MM82]. We use the standard notions for unification [Sie89].

Definition 5.3.1 (Notions for Unification) Two objects (terms, atoms, literals) t_1 and t_2 are called *unifiable*, iff there exists a substitution σ such that $\sigma(t_1) = \sigma(t_2)$. In this case the substitution σ is called a *unifier* of t_1 and t_2. A unifier σ of two objects t_1 and t_2 is called an *mgu (most general unifier)*, iff for every unifier λ of t_1 and t_2 there exists a substitution τ, such that $\tau\sigma = \lambda$. If $L_1 = P(t_1, \ldots, t_n)$ and $L_2 = P(s_1, \ldots, s_n)$ are two literals with no variables in common, then $\Gamma = \{t_1 = s_1, \ldots, t_n = s_n\}$ is called the *unification problem* for L_1 and L_2. A substitution σ *solves* a unification problem $\Gamma = \{t_1 = s_1, \ldots, t_n = s_n\}$, iff $\sigma(t_1) = \sigma(s_1), \ldots, \sigma(t_n) = \sigma(s_n)$. A unification problem Γ is called *solved*, iff $\Gamma = \{x_{S_1} = t_1, \ldots, x_{S_n} = t_n\}$ where the x_{S_i} are variables, $x_{S_i} \notin V(t_j)$ and $x_{S_i} \neq x_{S_j}$ for every i and j.

Algorithm 5.3.2 (An Unsorted Unification Algorithm) The input of the algorithm is a unification problem Γ, which is changed by the following six rules until it is solved or the problem is found to be unsolvable:

- $(\{x_S = x_S\} \cup \Gamma) \to (\Gamma)$
- $(\{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} \cup \Gamma) \to (\{t_1 = s_1, \ldots, t_n = s_n\} \cup \Gamma)$
- $(\{x_S = t\} \cup \Gamma) \to (\{x_S = t\} \cup \{x_S \mapsto t\}\Gamma)$
  if x_S is a variable, t a non-variable term and $x_S \notin V(t), x_S \in V(\Gamma)$
- $(\{t = x_S\} \cup \Gamma) \to (\{x_S = t\} \cup \Gamma)$
  if x_S is a variable and t a non-variable term
- $(\{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} \cup \Gamma) \to$ STOP.FAIL
  if $f \neq g$
- $(\{x_S = t\} \cup \Gamma) \to$ STOP.FAIL
  if $x_S \in V(t)$

If $\Gamma = \{t_1 = s_1, \ldots, t_n = s_n\}$ is a unification problem, then the above unification algorithm always terminates on Γ. If the algorithm stops with failure there is no substitution σ solving Γ. Otherwise Γ is solved and the corresponding substitution σ is an mgu of the term pairs $(t_1, s_1), \ldots, (t_n, s_n)$.

Assumption 5.3.3 (Properties of Unifiers) Note that every substitution σ corresponding to a unification problem Γ solved by the above unification algorithm is idempotent and introduces no new variables. So from now on we assume that unifiers are idempotent substitutions in the variables of the unified terms.

Lemma 5.3.4 (Relationship MGU and Ground Unifier) Let $L_1, \ldots, L_n$ be some atoms, σ a ground substitution with $\sigma(L_1) = \ldots = \sigma(L_n)$ and λ an mgu of the L_i. Then $\sigma\lambda = \sigma$.

Proof: As λ is an mgu, we know the existence of a ground substitution τ with $\tau\lambda = \sigma$. Now we have $\sigma\lambda = \tau\lambda\lambda = \tau\lambda = \sigma$, where the second equation holds because λ is idempotent. $\square$

5.4 General Resolution

The idea of the general resolution rule is to simulate every ground resolution step on the general level using unification. Thus the general resolution rule is similar to the ground rule, except that the syntactical equality of literals is not given in advance, but has to be produced by unification.

The idea of the completeness proof for general resolution is to simulate every step of the ground proof by a general resolution step. The problem that always arises at this point is the merging of literals on ground level. Consider the ground clause set

$$\big\{\{P(a), Q(a), a \not< S, a \not< T\}, \{\neg P(a)\}, \{\neg Q(a)\}, \{a{<}T\}, \{a{<}S\}\big\}$$

obtained from the clause set

$$\big\{\{P(x_S), P(x_T), Q(y_S)\}, \{\neg P(a)\}, \{\neg Q(a)\}, \{a{<}T\}, \{a{<}S\}\big\}$$

using conditioned instantiation by the ground substitution

$$\sigma = \{x_S \mapsto a, x_T \mapsto a, y_S \mapsto a\}$$

On ground level we can apply the ground resolution rule to the clauses $\{P(a),$ $Q(a), a \not< S, a \not< T\}$ and $\{\neg P(a)\}$ yielding the clause $\{Q(a), a \not< S, a \not< T\}$. On the general level we can compute the clauses $\{P(x_T), Q(y_S), a \not< S\}$,$\{P(x_S), Q(y_S), a \not< T\}$, but both still contain a literal starting with predicate symbol P. What we need is a kind of merging rule on the general level. This rule is usually called "factorization" and allows the merging of literals having equal signs and predicate symbols by instantiation. Therefore we apply the factorization substitution $\tau = \{x_S \mapsto x_T\}$ for our example first resulting in the clause $\{P(x_T), Q(y_S), x_T \not< S\}$ and then perform the resolution step with $\{\neg P(a)\}$ yielding $\{Q(y_S), a \not< S, a \not< T\}$. This clause has the desired property of the corresponding ground resolvent $\{Q(a), a \not< S, a \not< T\}$ being an instantiation of it.

The factorization problem is also well known from unsorted first-order logic. In $\mathcal{L}_\mathcal{S}$ there are two more problems resulting from the conditioned literals. Let's consider the above example again. On ground level we can perform a resolution step between the clauses $\{P(a), Q(a), a \not< S, a \not< T\}$ and $\{a{<}S\}$ yielding $\{P(a), Q(a), a \not< T\}$. On the general level this step is not possible, because there is no complementary literal for $a{<}S$ in $\{P(x_S), P(x_T), Q(y_S)\}$. Note that the literal $a \not< S$ on ground level is introduced by conditioned instantiation. Thus we cannot lift every ground level step to the general level, but we will show that there is always an ordering on the ground level steps such that every step is liftable. Coming back to our example we first have to resolve between $\{P(x_S), P(x_T), Q(y_S)\}$ and $\{\neg P(a)\}$ yielding $\{Q(y_S), a \not\lessdot S, a \not\lessdot T\}$. Afterwards we can lift the resolution step using $a \lessdot S$.

At this point the second problem occurs. The general resolvent of $\{Q(y_S), a \not\lessdot S, a \not\lessdot T\}$ and $\{a \lessdot S\}$ is $\{Q(y_S), a \not\lessdot T\}$, but the corresponding ground resolvent $\{Q(a), a \not\lessdot T\}$ is not a conditioned instance of the general clause. The corresponding conditioned instance would be $\{Q(a), a \not\lessdot T, a \not\lessdot S\}$. The reason for this divergence is the merging of two conditioned literals on ground level. In general this problem cannot be solved by reordering the resolution steps, so we may have to perform several steps on the general level to lift one ground level step.

Definition 5.4.1 (Factor) Let $C = \{L_1, \ldots, L_n\}$ be a clause, $\{L_i, L_j\} \subseteq C$ $(i \neq j)$, and σ a mgu of $\{L_i, L_j\}$, i.e. $\sigma(L_i) = \sigma(L_j)$, then $F(C, L_i, L_j) := \sigma{\downarrow}C$ is called a *factor* of C.

Lemma 5.4.2 (Factorization is Sound)
If $C = \{L_1, \ldots, L_n\}$ is a clause and $F(C, L_i, L_j)$ a factor of C for every interpretation $\Im$ having $\Im \models \forall\,(C)$, $\Im \models \forall\,(F(C, L_i, L_j))$ holds.

Proof: Follows from Lemma 5.1.3 □

Definition 5.4.3 (General Resolution Rule) Let C_1 and C_2 be two clauses with no variables in common, $L \in C_1$ and $K \in C_2$. If there exists an mgu σ of L and K, such that $\sigma(L)$ and $\sigma(K)$ become two complementary literals then

$$R(C_1, C_2, L, K) := \sigma{\downarrow}\big((C_1 \setminus \{L\}) \cup (C_2 \setminus \{K\})\big)$$

is a *resolvent* of C_1 and C_2.

Definition 5.4.4 (General $\top$-Reduction Rule) Let C be a clause and $(t \not\lessdot \top) \in C$, then

$$E(C, t \not\lessdot \top) := (C \setminus \{t \not\lessdot \top\}) \cup \{x_S \not\lessdot S \mid x_S \in V(t),\ S \neq \top\}$$

is called a $\top$-*Reduction* of C.

Lemma 5.4.5 (Resolution and $\top$-Reduction are Sound) Let C_1 and C_2 be two clauses with no variables in common, $L \in C_1$ and $K \in C_2$. If resolution using the literals L and K is possible then for every interpretation $\Im$ having $\Im \models \forall\,(C_1)$ and $\Im \models \forall\,(C_2)$, $\Im \models \forall\,(R(C_1, C_2, L, K))$ holds. If $(t \not\lessdot \top) \in C_1$ then for every interpretation $\Im$ with $\Im \models \forall\,(C_1)$, $\Im \models \forall\,(E(C_1, t \not\lessdot \top))$ holds.

Proof: For the resolution rule we have: for every interpretation $\Im$, $\Im \models \sigma(L)$ and $\Im \models \sigma(K)$ do not hold both. Thus together with Lemma 5.4.2 and Lemma 5.1.3 we obtain the desired result.

For the $\top$-reduction rule we know that $\Im \models \forall\,(t \not\lessdot \top)$ iff there is at least one variable $x_S \in V(t)$ such that $S_\Im = \emptyset$ iff $\Im \models \forall x_S\ x_S \not\lessdot S$. □

Lemma 5.4.6 (Refutations yield ground substitutions) Let CS be a set of clauses such that different clauses are variable disjoint. If CS can be refuted by resolution, i.e. the empty clause can be derived, then the resulting overall substitution is a ground substitution.

Proof: by induction on the number n of resolution steps.

$n = 1$ The only possibility for a one step refutation are two ground unit clauses or a clause $\{t \not\in \top\}$.

$n > 1$ Let us choose an arbitrary resolution step of the refutation between clauses C_1 and C_2, literals L,K and substitution σ, according Definition 5.4.3. By induction hypothesis the refutation using $CS \cup R(C_1, C_2, L, K)$ yields a ground substitution τ for the variables of the clauses used in the refutation. Now assume for a variable $x_S \in (V(C_1) \cup V(C_2))$ that $x_S \notin DOM(\tau)$. Then $x_S \in DOM(\sigma)$, with $\sigma(x_S) = t$ and $(t \not\in S) \in R(C_1, C_2, L, K)$. Thus x_S is substituted by the ground term $\tau(t)$.

The same argumentation holds for a factorization or an $\top$-reduction step. □

Lemma 5.4.7 (Distributivity of Conditioned Instantiation) Let $C_1 = \{L_1, \ldots, L_n\}$ and $C_2 = \{K_1, \ldots, K_m\}$ be two clauses having no variables in common, $L_1 \in C_1$ and $K_1 \in C_2$ and σ be a substitution. If $\sigma(L_1) \notin Cond(\sigma)$ and $\sigma(K_1) \notin Cond(\sigma)$, then

$$\sigma{\downarrow}\big((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{K_1\})\big) = (\sigma{\downarrow}C_1 \setminus \{\sigma(L_1)\}) \cup (\sigma{\downarrow}C_2 \setminus \{\sigma(K_1)\})$$

Proof:

$$\begin{aligned}
& \sigma{\downarrow}\big((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{K_1\})\big) \\
= \; & \sigma{\downarrow}(\{L_2, \ldots, L_n\} \cup \{K_2, \ldots, K_m\}) \\
= \; & \{\sigma(L_2), \ldots, \sigma(L_n)\} \cup \{\sigma(K_2), \ldots, \sigma(K_m)\} \cup Cond(\sigma) \\
& \text{according Definition 5.1.2} \\
= \; & (\{\sigma(L_2), \ldots, \sigma(L_n)\} \cup Cond(\sigma)) \cup (\{\sigma(K_2), \ldots, \sigma(K_m)\} \cup Cond(\sigma)) \\
= \; & \big((\{\sigma(L_1), \ldots, \sigma(L_n)\} \setminus \{\sigma(L_1)\}) \cup Cond(\sigma)\big) \cup \big((\{\sigma(K_1), \ldots, \sigma(K_m)\} \setminus \{\sigma(K_1)\}) \cup \\
& Cond(\sigma)\big) \\
= \; & \big((\{\sigma(L_1), \ldots, \sigma(L_n)\} \cup Cond(\sigma)) \setminus \{\sigma(L_1)\}\big) \cup \big((\{\sigma(K_1), \ldots, \sigma(K_m)\} \cup Cond(\sigma)) \setminus \\
& \{\sigma(K_1)\}\big) \\
= \; & (\sigma{\downarrow}C_1 \setminus \{\sigma(L_1)\}) \cup (\sigma{\downarrow}C_2 \setminus \{\sigma(K_1)\})
\end{aligned}$$

□

Lemma 5.4.8 (Ground Substitutions swallow more general substitutions) Let $C = \{L_1, \ldots, L_n\}$ be a clause, $C' = \{K_1, \ldots, K_m\}$ be a set of atoms, σ a ground substitution with $DOM(\sigma) = V(C')$ and $\sigma(K_1) = \ldots = \sigma(K_m)$ and let λ be the mgu of $\{K_1, \ldots, K_m\}$, then

$$\sigma{\downarrow}(\lambda{\downarrow}C) = \sigma{\downarrow}C$$

Proof:

$$\begin{aligned}
& \sigma{\downarrow}(\lambda{\downarrow}C) \\
= \; & \sigma{\downarrow}(\{\lambda(L_1), \ldots, \lambda(L_n)\} \cup \mathit{Cond}(\lambda)) \\
= \; & \{(\sigma\lambda)(L_1), \ldots, (\sigma\lambda)(L_n)\} \cup \sigma(\mathit{Cond}(\lambda)) \cup \mathit{Cond}(\sigma) \\
= \; & \{\sigma(L_1), \ldots, \sigma(L_n)\} \cup \sigma(\mathit{Cond}(\lambda)) \cup \mathit{Cond}(\sigma) \\
& \text{follows from Lemma 5.3.4} \\
= \; & \{\sigma(L_1), \ldots, \sigma(L_n)\} \cup \mathit{Cond}(\sigma) \\
& \text{as } \lambda \text{ is more general than } \sigma \text{, we have } \sigma(\mathit{Cond}(\lambda)) \subseteq \mathit{Cond}(\sigma) \\
= \; & \sigma{\downarrow}C
\end{aligned}$$

□

Lemma 5.4.9 (Resolvent Lifting I) Let $C_1 = \{L_1, \ldots, L_n\}$ and $C_2 = \{K_1, \ldots, K_m\}$ be two clauses with no variables in common. Let σ be a ground substitution such that for two arbitrary literals L_1 and K_1 with $\sigma(L_1) \notin \mathit{Cond}(\sigma)$ and $\sigma(K_1) \notin \mathit{Cond}(\sigma)$, $\sigma(L_1)$ and $\sigma(K_1)$ are complementary there exists a sequence of factorization steps and a resolvent $R(C_1', C_2', L', K')$ such that

$$R_g(\sigma{\downarrow}C_1, \sigma{\downarrow}C_2, \sigma(L_1), \sigma(K_1)) = \sigma{\downarrow}R(C_1', C_2', L_1', K_1')$$

Proof: Let $\{L_{i_1}, \ldots, L_{i_k}\} \subseteq C_1$ be the set of literals which merge with L_1 by σ and $\{K_{j_1}, \ldots, K_{j_h}\}$ the corresponding set for C_2. Let λ_1 be the mgu of $\{L_1, L_{i_1}, \ldots, L_{i_k}\}$, λ_2 the mgu of $\{K_1, K_{j_1}, \ldots, K_{j_h}\}$ and $\lambda = \lambda_1 \cup \lambda_2$, $F^*(C_1, L_1, L_i) = \lambda_1 \downarrow C_1$, $F^*(C_2, K_1, K_j) = \lambda_2{\downarrow}C_2$ (we use $F^*()$ as an abbreviation for the necessary sequence of binary factorization steps) and δ the mgu of $\lambda(L_1)$ and $\lambda(K_2)$.

$$\begin{aligned}
& \sigma{\downarrow}R(\lambda_1{\downarrow}C_1, \lambda_2{\downarrow}C_2, \lambda_1(L_1), \lambda_2(K_1)) \\
= \; & \sigma{\downarrow}(\delta{\downarrow}((\lambda_1{\downarrow}(C_1 \setminus \{L_1\}) \cup (\lambda_2{\downarrow}(C_2 \setminus \{K_1\}))) \\
& \text{according Definition 5.4.3} \\
= \; & \sigma{\downarrow}(\delta{\downarrow}(\lambda{\downarrow}((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{K_1\}))) \\
& \text{because } \lambda = \lambda_1 \cup \lambda_2 \text{ and the clauses are variable disjoint} \\
= \; & \sigma{\downarrow}((\delta\lambda){\downarrow}((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{K_1\}))) \\
& \text{follows from Lemma 5.1.4} \\
= \; & \sigma{\downarrow}((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{K_1\}))) \\
& \text{follows from Lemma 5.4.8, because } (\delta\lambda) \text{ is the mgu of the set of atoms} \\
& \{atom(L_1), atom(L_{i_1}), \ldots, atom(L_{i_k}), atom(K_1), atom(K_{j_1}), \ldots, atom(K_{j_h})\}
\end{aligned}$$

$= (\sigma{\downarrow}C_1 \setminus \{\sigma(L_1)\}) \cup (\sigma{\downarrow}C_2 \setminus \{\sigma(K_1)\})$
follows from Lemma 5.4.7

$= R_g(\sigma{\downarrow}C_1, \sigma{\downarrow}C_2, \sigma(L_1), \sigma(K_1))$

□

Lemma 5.4.10 ($\top$-Reduction Lifting I) Let C be a clause, $(t \not\Subset \top) \in C$, σ a ground substitution with $\sigma(t \not\Subset \top) \not\Subset \mathit{Cond}(\sigma)$, then there exists an $\top$-reduction such that

$$E_g(\sigma{\downarrow}C, \sigma(t \not\Subset \top)) = \sigma{\downarrow}E(C', t' \not\Subset \top)$$

Proof: Let $\{L_1, \dots, L_k\} \subseteq C$ be the set of literals which merge with $t \not\Subset \top$ by σ. Let λ be the mgu of $\{t \not\Subset \top, L_1, \dots, L_k\}$, such that $F^*(C, t \not\Subset \top, L_i) = \lambda{\downarrow}C$.

$\sigma{\downarrow}E(\lambda{\downarrow}C, \lambda(t \not\Subset \top))$

$= \sigma{\downarrow}(\lambda{\downarrow}C \setminus \{\lambda(t \not\Subset \top)\} \cup \lambda(\{x_S \not\Subset S \mid x_S \in V(\lambda(t)),\ S \neq \top\}))$

$= \sigma{\downarrow}(\lambda{\downarrow}C) \setminus \{\sigma(\lambda(t \not\Subset \top))\} \cup \sigma(\lambda(\{x_S \not\Subset S \mid x_S \in V(\lambda(t)),\ S \neq \top\}))$
by Lemma 5.4.7

$= \sigma{\downarrow}C \setminus \{\sigma(t \not\Subset \top)\} \cup \sigma(\{x_S \not\Subset S \mid x_S \in V(\lambda(t)),\ S \neq \top\})$
by Lemma 5.4.8

$= \sigma{\downarrow}C \setminus \{\sigma(t \not\Subset \top)\}$
because for all $x_S \in V(t)$ we have $x_S \in V(C)$ and for the added literals of the form $x_S \not\Subset S$ we have $S \neq \top$

$= E_g(\sigma{\downarrow}C, \sigma(t \not\Subset \top))$

□

Lemma 5.4.11 (Resolvent Lifting II) Let $C_1 = \{L_1, \dots, L_n\}$ and $C_2 = \{K_1, \dots, K_m\}$ be two clauses with no variables in common. Let σ be a ground substitution such that for two arbitrary literals L_1 and K_1 with $\sigma(L_1)$ and $\sigma(K_1)$ are complementary and wlog. let $\sigma(L_1) \in \mathit{Cond}(\sigma)$, then there exists a resolvent $R(C_1, C_2, L, K)$ such that

$$R_g(\sigma{\downarrow}C_1, \sigma{\downarrow}C_2, \sigma(L_1), \sigma(K_1)) \cup \{\sigma(L_1)\} = \sigma{\downarrow}R(C_1, C_2, L_1, K_1)$$

Proof: The proof is similar to the proof of Lemma 5.4.9. We only have to change Lemma 5.4.7 slightly in $\sigma{\downarrow}((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{K_1\})) = (\sigma{\downarrow}C_1 \setminus \{\sigma(L_1)\}) \cup (\sigma{\downarrow} C_2 \setminus \{\sigma(K_1)\}) \cup \{\sigma(L_1\}$. □

Remark 5.4.12 (Relaxing the lifting Lemmata) We can relax the premises of Lifting I and Lifting II by assuming two clauses C_1,C_2 and two ground clauses C_{1g},C_{2g} with $\sigma{\downarrow}C_1 = C_{1g} \cup \{L_{1g}, \dots, L_{n'g}\}$ and $\sigma{\downarrow}C_2 = C_{2g} \cup \{K_{1g}, \dots, K_{m'g}\}$. For the potentially complementary literals L_1 and K_1 we have $\sigma(L_1) \in C_{1g}$ and $\sigma(K_1) \in C_{2g}$. Then the following equations

$$R_g(C_{1g}, C_{2g}, \sigma(L_1), \sigma(K_1)) \cup \{L_{1g}, \ldots, L_{n'g}\} \cup \{K_{1g}, \ldots, K_{m'g}\}$$
$$=$$
$$\sigma{\downarrow}R(C_1, C_2, L_1, K_1)$$

$$R_g(C_{1g}, C_{2g}, \sigma(L_1), \sigma(K_1)) \cup \{L_{1g}, \ldots, L_{n'g}\} \cup \{K_{1g}, \ldots, K_{m'g}\} \cup \{\sigma(L_1)\}$$
$$=$$
$$\sigma{\downarrow}R(C_1, C_2, L_1, K_1)$$

hold for Lifting I and Lifting II respectively.

For the $\top$-reduction rule we also have the corresponding lifting Lemma II and the corresponding relaxation.

Theorem 5.4.13 (Completeness of the Rules) Let *CS* be an unsatisfiable set of clauses. Then there exists a derivation of the empty clause using resolution, factorization and $\top$-reduction.

Proof: If *CS* is unsatisfiable, Theorem 5.1.8 guarantees the existence of a finite and unsatisfiable set CS_H of ground clauses. With Lemma 5.2.5 we know that it is possible to derive the empty clause from CS_H using ground resolution and ground $\top$-reduction. Now we show by induction on the k-parameter that the ground proof can be lifted to a proof using resolution, factorization and $\top$-reduction as defined in Definitions 5.4.3, 5.4.4, 5.4.1.

$k(CS_H) = 0$: We have derived the empty clause on ground level from two ground unit clauses C_{1g} and C_{2g} or a ground unit clause $\{t \not\leq \top\}$. In the case of having performed performed a resolution step, the parent clauses are conditioned instances of two clauses C_1,C_2 using a ground substitution σ, e.g. $\{C_1, C_2\} \subseteq CS$, $\sigma{\downarrow}C_1 = C_{1g}$ and $\sigma{\downarrow}C_2 = C_{2g}$. Now we have to distinguish two cases.

Firstly, if the premises of Lemma 5.4.9 are satisfied by C_1, C_2, C_{1g} and C_{2g} we can directly lift this step and have derived the empty clause on the general level. Secondly, if the premises of Lemma 5.4.11 are satisfied by C_1, C_2, C_{1g} and C_{2g} we can also lift the ground step, but we obtain a clause $C' \neq \Box$ on the general level. But as C_{1g} and C_{2g} are ground unit clauses, we have $C' = C_{1g}$ or $C' = C_{2g}$. Thus one more resolution step yields the empty clause on the general level, too.

If we perform a $\top$-reduction step in order to get the empty clause, this step can be lifted to a step on the general level directly.

$k(CS_H) > 0$: There is at least one clause $C_g{\in}CS_H$ with $|C_g| > 1$. We also know of the existence of a clause $C{\in}CS$ with $\sigma{\downarrow}C = C_g$. Again we have to distinguish two cases.

Firstly, if we can split the clause C_g into two nonempty clauses C_{1g} and C_{2g} such that we can also split C into two nonempty clauses C_1 and C_2 with $\sigma_1{\downarrow}C_1 = C_{1g}$ and $\sigma_2{\downarrow}C_2 = C_{2g}$ we are done by induction hypothesis.

Secondly, if we cannot split C_g in the way mentioned above, we have at least one literal $L_g{\in}C_g$ and a literal $L{\in}C$ with $\sigma(L) = L_g$. By induction hypothesis we can lift the refutation of $CS_H \setminus \{C_g\} \cup \{\{L_g\}\}$ using C as the general clause for $\{L_g\}$. This can be done following Remark 5.4.12 where we use the relaxed lifting lemma versions for steps involving clauses stemming from $\{L_g\}$. As a result we obtain a general clause C' with either $\sigma'{\downarrow}C' = C_g \setminus \{L_g\}$ or $\sigma'{\downarrow}C' = C_g$ (σ' is the restriction of σ to the variables occurring in C'). In the first case we have finished the proof by induction hypothesis. In the second case, we again try to split C_g and C' in the described way. This process terminates because as a consequence of Lemma 5.4.6 $V(C') \subset V(C)$ is valid. $\square$

Chapter 6

The Calculus $\mathcal{CL}_{\mathcal{S}}$

The calculus introduced in Chapter 5 is on half the way to a sorted calculus. It uses the additional expressiveness of sorted logic but doesn't exploit the information about sorts by special reasoning mechanisms. For sorted logic the right way to do the sorted reasoning is to use a sorted unification algorithm. This point of view is more efficient than a theory resolution [Sti85] or constraint resolution [Bue91] approach to sorted reasoning.

In this chapter we introduce our sorted unification algorithm which is an extension of the unification algorithm given in [SS89]. For the algorithm we need one additional rule rejecting unification problems which contain variables with empty sorts. Additionally we need to be able to deal with "conditioned" declarations. Translating the sort structure of [Wal87, SS89] into sorted logic we obtain a set of unit clauses containing declarations. In our logic declarations may occur together in a clause with arbitrary literals. They are "conditioned" by other literals. In Section 6.3 we will show that me must consider "conditioned" declarations in order to get a complete calculus.

6.1 Sorted Unification

Definition 6.1.1 (Conditioned Objects) A pair $(t{\lessdot}S, C)$ is called a *conditioned declaration* (*conditioned term*, *conditioned substitution*) if C is a set of literals and $t{\lessdot}S$ a declaration (term, substitution). A conditioned object (t, C) is called ground if t is ground.

Definition 6.1.2 (Well Sorted Terms) Let $\mathbf{L}_{\lessdot}$ be a set of conditioned declarations. Then the set of conditioned well sorted terms $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ of sort S is recursively defined by:

- for every variable $x_S{\in}\mathbf{V}$, $(x_S, \emptyset){\in}\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$
- for every conditioned declaration $(t{\lessdot}S, C){\in}\mathbf{L}_{\lessdot}$ $(S \neq \top)$ with $V(t) = \{x_{S_1}, \ldots, x_{S_n}\}$ and conditioned terms $(t_i, C_i){\in}\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S_i}$, $\sigma{:=}\ \{x_{S_1} \mapsto t_1, \ldots, x_{S_n} \mapsto t_n\}$, $(\sigma(t), \sigma(C) \cup C_1 \ldots \cup C_n){\in}\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$

- for every term t we have $(t,\emptyset)\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},\top}$

We define $\mathbf{T}_{\mathbf{L}_{\lessdot},S} := \{(t,C) \mid (t,C)\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ and $C=\emptyset\}$. $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,S}$ stands for the set of conditioned well sorted ground terms of sort S. Obviously $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,S} = \{(t,C) \mid (t,C)\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ and t is ground $\}$. Thus we always have $\mathbf{T}_{\mathbf{L}_{\lessdot},gr,\top} = \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,\top} \neq \emptyset$, $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,S} \subseteq \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ and $\mathbf{T}_{\mathbf{L}_{\lessdot},S} \subseteq \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$.

For the unification algorithm it is useful to define a binary relation $\sqsubseteq$ which denotes the subsort relationship. If S and T are sorts, then we define $S \sqsubseteq T$ iff there exists a variable x_S with $x_S\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$. Note that if there exists a variable $x_S\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$ there are infinitely many variables of sort S in $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$. The relation $S \sqsubseteq T$ implies $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S} \subseteq \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$.

Seldom we are interested in the condition part of a conditioned object. We say that $t\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ if there is a set of literals C such that $(t,C)\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$. Similarly we say that a declaration $(t\lessdot S)\in\mathbf{L}_{\lessdot}$ if there is a set of literals C such that $(t\lessdot S, C)\in\mathbf{L}_{\lessdot}$. If not necessary we do not mention the conditioned part of a conditioned substitution.

Lemma 6.1.3 (Correctness of Well Sorted Terms) Let $\mathbf{L}_{\lessdot}$ be a set of conditioned declarations. Then for every interpretation $\Im = (\mathcal{M},\varphi)$ with $\Im \models \mathbf{L}_{\lessdot}$ and every conditioned term $(t,C)\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ with $V((t,C)) \subseteq \mathit{DOM}(\varphi)$ we have $\varphi(t)\in\Im(S)$ or $\Im \models C$.

Proof: By structural induction according to Definition 6.1.2 □

Definition 6.1.4 (Well Sorted Substitutions) A conditioned substitution $\sigma^{\mathbf{c}} = (\sigma, C)$ is called *well sorted* with respect to the set of declarations $\mathbf{L}_{\lessdot}$ if for every $x_{S_i}\in\mathit{DOM}(\sigma)$, $(\sigma(x_{S_i}),C_i)\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S_i}$ and $C=\bigcup C_i$.

The composition of two well sorted substitutions can be computed by $\sigma^{\mathbf{c}}(\tau^{\mathbf{c}}) := (\sigma(\tau), \sigma(K)\cup C')$, where $\sigma^{\mathbf{c}} = (\sigma, C)$, $\tau^{\mathbf{c}} = (\tau, K)$ and C' is the restriction of C to the conditions of the variables in $\mathit{DOM}(\sigma)\setminus\mathit{DOM}(\tau)$. The result of the composition is again a well sorted substitution. Thus the set of all well sorted substitutions builds a monoid.

Definition 6.1.5 (Notions for Sorted Unification) In order to define well sorted unification, we have to extend Definition 5.3.1. A unification problem $\Gamma = \{x_{S_1} = t_1, \ldots, x_{S_n} = t_n\}$ is called *well sorted solved* with respect to a set of declarations $\mathbf{L}_{\lessdot}$ if Γ is solved and for every variable x_{S_i} we have $t_i\in\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S_i}$.

Algorithm 6.1.6 (The Sorted Unification Algorithm) The input of the algorithm is a unification problem Γ, which is changed by the following thirteen rules until it is solved or the problem is found to be unsolvable:

(1) $(\{x_S = x_S\}\cup\Gamma) \rightarrow \Gamma$

(2) $(\{x_S = t\}\cup\Gamma) \rightarrow (\{x_S = t\}\cup\{x_S \mapsto t\}\Gamma)$
if x_S is a variable, t a term and $x_S\notin V(t)$, $x_S\in V(\Gamma)$

(3) $(\{t = x_S\} \cup \Gamma) \rightarrow (\{x_S = t\} \cup \Gamma)$
if x_S is a variable and t a non-variable term

(4) $(\{x_S = t\} \cup \Gamma) \rightarrow$ STOP.FAIL
if $x_S \in V(t)$ and $t \neq x_S$

(5) $(\{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} \cup \Gamma) \rightarrow (\{t_1 = s_1, \ldots, t_n = s_n\} \cup \Gamma)$

(6) $(\{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} \cup \Gamma) \rightarrow$ STOP.FAIL
if $f \neq g$

(7) $\Gamma \rightarrow$ STOP.FAIL
if there exists a variable $x_S \in V(\Gamma)$ such that $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,S} = \emptyset$

(8) $(\{x_S = y_T\} \cup \Gamma) \rightarrow$ STOP.FAIL
if $S \neq \top$, $T \neq \top$, $y_T \notin \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ and $x_S \notin \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$ and there is no common subsort of S and T and there are no conditioned declarations $(f(s_1, \ldots, s_n) \lessdot S') \in \mathbf{L}_{\lessdot}$, $S' \sqsubseteq S$ and $(f(t_1, \ldots, t_n) \lessdot T') \in \mathbf{L}_{\lessdot}$, $T' \sqsubseteq T$

(9) $(\{x_S = f(t_1, \ldots, t_n)\} \cup \Gamma) \rightarrow$ STOP.FAIL
if $S \neq \top$, $x_S \notin V(f(t_1, \ldots, t_n))$, $f(t_1, \ldots, t_n) \notin \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$ and there is no declaration $(f(s_1, \ldots, s_n) \lessdot S') \in \mathbf{L}_{\lessdot}$, $S' \sqsubseteq S$

(10) $(\{x_S = f(t_1, \ldots, t_n)\} \cup \Gamma) \rightarrow (\{x_S = f(t_1, \ldots, t_n)\} \cup \{t_1 = s_1, \ldots, t_n = s_n\} \cup \Gamma)$
if $S \neq \top$, $x_S \notin V(f(t_1, \ldots, t_n))$, $f(t_1, \ldots, t_n) \notin \mathbf{T}_{\mathbf{L}_{\lessdot},S}$ and there is a conditioned declaration $(f(s_1, \ldots, s_n) \lessdot S') \in \mathbf{L}_{\lessdot}$, $S' \sqsubseteq S$

(11) $(\{x_S = y_T\} \cup \Gamma) \rightarrow (\{y_T = x_S\} \cup \Gamma)$
if $x_S \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$ and $y_T \notin \mathbf{T}_{\mathbf{L}_{\lessdot},S}$

(12) $(\{x_S = y_T\} \cup \Gamma) \rightarrow (\{x_S = z_V\} \cup \{y_T = z_V\} \cup \Gamma)$
if $S \neq \top$, $T \neq \top$, $x_S \notin \mathbf{T}_{\mathbf{L}_{\lessdot},T}$ and $y_T \notin \mathbf{T}_{\mathbf{L}_{\lessdot},S}$ and V is a maximal sort with $V \sqsubseteq S$ and $V \sqsubseteq T$

(13) $(\{x_S = y_T\} \cup \Gamma) \rightarrow (\{x_S = f(s_1, \ldots, s_n)\} \cup \{y_T = f(t_1, \ldots, t_n)\} \cup \{s_1 = t_1, \ldots, s_n = t_n\} \cup \Gamma)$
if $S \neq \top$, $T \neq \top$, $y_T \notin \mathbf{T}_{\mathbf{L}_{\lessdot},S}$ and $x_S \notin \mathbf{T}_{\mathbf{L}_{\lessdot},T}$ and there are conditioned declarations $(f(s_1, \ldots, s_n) \lessdot S') \in \mathbf{L}_{\lessdot}$, $S' \sqsubseteq S$ and $(f(t_1, \ldots, t_n) \lessdot T') \in \mathbf{L}_{\lessdot}$, $T' \sqsubseteq T$ and $S' \not\sqsubseteq T'$ and $T' \not\sqsubseteq S'$

In order to compute a well sorted substitution from a solved unification problem, we have to do the following. Let $\Gamma = \{x_{S_1} = t_1, \ldots, x_{S_n} = t_n\}$ be the solved unification problem, then $\sigma := \{x_{S_1} \mapsto t_1, \ldots, x_{S_n} \mapsto t_n\}$ is the corresponding unconditioned most general well sorted unifier. $\sigma^{\mathbf{c}} := (\sigma, C_1 \cup \ldots \cup C_n \cup D)$ is a most general well sorted unifier if we have $(t_i, C_i) \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S_i}$ and D is a minimal set of conditions under which we have $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,S_i} \neq \emptyset$ for all $x_{S_i} \in I(\sigma)$. Thus from a solved unification problem we may be able to compute several (but only finitely many) most general well sorted unifiers.

Lemma 6.1.7 (Correctness of the Unification Algorithm) The unification algorithm is correct.

Proof: Can be easily proved for the rules. The proof is achieved by showing that every well sorted (ground) substitution solving the problem after the application of a rule solves the original problem. □

If there is no conditioned declaration in $\mathbf{L}_{\lessdot}$ and sorts are assumed to be non-empty our unification algorithm simplifies to the algorithm of [SS89]. In this case our algorithm computes a minimal set of most general unifiers and we know that the unification problem is undecidable and of type infinitary. In the case of conditioned declarations it is not possible to characterize a minimal set of mgu's without "looking into the conditions". At the end of the algorithm we just collect the conditions. Therefore our algorithm may compute a non minimal set of mgu's. Nevertheless the algorithm is complete in the sense that we can lift every ground step using sorted unification.

Lemma 6.1.8 (Completeness of the Unification Algorithm) Let Γ be a unification problem and $\tau^{\mathbf{c}}$ a well sorted ground substitution which solves Γ. Then our algorithm computes a well sorted mgu $\sigma^{\mathbf{c}}$ which solves Γ such that there is a well sorted ground substitution $\lambda^{\mathbf{c}}$ with $\lambda^{\mathbf{c}}(\sigma^{\mathbf{c}}) = \tau^{\mathbf{c}}$.

Proof: Straightforward using the techniques given in [SS89]. □

Our completeness result is weaker than the completeness result given in [SS89]. But it is sufficient to show the completeness of our sorted calculus. It is not straightforward (and maybe not possible) to obtain a more general result, because this implies a very careful and exact examination of the conditions. The theory coded in the conditions may be arbitrarily complex. Nevertheless there are possible improvements, e.g. if $\sigma^{\mathbf{c}} = (\sigma, C)$ and $\tau^{\mathbf{c}} = (\tau, K)$ are two mgu's generated by the algorithm and if there is a well sorted substitution $\lambda^{\mathbf{c}} = (\lambda, D)$ such that $\lambda(\sigma) = \tau$ and $(\lambda(C) \cup D') \subseteq K$, where D' is the restriction of D to the conditions of the variables in $DOM(\lambda) \setminus DOM(\sigma)$ we can skip $\tau^{\mathbf{c}}$. This corresponds to an application of subsumption on the clause level.

6.2 Examples for Sorted Unification

In the following we present a bunch of examples illustrating the different phenomena coming up with our sorted unification algorithm. Firstly, we show the properties of the algorithm not using conditioned declarations. Secondly, we target on the question of what happens if we need to consider conditioned declarations. The application of the algorithm to our examples is discussed in Chapter 8.

We start with the following set of (unconditioned) declarations: $\mathbf{L}_{\lessdot} := \{(a \lessdot S, \emptyset), (a \lessdot T, \emptyset), (f(y_S) \lessdot S, \emptyset), (f(y_T) \lessdot T, \emptyset)\}$ and the unification problem $\Gamma_0 = \{x_S = x_T\}$.

We have $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<},S} = \mathbf{T}_{\mathbf{L}_{<},S} = \{a, f(a), f(y_S), f(f(a)), \ldots, x_S, y_S, \ldots\}$ and $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<},T} = \mathbf{T}_{\mathbf{L}_{<},T} = \{a, f(a), f(y_T), f(f(a)), \ldots, x_T, y_T, \ldots\}$, where all well sorted terms are unconditioned. Applying the algorithm results in the following unification problems:

$$\Gamma_0 \quad = \quad \{x_S = x_T\}$$

The only rule applicable is rule 13 resulting in the two new problems

$$\begin{array}{rcl} \Gamma_{01} & = & \{x_S = a, x_T = a\} \\ \Gamma_{02} & = & \{x_S = f(y_S), x_T = f(y_T), y_S = y_T\} \end{array}$$

Problem Γ_{01} is solved. Again we are able to apply rule 13 in order to solve problem Γ_{02}. Thus we obtain the following mgu's

$$\begin{array}{rcl} \sigma_1 & = & \{x_S \mapsto a, x_T \mapsto a\} \\ \sigma_2 & = & \{x_S \mapsto f(a), x_T \mapsto f(a)\} \\ \sigma_3 & = & \{x_S \mapsto f(f(a)), x_T \mapsto f(f(a))\} \\ & & \vdots \end{array}$$

Again all declarations are unconditioned and therefore we can skip the conditioned part of the unifiers. This well known example [Wal87, SS89] shows that we may obtain infinitely many most general well sorted unifiers.

Now we replace some of the unconditioned declarations by conditioned declarations. We choose $\mathbf{L}_{<} := \{(a<S, \emptyset), (a<T, \emptyset), (f(y_S)<S, \{P(y_S)\}), (f(y_T)<T, \{R(y_T)\})\}$. Thus $\mathbf{T}_{\mathbf{L}_{<},S} = \{a, x_S, y_S, \ldots\}$ and $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<},S} = \{a, x_S, y_S, \ldots, (f(a), \{P(a)\}), (f(y_S), \{P(y_S)\}), (f(f(a)), \{P(a), P(f(a))\}), \ldots\}$. The sets $\mathbf{T}_{\mathbf{L}_{<},T}$ and $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<},T}$ have a similar structure. Again an unconditioned term is denoted by the term itself. Solving the same problem as above, we get the same solved problems but different conditioned unifiers:

$$\begin{array}{rcl} \sigma_1^{\mathbf{c}} & = & (\{x_S \mapsto a, x_T \mapsto a\}, \emptyset) \\ \sigma_2^{\mathbf{c}} & = & (\{x_S \mapsto f(a), x_T \mapsto f(a)\}, \{P(a), R(a)\}) \\ \sigma_3^{\mathbf{c}} & = & (\{x_S \mapsto f(f(a)), x_T \mapsto f(f(a))\}, \\ & & \qquad \{P(a), R(a), P(f(a)), R(f(a))\}) \\ & & \vdots \end{array}$$

In order to make clear why we choose $t \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<},S}$ as a success condition for a pair $x_S = t$ occurring in a unification problem but $t \in \mathbf{T}_{\mathbf{L}_{<},S}$ as a stop condition for the sorted rules (no sorted rule is applicable to an equation $x_S = t$ with $t \in \mathbf{T}_{\mathbf{L}_{<},S}$) consider the next example. We choose $\mathbf{L}_{<} := \{(x_S<T, \{P(x_S)\}), (x_S<V, \emptyset), (x_A<S, \emptyset), (x_A<T, \emptyset), (a<A, \{Q(a)\})\}$ and solve the unification problem

$$\Gamma_0 \quad = \quad \{x_V = x_T\}$$

The only applicable rule is rule 12 using the conditioned declarations $x_S<T$ and $x_S<V$ resulting in the problem

$$\Gamma_{01} \;=\; \{x_V = x_S, x_T = x_S\}$$

The problem is solved because we have $x_S{\in}\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},V}$ and $x_S{\in}\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},T}$. But the rule 12 is still applicable because $x_S{\notin}\mathbf{T}_{\mathbf{L}_{\lessdot},T}$. Another application of rule 12 using the declarations $x_A{\lessdot}S$ and $x_A{\lessdot}T$ leads to

$$\Gamma_{011} \;=\; \{x_V = x_S, x_T = x_A, x_S = x_A\}$$

and after another application of rule 2 using the equation $x_S = x_A$ we end up in the solved problem

$$\Gamma_{0111} \;=\; \{x_V = x_A, x_T = x_A, x_S = x_A\}$$

The problems Γ_{01} and Γ_{0111} represent the two conditioned unifiers

$$\begin{aligned} \sigma_{01} &= (\{x_V \mapsto x_S, x_T \mapsto x_S\}, \{P(x_S), Q(a)\}) \\ \sigma_{0111} &= (\{x_V \mapsto x_A, x_T \mapsto x_A, x_S \mapsto x_A\}, \{Q(a)\}) \end{aligned}$$

The unifier σ_{0111} is not subsumed by σ_{01} because it contains less conditions. The condition $Q(a)$ ensures that the sort A is not empty, i.e. $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,A} \neq \emptyset$.

6.3 The Sorted Inference Rules

Now we present the sorted counterparts of the inference rules given in Chapter 5. Instead of adding conditional literals, the sorted rules apply the sorted unification algorithm where the well sortedness of the unifier is checked.

Definition 6.3.1 (Sorted Factor) Let $C = \{L_1, \ldots, L_n\}$ be a clause, $\{L_i, L_j\} \subseteq C$ $(i \neq j)$, and $\sigma^{\mathbf{c}} = (\sigma, D)$ a well sorted mgu of $\{L_i, L_j\}$, i.e. $\sigma(L_i) = \sigma(L_j)$, then $F(C, L_i, L_j) := \sigma(C) \cup D$ is called a *sorted factor* of C.

Definition 6.3.2 (Sorted Resolution Rule) Let C_1 and C_2 be two clauses with no variables in common, $L{\in}C_1$ and $K{\in}C_2$. If there exists a well sorted mgu $\sigma^{\mathbf{c}} = (\sigma, D)$ of L and K, such that $\sigma(L)$ and $\sigma(K)$ become two complementary literals then

$$R(C_1, C_2, L, K) := \sigma((C_1 \setminus \{L\}) \cup (C_2 \setminus \{K\})) \cup D$$

is a *sorted resolvent* of C_1 and C_2.

Definition 6.3.3 (Sorted $\top$-Reduction Rule) Let C be a clause and let $(t{\not\lessdot}\top){\in}C$, with $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,S} \neq \emptyset$ for every $x_S{\in}V(t)$ under some minimal conditions D then

$$E(C, t{\not\lessdot}\top) := (C \setminus \{t{\not\lessdot}\top\}) \cup D$$

is called a *sorted* $\top$*-Reduction* of C.

Lemma 6.3.4 (Correctness of the Rules) Let *CS* be a set of clauses and $\mathbf{L}_{\lessdot}$ a set of declarations, such that every declaration in $\mathbf{L}_{\lessdot}$ occurs in a clause in *CS* and the other literals of the clause constitute the condition of the declaration. Then the rules sorted resolution, sorted factorization, and sorted $\top$-reduction are correct.

Proof: Follows from the form of the rules, Lemma 6.1.3 and Lemma 6.1.7 $\Box$

There still remains an open question: which declarations we have to select for $\mathbf{L}_{\lessdot}$ in order to obtain a complete calculus. The general idea is to reduce the number of declarations which have to be taken into account deciding unification. We will show that it is necessary and sufficient to arbitrarily choose from every clause containing declarations only, one declaration to build the well sorted terms. This has to be done dynamically during the derivation process, i.e. if we have derived a new clause only containing positive declarations we may have to change the set $\mathbf{L}_{\lessdot}$. On the other hand no declaration which occurs in a clause containing different literals has to be taken into consideration. These clauses typically occur if the membership of an object to a sort depends on certain properties or some other relations are defined by means of sorts.

The minimal set of literals for building the well sorted terms, is the set of declarations occurring in unit clauses, because every interpretation satisfying the clause set satisfies these literals. The following example shows that this set is too small for a complete calculus.

Example 6.3.5 (Considering Declarations in Unit Clauses is not Sufficient) Let *CS* be the clause set $CS = \Big\{\{P(x_A)\}, \{P(x_B)\}, \{\neg P(a)\}, \{\neg P(b)\}, \{a \lessdot A, b \lessdot B\}\Big\}$. If we only take declarations in unit clauses into account we have $\mathbf{L}_{\lessdot} = \emptyset$ and therefore $\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,A} = \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},gr,B} = \emptyset$. There is no well sorted resolution step. But *CS* is unsatisfiable as the following refutation shows (we use the calculus given in Chapter 5). From $\{P(x_A)\}$ and $\{\neg P(a)\}$ we derive $\{a \not\lessdot A\}$. Using $\{a \lessdot A, b \lessdot B\}$ we conclude $\{b \lessdot B\}$. $\{P(x_B)\}$ and $\{\neg P(b)\}$ lead to $\{b \not\lessdot B\}$ and finally we succeed in deriving the empty clause by using $\{b \lessdot B\}$.

The problem in Example 6.3.5 is the clause $\{a \lessdot A, b \lessdot B\}$ which triggers a case analysis in the refutation proof. The example shows that we can not disregard such clauses. We have to extend the set of declarations used for building the well sorted terms by exactly one literal from each clause containing declarations only. But these declarations are conditioned, i.e. not every interpretation satisfying the clause set satisfies the declaration we choose. Therefore we must keep track of the conditions for the declaration. This leads to the definitions of the previous section.

Now we will develop the definitions and lemmata which enable us to proof the completeness of the calculus.

Definition 6.3.6 (Declaration Clause) A clause C is called a *declaration* clause if C consists of declarations only.

Let CS be a clause set. We define $\mathbf{L}_{<}^{\mathbf{U}} := \{(t{<}S, \emptyset) \mid t{<}S$ occurs in CS in a unit clause$\}$, $\mathbf{L}_{<}^{\mathbf{M}} := \{(t{<}S, C') \mid t{<}S$ occurs in a clause C in CS, and $C = \{t{<}S\} \cup C'\}$, and $\mathbf{L}_{<}^{\mathbf{C}} := \{(t_i{<}S_i, C_i') \mid$ such that for each declaration clause $C_i \in CS$ we choose exactly one declaration $t_i{<}S_i$ with $C_i = \{t_i{<}S_i\} \cup C_i'\}$.

Lemma 6.3.7 (Refutations yield Well Sorted Ground Substitutions) Let CS be a set of clauses. If we can derive the empty clause from CS using the inference rules of Chapter 5 then the resulting overall ground substitution τ is well sorted with respect to $\mathbf{L}_{<}^{\mathbf{M}}$.

Proof: by induction on the number of variables in $DOM(\tau)$
Wlog. we assume that CS is a clause set such that every clause in CS is used exactly once in the refutation proof and all clauses are variable disjoint. If τ is the overall ground substitution then $DOM(\tau) = V(CS)$. In order to be precise we show the existence of a well sorted ground substitution $\tau^{\mathbf{c}} = (\tau, K)$. As the case $S = \top$ is trivial we assume $S \neq \top$.

$|DOM(\tau)| = 1$: then $DOM(\tau) = \{x_S\}$ and there is exactly one clause $C \in CS$ with $V(C) = \{x_S\}$ and for all clauses $C' \in CS$ with $C' \neq C$ we have $V(C') = \emptyset$. Now we focus on the deduction step where x_S is instantiated by a ground term t. For the resolvent (factor) R we know that $(t \not< S) \in R$. Therefore there is a positive literal $t{<}S$ occurring in some clause $C = \{t{<}S\} \cup C'$ in CS, thus $(t{<}S, C') \in \mathbf{L}_{<}^{\mathbf{M}}$, thus $(t, C') \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<}^{\mathbf{M}}, gr, S}$ and therefore $\tau^{\mathbf{c}} = (\{x_S \mapsto t\}, C')$ is well sorted.

$|DOM(\tau)| > 1$: We choose an arbitrary variable $x_S \in DOM(\tau)$ and focus on the resolution step where x_S is instantiated by a term t. Again we know that the resolvent R contains the literal $t \not< S$ and therefore there is a literal $t'{<}S$ occurring in some clause $C = \{t{<}S\} \cup C'$ in CS, $(t'{<}S, C') \in \mathbf{L}_{<}^{\mathbf{M}}$ which is used to eliminate $t \not< S$. We have $\tau[x_S](t) = \tau[x_S](t')$ and because $\tau^{\mathbf{c}}[x_S] = (\tau[x_S], K')$ is well sorted by induction hypothesis we have $(\tau[x_S](t), \tau[x_S](C') \cup K'') \in \mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{<}^{\mathbf{M}}, gr, S}$ where K'' is the restriction of K' to the variables in t. Thus $\tau^{\mathbf{c}} = (\tau, K) = (\tau[x_S] \cup \{x_S \mapsto \tau[x_S](t)\}, K' \cup \tau[x_S](C'))$ is a well sorted ground substitution. $\square$

Theorem 6.3.8 (Completeness of the Calculus: Choosing $\mathbf{L}_{<}^{\mathbf{C}}$ is Sufficient) Let CS be a clause set and let $\mathbf{L}_{<}^{\mathbf{C}}$ be the set of declarations dynamically chosen for unification. Dynamically chosen means that we have to update $\mathbf{L}_{<}^{\mathbf{C}}$ after the generation of a new declaration clause. If CS is unsatisfiable there exists a derivation of the empty clause using sorted resolution, sorted factorization, and sorted $\top$-reduction.

Proof: by induction on the k-parameter [AB70], $k(CS) := \sum \{|C| - 1 \mid C \in CS\}$ where $|C|$ is the number of literals in the clause C.

If $k(CS) = 0$: We know by Theorem 5.4.13 of the existence of a refutation. Lemma 6.3.7 states that this refutation produces a well sorted ground substitution with respect to $\mathbf{L}_{<}^{\mathbf{M}}$. As $k(CS) = 0$ we have $\mathbf{L}_{<}^{\mathbf{M}} = \mathbf{L}_{<}^{\mathbf{C}}$ and therefore the ground substitution is also well sorted with respect to $\mathbf{L}_{<}^{\mathbf{C}}$. Now Lemma 6.1.8 ensures that we can refute the clause set by a single sorted inference step.

If $k(CS) > 0$: There exists a non unit clause C. In a case analysis we separate C into two parts C_1 and C_2 and CS into the clause sets $CS_1 := CS \setminus \{C\} \cup \{C_1\}$ and $CS_2 := CS \setminus \{C\} \cup \{C_2\}$. Both clause sets are unsatisfiable and we have $k(CS_i) < k(CS)$. Thus by induction hypothesis we can refute the two clause sets. The two proofs can be combined, because Theorem 5.4.13 ensures the existence of a common (ground) substitution.

In order to apply the induction hypothesis we must guarantee that all bottom clause sets ($k(CS) = 0$) can be refuted. For the refutation of a bottom clause set we need $\mathbf{L}^{\mathbf{M}}_{\lessdot} = \mathbf{L}^{\mathbf{C}}_{\lessdot}$. If we split a declaration clause we can first refute the bottom clause set containing the declaration we choose for $\mathbf{L}^{\mathbf{C}}_{\lessdot}$. After this refutation we obtain a clause which may contain some additional instances of the other declarations of the splitted clause. Using the factorization rule we can generate a clause where just the literal we chose for $\mathbf{L}^{\mathbf{C}}_{\lessdot}$ is deleted. Now we must dynamically change $\mathbf{L}^{\mathbf{C}}_{\lessdot}$ and choose one of the other literals from the new, smaller declaration clause. If we split a clause which is not a declaration clause but contains some declarations we can first refute the part not containing declarations. The refutation results in a declaration clause. We change $\mathbf{L}^{\mathbf{C}}_{\lessdot}$ and continue. If we split a clause not containing any declaration we can continue in an arbitrary order. □

Example 6.3.9 *We again consider the clause set of Example 6.3.5. If we choose $\mathbf{L}^{\mathbf{C}}_{\lessdot} := \{(a \lessdot A, \{b \lessdot B\})\}$ there is only one possible resolution step between $\{P(x_A)\}$ and $\{\neg P(a)\}$ resulting in $\{b \lessdot B\}$. This clause subsumes the clause $\{a \lessdot A, b \lessdot B\}$ and we obtain the new clause set $\Big\{\{P(x_A)\}, \{P(x_B)\}, \{\neg P(a)\}, \{\neg P(b)\}, \{b \lessdot B\}\Big\}$. Now we also have to change $\mathbf{L}^{\mathbf{C}}_{\lessdot}$ into $\mathbf{L}^{\mathbf{C}}_{\lessdot} := \{(b \lessdot B, \emptyset)\}$. Again only one resolution step between $\{P(x_B)\}$ and $\{\neg P(b)\}$ is applicable which yields the empty clause.*

Note that we could also have chosen $\mathbf{L}^{\mathbf{C}}_{\lessdot} := \{(b \lessdot B, \{a \lessdot A\})\}$ as the starting set of declarations. We can derive $\{a \lessdot A\}$ by resolving $\{P(x_B)\}$ and $\{\neg P(b)\}$. The clause $\{a \lessdot A\}$ again subsumes $\{a \lessdot A, b \lessdot B\}$ and we obtain the new clause set $\Big\{\{P(x_A)\}, \{P(x_B)\}, \{\neg P(a)\}, \{\neg P(b)\}, \{a \lessdot A\}\Big\}$ and the new set of declarations $\mathbf{L}^{\mathbf{C}}_{\lessdot} := \{(a \lessdot A, \emptyset)\}$. Now we can refute the new clause set by resolving $\{P(x_A)\}$ and $\{\neg P(a)\}$.

The calculus established by Theorem 6.3.8 is not the only one possible. Depending on the set of declarations we choose and the way we deal with the conditions of conditioned declarations there are more different complete and correct calculi. We can give some examples.

- We choose $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ as the set of declarations for unification. The consequence of this choice is that we need not to change the set of declarations during deduction and that the calculus can be straightforward combined with all strategies (e.g. set of support) known from unsorted resolution preserving completeness. The disadvantage of choosing $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ may be a collapse of the sort structure due to the additional declarations. The clause $\{x_D \lessdot LL, \neg \mathit{Lies}(\mathit{lion}, x_D, y_D), y_D \lessdot LL\}$

of Example 4.2.5 demonstrates such a collapse. We have $(x_D{<}LL){\in}\mathbf{L}^{\mathbf{M}}_{\lessdot}$ and therefore the sorts D and LL collapse into one sort. This increases the number of resolution possibilities significantly such that nearly all advantages of the sorted formalization are lost.

- We change the unification Algorithm 6.1.6 in the following way: Every condition of type $t{\in}\mathbf{T}_{\mathbf{L}_{\lessdot},S}$ (in rules 10 to 13) is replaced by $t{\in}\mathbf{T}^{\mathbf{c}}_{\mathbf{L}_{\lessdot},S}$. The computation of the conditioned unifier from a solved problem is also changed. For each solved pair $x_{S_i} = t_i$ with $t_i{\notin}\mathbf{T}_{\mathbf{L}_{\lessdot},S_i}$ or $\mathbf{T}_{\mathbf{L}_{\lessdot},gr,S_j} = \emptyset$ for some $x_{S_j}{\in}V(t_i)$ we add the condition $t_i{\not\lessdot}\, S_i$ to the unconditioned unifier. No extra conditions are added with respect to empty sorts. Thus all conditions are of the form $t{\not\lessdot}\, S$. These changes simplify the computation of well sorted unifiers significantly. But the resulting calculus is less restrictive, because the conditioned literals of the form $t{\not\lessdot}\, S$ can be resolved with all positive declarations of the appropriate form occurring in the clause set. Instead of choosing the original unification algorithm only the declarations in $\mathbf{L}^{\mathbf{C}}_{\lessdot}$ are considered.

- In general we can establish any calculus between the calculus given in Section 5 using unsorted unification and the calculus of Theorem 6.3.8 using sorted unification with respect to $\mathbf{L}^{\mathbf{C}}_{\lessdot}$. The idea is to add to all pairs $x_S = t$ of an unsorted solved unification problem the conditional literal $t{\not\lessdot}\, S$ if we do not have $t{\in}\mathbf{T}_{\mathbf{L}^{\mathbf{U}}_{\lessdot},S}$ or $\mathbf{T}_{\mathbf{L}^{\mathbf{U}}_{\lessdot},gr,S'} \neq \emptyset$ for some $x_{S'}{\in}V(t_i)$. Thus we have all freedom which equations of the unification problem we want to solve in an unsorted way and which we want to weaken until they are well sorted.

Although the presented calculi are simpler, as they either simplify the unification algorithm or the process of choosing the relevant declarations, we believe that the calculus established by Theorem 6.3.8 is most efficient. This statement is based upon the fact that that in all "practical" examples the sorted unification problem computes a finite set of unifiers in polynomial time and that the number of conditioned declarations is pretty small.

The calculus choosing $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ as the set of declarations is close to the calculus presented in [WO90]. The treatment of the conditions makes the difference between the two calculi. In [WO90] we also considered $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ as the set of declarations, But residue literals of the form $t{\not\lessdot}\, S$ for every assignment $x_S{\mapsto}t$ were added to a clause after the application of a substitution. In $\mathcal{CL}_S$ we do not add literals of the form $t{\not\lessdot}\, S$, but literals which prove $t{\lessdot}S$ if they are deleted by the inference rules.

Chapter 7

Refinements on the Calculus $\mathcal{CL}_{\mathcal{S}}$

7.1 Compilation

If we reverse the relativization process of Section 3.4 we are able to compile unsorted first-order formulas into sorted formulas and we may optimize sorted formulas.

Algorithm 7.1.1 (Compilation into Sorted Logic) The input of the algorithm is a sentence of unsorted first-order logic.

(1) Attach the top sort $\top$ to all unsorted variables.

(2) Translate the formulas into CNF, see Chapter 4.

(3) Select a set of 1-place predicates for the transformation into sorts.

(4) Change all occurrences of $S(t)$ for the predicates in the selected set into $t \lessdot S$.

(5) Build $\mathbf{L}^{\mathbf{U}}_{\lessdot}$.

(6) For every clause C containing a negated declaration of the form $x_S \not\lessdot T$: if $S \sqsubseteq T$ (with respect to $\mathbf{L}^{\mathbf{U}}_{\lessdot}$) and $\mathbf{T}_{\mathbf{L}^{\mathbf{U}}_{\lessdot},gr,S} \neq \emptyset$ delete the literal $x_S \not\lessdot T$; if $T \sqsubseteq S$ change C into $\{x_S \mapsto x_T\}(C \setminus \{x_S \not\lessdot T\})$.

Note that the application of step 6 may produce the empty clause.

The steps 5 to 6 can also be applied in order to improve the representation of problems which are given in a sorted formalization. The compilation works for arbitrary unsorted clause sets. This is not the case for the compilation algorithm SOGEN of [SS89]. Our extensions to the work of [SS89] are necessary and sufficient to obtain this general result:

- Declarations can be used in the same way as any other atom.
- Sorts may denote empty sets.
- We have a fixed sort symbol $\top$ which denotes the topsort “Any”.

Let us apply Algorithm 7.1.1 to the clause normal forms of Example 4.2.5 and Example 4.2.6.

7.1.1 Compiling the Example “Condensed Detachment”

We start with the unsorted clause normal form of Example 4.2.6 where we have applied Step 1 and Step 2 of the algorithm.

$$\begin{array}{ll}
(1) & \{\neg P(i(x_\top, y_\top)), \neg P(x_\top), P(y_\top)\} \\
(2) & \{P(i(i(x_\top, y_\top), i(i(y_\top, z_\top), i(x_\top, z_\top))))\} \\
(3) & \{P(i(i(n(x_\top), x_\top), x_\top))\} \\
(4) & \{P(i(x_\top, i(n(x_\top), y_\top)))\} \\
(5) & \{\neg P(i(a, a))\}
\end{array}$$

As there is only one 1-place predicate symbol P we select P and turn it into a sort. Thus after Step 3 and Step 4 we get the clause set

$$\begin{array}{ll}
(1) & \{i(x_\top, y_\top) \notin P, x_\top \notin P, y_\top \in P\} \\
(2) & \{i(i(x_\top, y_\top), i(i(y_\top, z_\top), i(x_\top, z_\top))) \in P\} \\
(3) & \{i(i(n(x_\top), x_\top), x_\top) \in P\} \\
(4) & \{i(x_\top, i(n(x_\top), y_\top)) \in P\} \\
(5) & \{i(a, a) \notin P\}
\end{array}$$

So far the changes have not affected the structure of the clause set but the syntax. Step 5 leads to $\mathbf{L}^{\mathbf{U}}_{\in} = \{i(i(x_\top, y_\top), i(i(y_\top, z_\top), i(x_\top, z_\top))) \in P$, $i(i(n(x_\top), x_\top), x_\top) \in P$, $i(x_\top, i(n(x_\top), y_\top)) \in P\}$. The only candidate for application of Step 6 is the literal $x_\top \notin P$ in the first clause. As we have $P \sqsubseteq \top$ we change the sort of the variable $x_\top$ into x_P, delete the literal from the clause and result in the final compiled clause set

$$\begin{array}{ll}
(1) & \{i(x_P, y_\top) \notin P, y_\top \in P\} \\
(2) & \{i(i(x_\top, y_\top), i(i(y_\top, z_\top), i(x_\top, z_\top))) \in P\} \\
(3) & \{i(i(n(x_\top), x_\top), x_\top) \in P\} \\
(4) & \{i(x_\top, i(n(x_\top), y_\top)) \in P\} \\
(5) & \{i(a, a) \notin P\}
\end{array}$$

It seams as if the effect of the compilation is rather small, because we have saved one literal only. But in Section 8.3 we will show that the proof using the compiled clause set is unique whereas the proof using the original unsorted clause set requires search in an infinite search space.

7.1.2 Compiling the Clause Set of “The Lion and the Unicorn”

Now we consider Example 4.2.5. As the problem is given in a sorted form we start the compilation with Step 5. We get $\mathbf{L}^{\mathbf{U}}_{\in} = \{(1)1-(10)1, (15)1-(17)1, (22)1-(39)1\}$ (the notation $(n)m$ refers to the m'th literal of clause n). Step 6 is applicable to the clauses (40) to (47). The literals of the form $x_D \notin LL$ and $x_D \notin LU$ can be deleted after having changed the sort of the variables because we have $LU \sqsubseteq D$ and $LL \sqsubseteq D$ with respect to $\mathbf{L}^{\mathbf{U}}_{\in}$. Therefore we get the compiled clauses:

(40) $\{x_D {<\!\!\!\cdot}\, LL, \neg \mathit{Lies}(\mathit{lion}, x_D, y_D), y_D {<\!\!\!\cdot}\, LL\}$
(41) $\{x_D {<\!\!\!\cdot}\, LL, \mathit{Lies}(\mathit{lion}, x_D, y_{LL})\}$
(42) $\{\neg \mathit{Lies}(\mathit{lion}, x_{LL}, y_{LL})\}$
(43) $\{\mathit{Lies}(\mathit{lion}, x_{LL}, y_D), y_D {<\!\!\!\cdot}\, LL\}$
(44) $\{x_D {<\!\!\!\cdot}\, LU, \neg \mathit{Lies}(\mathit{uni}, x_D, y_D), y_D {<\!\!\!\cdot}\, LU\}$
(45) $\{x_D {<\!\!\!\cdot}\, LU, \mathit{Lies}(\mathit{uni}, x_D, y_{LU})\}$
(46) $\{\neg \mathit{Lies}(\mathit{uni}, x_{LU}, y_{LU})\}$
(47) $\{\mathit{Lies}(\mathit{uni}, x_{LU}, y_D), y_D {<\!\!\!\cdot}\, LU\}$

Here we could delete all literals of the form $x_D \not{<\!\!\!\cdot}\, LL$ ($x_D \not{<\!\!\!\cdot}\, LU$) in the clauses (40) to (47). This deletion will allow us to prove the theorem exploiting a very small search space, see Section 8.2.

7.2 Refinements on the Inference Rules

Some refinements concerning the interaction of the unification algorithm and the inference rules are possible. Consider the clauses

(1) $\{P(x_S), Q(x_S)\}$
(2) $\{a {<\!\!\!\cdot}\, S, f(x_\top) {<\!\!\!\cdot}\, V\}$
(3) $\{\neg P(y_S)\}$
(4) $\{\neg Q(y_S)\}$
(5) $\{f(x_\top) \not{<\!\!\!\cdot}\, V\}$

We choose $\mathbf{L}^{\mathbf{C}}_{<\!\!\!\cdot} = \{(a {<\!\!\!\cdot}\, S, \{f(x_\top) {<\!\!\!\cdot}\, V\})\}$ and compute a resolvent between (1)1 and (3)1

(6) $\{Q(x_S), f(y_\top) {<\!\!\!\cdot}\, V\}$

with unconditioned well sorted unifier $\sigma = \{y_S \mapsto x_S\}$ and conditioned unifier $\sigma^{\mathbf{c}} = (\{y_S \mapsto x_S\}, \{f(y_\top) {<\!\!\!\cdot}\, V\})$. The condition $f(y_\top) {<\!\!\!\cdot}\, V$ is the (only) minimal condition which guarantees $\mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{<\!\!\!\cdot}, gr, S} \neq \emptyset$. Next we built a resolvent between (6)1 and (4)1

(7) $\{f(y_\top) {<\!\!\!\cdot}\, V, f(y'_\top) {<\!\!\!\cdot}\, V\}$

Again we get the condition $f(y'_\top) {<\!\!\!\cdot}\, V$, because the sort S is the sort of a variable in the codomain of the unconditioned unifier. We apply factorization to (7)1 and (7)2

(8) $\{f(y_\top) {<\!\!\!\cdot}\, V\}$

and finally refute the clause set by resolving (8)1 and (5)1

(9) $\square$

As the top sort $\top$ is non empty by definition, no conditions need to be added to the clauses (8) and (9).

What we want to show with the example is that the conditions coming from $\mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{\lessdot},gr,S} \neq \emptyset$ need not to be added to the resolvent if there are some variables of sort S still occurring in the resolvent. Therefore we can get a closer interaction between the inference rules and the unification algorithm.

In the following assume that we have solved a unification problem in order to apply one of our inference rules. Let σ be the unconditioned well sorted unifier and $\mathbf{S}_R$ the set of sorts occurring in the resulting clause after application of σ. Then for the conditioned unifier $\sigma^{\mathbf{c}}$, we only need to add non-emptiness conditions for sorts S for which we have: there is a variable $x_S \in I(\sigma)$ and $S \notin \mathbf{S}_R$. It should be clear that this change preserves correctness, because if the sort S is empty in some interpretation there is still a variable of sort S in the clause or we have added the conditions. Completeness is also guaranteed because the resulting clause is smaller.

Solving the example using the modified rules, we again compute a resolvent between (1)1 and (3)1

$$(6') \quad \{Q(x_S)\}$$

We have not added the condition $f(y_\top) \lessdot V$, because x_S occurs in the resolvent $\{Q(x_S)\}$. Next we built a resolvent between (6′)1 and (4)1

$$(7') \quad \{f(y_\top) \lessdot V\}$$

As the sort S no longer occurs in the resolvent, we must add the condition $f(y_\top) \lessdot V$. Building a resolvent between (7′) and (5)1 refutes the clause set.

$$(8') \quad \Box$$

The example demonstrates that a tight coupling between unification and the inference rules results in shorter proofs.

7.3 Refinements on the Unification Algorithm

It may happen that the unification algorithm runs in a cycle, i.e. the algorithm does not terminate. Sometimes this is a result of producing infinitely many unifiers (see Section 6.2), but very often it can be easily detected that the loop can not contribute to a solution. Consider the unification problem

$$\Gamma_{01} \quad = \quad \{x_D = x_{LL}, y_D = f(x_{LL})\}$$

related to Example 4.2.5. We choose $\mathbf{L}^{\mathbf{C}}_{\lessdot} = \{(1)1-(10)1, (15)1-(17)1, (22)1-(39)1\}$ and solve Γ_{01}. Only rule 10 of the unification algorithm is applicable to the second equation which yields the following the seven problems

$$\begin{array}{lcl}
\Gamma_{011} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Mo}\} \\
\Gamma_{012} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Tu}\} \\
\Gamma_{013} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{We}\} \\
\Gamma_{014} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Th}\} \\
\Gamma_{015} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Fr}\} \\
\Gamma_{016} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Sa}\} \\
\Gamma_{017} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Su}\}
\end{array}$$

The problems Γ_{011}-Γ_{013} can be solved (see Section 8.2) but the other problems loop forever. For example after applying rule 2 to Γ_{014}, the only applicable rule is rule 13 to the last equation yielding the problems

$$\begin{array}{lcl}
\Gamma_{0141} & = & \{x_D = x_{LL}, y_D = f(x_{Th}), x_{LL} = f(z_{Tu}), \\
 & & x_{Th} = f(x_{Fr}), z_{Tu} = x_{Fr}\} \\
\Gamma_{0142} & = & \{x_D = x_{LL}, y_D = f(x_{Th}), x_{LL} = f(z_{We}), \\
 & & x_{Th} = f(x_{Fr}), z_{We} = x_{Fr}\} \\
\Gamma_{0143} & = & \{x_D = x_{LL}, y_D = f(x_{Th}), x_{LL} = f(z_{Th}), \\
 & & x_{Th} = f(x_{Fr}), z_{Th} = x_{Fr}\}
\end{array}$$

And again after applying rule 2, rule 13 is applicable to the last equation. Knowing the example, it becomes clear that equations of the form $z_{We} = x_{Fr}$ can not be well sorted solved. This can be detected during the unification process. As there are only finitely many possibilities of combining variables of two different sorts (days of the week), after some more steps an equation will arise which we tried to solve before (modulo renaming). The algorithm stucks in a loop. This fact can be detected and results in the following algorithm.

Algorithm 7.3.1 (Loop Detection) *We attach to a unification problem Γ a set of equations E_Γ. Initially we put $E_\Gamma = \Gamma$. The straightforward way to implement the unification algorithm is to work on a list of open unification problems. Always the first problem is selected, all rules are applied and the new unification problems are appended to the end of the list. Now we increment this unification algorithm by the following rules:*

- *After the application of the rules 5, 10 to the actual problem Γ, 12, and 13 the new equations are added to E_Γ.*
- *Every time a new equation is created or an equation is changed in the actual problem Γ, it is checked whether it occurs (modulo renaming) in E_Γ. If the equation occurs in E_Γ, the unification problem is suspended.*
- *No rules are applied to suspended problems.*
- *If there are only suspended problems left, all problems are deleted.*
- *If a problem Γ is solved, the set E_Γ is subtracted from all sets E_Δ of problems Δ which are not solved and still contained in the list. If after this operation no equation of Δ occurs in E_Δ, the suspendation of Δ is undone.*

7.4 Reduction

Reduction techniques are an important tool in automated reasoning in order to restrict the search space. The general idea is to delete clauses which are not necessary for the derivation of the empty clause. We will discuss purity, tautology deletion and subsumption within $\mathcal{CL}_\mathcal{S}$.

7.4.1 Tautology Deletion

A clause is called a *tautology* if it contains two complementary literals. Obviously a tautology is valid in all interpretations. If CS is an unsatisfiable clause set containing a tautology C then $CS \setminus \{C\}$ is still unsatisfiable. Therefore the clause C can be deleted.

In general we have to be careful while deleting tautologies, because application of the rules in a calculus may be restricted in a way such that some proofs can only be found using tautologies [BG90]. But this is not the case for $\mathcal{CL}_\mathcal{S}$. Thus we can delete tautologies without affecting completeness.

7.4.2 Purity Deletion

A clause C (or a literal) is called *pure* if no inference rule is applicable to at least one of it's literals (to the literal). As a consequence this literal cannot be deleted by the application of an inference rule. Therefore the clause containing the literal cannot contribute to a derivation of the empty clause. Thus pure clauses can be deleted.

According to this definition adding purity deletion to our rules results in an (highly) incomplete (but highly restrictive) calculus. On one hand, nearly all clauses containing declarations can be deleted (e.g. see Example 4.2.4). On the other hand, even if we forbid the consideration of declarations as pure literals the rule is still too strong. For example in the clause set

$$\begin{array}{ll} (1) & \{Q(a)\} \\ (2) & \{\neg Q(a), a{<}S\} \\ (3) & \{P(x_S)\} \\ (4) & \{\neg P(y_S)\} \end{array}$$

the literals $a{<}S$, $P(x_S)$ and $\neg P(y_S)$ are pure if we choose $\mathbf{L}^{\mathbf{C}}_{<} = \emptyset$ as the set of declarations considered in unification. But if we delete one of the clauses $\{P(x_S)\}$, $\{\neg P(y_S)\}$ we are no longer able to derive the empty clause. We can refute the clause set by resolving between (1)1 and (2)1

$$(5) \quad \{a{<}S\}$$

Change our set of declarations into $\mathbf{L}^{\mathbf{C}}_{<} = \{(a{<}S, \emptyset)\}$ and derive the empty clause in one resolution step using (3)1 and (4)1

(6) $\square$

The example shows that choosing $\mathbf{L}^{\mathbf{C}}_{<}$ is too strong for purity deletion. But it can easily be proved that choosing $\mathbf{L}^{\mathbf{M}}_{<}$ for purity deletion and disregarding declarations yields a complete calculus. Note that for the above example we have $\mathbf{L}^{\mathbf{M}}_{<} = \{(a{<}S, \{\neg Q(a)\})\}$ and using this set of declarations the literals $P(x_S)$ and $\neg P(y_S)$ are not pure.

7.4.3 Subsumption

Subsumption is the most important reduction rule for resolution based calculi. If C and D are two clauses and σ a substitution such that $\sigma(C) \subseteq D$, then C *subsumes* D. The clause D can be deleted. The deletion does not affect completeness because if we use D in a refutation we can replace D by C because C contains less literals and is more general.

The above definition is an intuitive definition of subsumption and has to be used carefully. For example, every factor D of a clause C is subsumed by C according to the above definition. But factors are needed in order to guarantee completeness. The crux here is to view clauses as sets of literals. If we view clauses as multisets of literals the application of a substitution to a clause does not merge literals and therefore a factor is no longer subsumed by it's parent clause. This point is also stressed in [BG90]. From a computational point of view the detection of a clause which can be subsumed (or subsumes) by a different clause is an NP-complete problem [KN86]. Nevertheless subsumption is needed, because the deletion of a subsumed clause may avoid the generation of infinitely many clauses. Additionally, techniques [LO80, Ohl90, Soc91] have been developed in order to make the problem tractable in practise.

In our sorted logic we have to apply subsumption modulo well sorted unifiers (matchers). If we consider conditioned declarations in order to find the appropriate matcher the application of the matcher may add new literals to the clause. This makes the problem of deciding subsumption very hard. Thus we suggest to use $\mathbf{L}^{\mathbf{U}}_{<}$ to decide the well sortedness of the matcher. Then the problem has the same complexity than in the unsorted case, because sorted matching can be decided in polynomial time. We will show how to apply subsumption in Section 8.2.

7.5 Combination with Strategies

There are many strategies discussed in the literature (e.g. [CL73, Lov78]). The general idea is to restrict the application of inference rules to "useful steps". The most famous and mostly applied strategy is the "Set of Support" (in the following SOS) strategy [WRC65]. SOS is implemented in nearly any automated reasoning system based on resolution.

The SOS strategy forbids resolution steps between clauses coming from the axioms. In order to be precise: If we want to prove $Ax \Rightarrow Th$ where Ax is a set of axioms and Th a set of theorems we show that $Ax \ \wedge \ \neg Th$ is unsatisfiable. As it does not make sense to assume Ax to be unsatisfiable, every proof must involve clauses coming from $\neg Th$. Thus it is sufficient to allow resolution steps between clauses coming from $\neg Th$ and clauses coming from Ax only. In order to put it in other words: We forbid resolution steps between clauses coming from the axioms.

The SOS strategy is complete for unsorted first-order logic. It is not complete in combination with our sorted calculus, as the following example shows. If we want to prove

$$(Q(a) \ \wedge \ \forall x_\top \, (Q(x_\top) \Rightarrow x_\top \lessdot S) \ \wedge \ \forall x_S \, P(x_S)) \Rightarrow \exists y_S \, P(y_S)$$

we obtain the clause set

(1) $\{Q(a)\}$
(2) $\{\neg Q(x_\top), x_\top \lessdot S\}$
(3) $\{P(x_S)\}$
(4) $\{\neg P(y_S)\}$

where clause (4) comes from the negated theorem. The SOS strategy only allows resolution steps between clause (4) and one of the clauses (1), (2), or (3). Resolution between (3) and (4) produces the unification problem $\Gamma = \{x_S = y_S\}$ which cannot be solved with respect to $\mathbf{L}^{\mathbf{C}}_{\lessdot} = \emptyset$. Thus there is no inference rule applicable. But the clause set is unsatisfiable. From (1) and (2) we derive

(5) $\{a \lessdot S\}$

and change $\mathbf{L}^{\mathbf{C}}_{\lessdot} = \{a \lessdot S\}$. Now the previous unification problem Γ is in solved form (we can no longer apply rule 7) and using (4) and (3) we conclude

(6) $\square$

As SOS is s useful strategy (we will solve all examples in Chapter 8 using SOS), we are interested in a (complete) combination of SOS and our sorted calculus. There are several possibilities to overcome incompleteness:

a) We can choose $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ for unification. The choice results in a complete calculus but as we pointed our at the end of Chapter 6 a consequence of choosing $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ may be the lost of all advantages of the sorted machinery. Therefore we may allow a mixture of SOS steps with respect to $\mathbf{L}^{\mathbf{M}}_{\lessdot}$ and steps with respect to $\mathbf{L}^{\mathbf{C}}_{\lessdot}$.

b) We can allow steps between clauses coming from the axioms but still use the inference rules with respect to $\mathbf{L}^{\mathbf{C}}_{\lessdot}$. For example, we may allow one axiom step after six SOS steps. The axiom steps can be restricted to clauses containing declarations, i.e. we only allow steps between two clauses where at least one clause contains a declaration.

c) We derive in the standard SOS way with respect to $\mathbf{L}^{\mathbf{C}}_{\lessdot}$ until the empty clause is derived or no rule is applicable. If no step is possible we allow one step between axiom clauses as mentioned in b) and then try to continue with SOS. This strategy is again incomplete in general but tractable in practise.

All examples in Chapter 8 are solved with strategy c). There is no real disadvantage in using strategy a) or b), because the additional steps may lead to shorter proofs (e.g. see [MW92]).

Chapter 8

Solving The Examples

Now we will discuss the solutions of the examples introduced in Section 3.3. We will refer to clauses and literals by their number using the format (*clause number*)*literal number*. For example in the Schubert's Steamroller problem (see 4.2.4) (13)2 denotes the literal $\neg M(z_A, x_A)$ in clause $\{E(x_A, y_P), \neg M(z_A, x_A), \neg E(z_A, v_P), E(x_A, z_A)\}$.

Unification problems are denoted by the Greek letter Γ and an index which is a sequence of the digits 0 to 9. The idea is to represent the tree structure arising during the application of the unification algorithm. Thus if we have a problem Γ_{12} we know that Γ_{12} is the second problem stemming from Γ_1 after the application of a rule of the algorithm. We have used this convention in Section 6.2.

8.1 Schubert's Steamroller

Schubert's Steamroller is one of the most famous examples showing the efficiency of sorted deduction. It represents a standard example in automated theorem proving. It can also be solved using techniques different to sorted reasoning, e.g. UR-resolution (unit resulting resolution [AO83, MOW76]) or other strategies [BLS87]. We just want to show that our approach works in the same way the much simpler logics of [Wal87, SS89] do, if the formulas are simple. The starting clause set computed in Example 4.2.4 is

(1)	$\{x_W \leqslant A\}$	(2)	$\{x_F \leqslant A\}$	(3)	$\{x_B \leqslant A\}$
(4)	$\{x_C \leqslant A\}$	(5)	$\{x_S \leqslant A\}$	(6)	$\{lupo \leqslant W\}$
(7)	$\{foxy \leqslant F\}$	(8)	$\{tweety \leqslant B\}$	(9)	$\{raupy \leqslant C\}$
(10)	$\{schnecky \leqslant S\}$	(11)	$\{muesli \leqslant G\}$	(12)	$\{x_G \leqslant P\}$
(13)	$\{E(x_A, y_P), \neg M(z_A, x_A), \neg E(z_A, v_P), E(x_A, z_A)\}$				
(14)	$\{M(x_C, z_B)\}$	(15)	$\{M(y_S, z_B)\}$	(16)	$\{M(z_B, x_F)\}$
(17)	$\{M(x_F, y_W)\}$	(18)	$\{\neg E(z_W, y_F)\}$	(19)	$\{\neg E(z_W, x_G)\}$
(20)	$\{E(x_B, y_C)\}$	(21)	$\{\neg E(x_B, z_S)\}$	(22)	$\{E(x_C, f(x_C))\}$
(23)	$\{f(x_C) \leqslant P\}$	(24)	$\{E(x_S, g(x_S))\}$	(25)	$\{g(x_S) \leqslant P\}$
(26)	$\{\neg E(x_A, y_A), \neg E(y_A, z_G)\}$				

For this example we have $\mathbf{L}^{\mathbf{C}}_{\lessdot} = \mathbf{L}^{\mathbf{M}}_{\lessdot} = \mathbf{L}^{\mathbf{U}}_{\lessdot} = \{(1)1 - (12)1, (23)1, (25)1\}$. This fact ($\mathbf{L}^{\mathbf{C}}_{\lessdot} = \mathbf{L}^{\mathbf{M}}_{\lessdot} = \mathbf{L}^{\mathbf{U}}_{\lessdot}$) is characteristic for a representation of the logics of [Wal87, SS89] in our logic. It may happen that Skolemization (see Example 4.2.3) results in conditioned declarations in our logic but in unconditioned declarations in the logic of [Wal87, SS89]. In order to overcome this, we have to optimize the CNF-algorithm 4.2.2 (especially Step 5) by checking the non-emptiness of the sorts involved in Skolemization. The refutation performs as follows (we present a linear proof found by the MKRP [OS91] theorem prover, which uses the logic of [Wal87]):

(27) $\{E(x_A, y_P), \neg M(y_A, x_A), \neg E(y_A, v_P), \neg E(y_A, z_G)\}$
Resolution between (26)1 and (13)4

(28) $\{E(x_A, y_P), \neg M(y_A, x_A), \neg E(y_A, z_G)\}$
Factorization between (27)3 and (27)4

(29) $\{\neg M(y_A, x_W), \neg E(y_A, z_G)\}$
Resolution between (28)1 and (19)1

(30) $\{\neg E(x_F, z_G)\}$
Resolution between (29)1 and (17)1

(31) $\{\neg M(z_A, x_F), \neg E(z_A, v_P), E(x_F, z_A)\}$
Resolution between (30)1 and (13)1

(32) $\{\neg E(z_B, v_P), E(x_F, z_B)\}$
Resolution between (31)1 and (16)1

(33) $\{\neg E(z_B, v_P), \neg E(x_B, z_G)\}$
Resolution between (32)2 and (26)1

(34) $\{\neg E(x_B, z_G)\}$
Factorization between (33)1 and (33)2

(35) $\{\neg M(z_A, x_B), \neg E(z_A, v_P), E(x_B, z_A)\}$
Resolution between (34)1 and (13)1

(36) $\{\neg E(y_S, v_P), E(x_B, y_S)\}$
Resolution between (35)1 and (15)1

(37) $\{E(x_B, x_S)\}$
Resolution between (36)1 and (24)1

(38) $\Box$
Resolution between (37)1 and (21)1

Unification for this example is straightforward, as we only have declarations occurring in unit clauses and all sorts are non-empty from the beginning.

8.2 The Lion and the Unicorn

The Lion and the Unicorn example can be seen as a descendant of Schubert's Steamroller. It is more difficult, because the sorts are not interpreted as finite sets in the initial model. Therefore the techniques developed in [AO83, MOW76] are not successful and the mechanism given in [BLS87] cannot be applied. We will discuss the problem at the end of this section in greater detail. Let us now just prove the theorem. Here is the clause set given after optimization (see Section 7.1.2):

(1)	$\{mon \lessdot Mo\}$	(2)	$\{tue \lessdot Tu\}$	(3)	$\{wed \lessdot We\}$
(4)	$\{thu \lessdot Th\}$	(5)	$\{fri \lessdot Fr\}$	(6)	$\{sat \lessdot Sa\}$
(7)	$\{sun \lessdot Su\}$	(8)	$\{x_{Mo} \lessdot LL\}$	(9)	$\{x_{Tu} \lessdot LL\}$
(10)	$\{x_{We} \lessdot LL\}$	(11)	$\{x_{Th} \not\lessdot LL\}$	(12)	$\{x_{Fr} \not\lessdot LL\}$
(13)	$\{x_{Sa} \not\lessdot LL\}$	(14)	$\{x_{Su} \not\lessdot LL\}$	(15)	$\{x_{Th} \lessdot LU\}$
(16)	$\{x_{Fr} \lessdot LU\}$	(17)	$\{x_{Sa} \lessdot LU\}$	(18)	$\{x_{Su} \not\lessdot LU\}$
(19)	$\{x_{Mo} \not\lessdot LU\}$	(20)	$\{x_{Tu} \not\lessdot LU\}$	(21)	$\{x_{We} \not\lessdot LU\}$
(22)	$\{x_{Mo} \lessdot D\}$	(23)	$\{x_{Tu} \lessdot D\}$	(24)	$\{x_{We} \lessdot D\}$
(25)	$\{x_{Th} \lessdot D\}$	(26)	$\{x_{Fr} \lessdot D\}$	(27)	$\{x_{Sa} \lessdot D\}$
(28)	$\{x_{Su} \lessdot D\}$	(29)	$\{x_{LL} \lessdot D\}$	(30)	$\{x_{LU} \lessdot D\}$
(31)	$\{lion \lessdot C\}$	(32)	$\{uni \lessdot C\}$	(33)	$\{f(x_{Mo}) \lessdot Su\}$
(34)	$\{f(x_{Tu}) \lessdot Mo\}$	(35)	$\{f(x_{We}) \lessdot Tu\}$	(36)	$\{f(x_{Th}) \lessdot We\}$
(37)	$\{f(x_{Fr}) \lessdot Th\}$	(38)	$\{f(x_{Sa}) \lessdot Fr\}$	(39)	$\{f(x_{Su}) \lessdot Sa\}$

(40) $\{x_D \lessdot LL, \neg Lies(lion, x_D, y_D), y_D \lessdot LL\}$
(41) $\{x_D \lessdot LL, Lies(lion, x_D, y_{LL})\}$
(42) $\{\neg Lies(lion, x_{LL}, y_{LL})\}$
(43) $\{Lies(lion, x_{LL}, y_D), y_D \lessdot LL\}$
(44) $\{x_D \lessdot LU, \neg Lies(uni, x_D, y_D), y_D \lessdot LU\}$
(45) $\{x_D \lessdot LU, Lies(uni, x_D, y_{LU})\}$
(46) $\{\neg Lies(uni, x_{LU}, y_{LU})\}$
(47) $\{Lies(uni, x_{LU}, y_D), y_D \lessdot LU\}$
(48) $\{\neg Lies(lion, x_D, f(x_D)), \neg Lies(uni, x_D, f(x_D))\}$

We choose $\mathbf{L}^{\mathbf{C}}_{\lessdot} = \{(1)1 - (10)1, (15)1 - (17)1, (22)1 - (39)1\}$ and refute the problem using an SOS strategy. The initial set of support consists of the theorem clause $\{\neg Lies(lion, x_D, f(x_D)),\ \neg Lies(uni, x_D, f(x_D))\}$ only. There are two resolution possibilities for the literal $\neg Lies(lion, x_D, f(x_D))$. We can resolve (48)1 and (43)1 or (48)1 and (41)1. We start examination of sorted unification between (48)1 and (43)1. The original unification problem is

$$\Gamma_0 \quad = \quad \{lion = lion, x_D = x_{LL}, y_D = f(x_D)\}$$

Applying the rules 5 (to the first equation) and 2 (using the second equation) we get

$$\Gamma_{01} \quad = \quad \{x_D = x_{LL}, y_D = f(x_{LL})\}$$

Here only rule 10 is applicable to the second equation which yields the following the seven problems

$$\begin{array}{lcl}
\Gamma_{011} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Mo}\} \\
\Gamma_{012} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Tu}\} \\
\Gamma_{013} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{We}\} \\
\Gamma_{014} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Th}\} \\
\Gamma_{015} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Fr}\} \\
\Gamma_{016} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Sa}\} \\
\Gamma_{017} & = & \{x_D = x_{LL}, y_D = f(x_{LL}), x_{LL} = x_{Su}\}
\end{array}$$

The problems Γ_{014}-Γ_{017} can be deleted by applying Algorithm 7.3.1 to the problems (see also Section 7.3). Applying rule 3 (using the third equation) once we get the three solved problems

$$\begin{array}{lcl}
\Gamma_{0111} & = & \{x_D = x_{Mo}, y_D = f(x_{Mo}), x_{LL} = x_{Mo}\} \\
\Gamma_{0121} & = & \{x_D = x_{Tu}, y_D = f(x_{Tu}), x_{LL} = x_{Tu}\} \\
\Gamma_{0131} & = & \{x_D = x_{We}, y_D = f(x_{We}), x_{LL} = x_{We}\}
\end{array}$$

These three problems yield the three resolvents

(49) $\{\neg Lies(uni, x_{Mo}, f(x_{Mo})), f(x_{Mo}) \lessdot LL\}$
(50) $\{\neg Lies(uni, x_{Tu}, f(x_{Tu})), f(x_{Tu}) \lessdot LL\}$
(51) $\{\neg Lies(uni, x_{We}, f(x_{We})), f(x_{We}) \lessdot LL\}$

The clauses (50) and (51) are subsumed by the clauses (8) and (9) respectively. Thus we only add (49) to the set of support. Building a resolvent between (48)1 and (41)1 is analogous to the previous step. Three clauses are generated, two can be subsumed and thus the new set of support contains

(49) $\{\neg Lies(uni, x_{Mo}, f(x_{Mo})), f(x_{Mo}) \lessdot LL\}$
(52) $\{\neg Lies(uni, x_{Th}, f(x_{Th})), x_{Th} \lessdot LL\}$

The first literal of clause (49) is pure with respect to $\mathbf{L}^{\mathbf{C}}_{\lessdot}$. The literal (49)1 is not pure with respect to $\mathbf{L}^{\mathbf{M}}_{\lessdot}$, because $(x_D \lessdot LU, \{Lies(uni, x_D, y_{LU})\}) \in \mathbf{L}^{\mathbf{M}}_{\lessdot}$. In section 7.4.2 we explained that we must not delete the clause but because no inference rule is applicable to the literal, we continue with clause (52). Again there are two candidates for the literal (52)1: (47)1 and (45)1. The unification problem resulting from (45)1 cannot be well sorted solved, because $f(x_{Th}) \notin \mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{\lessdot}, LU}$. Therefore the only possible step is to resolve (52)1 and (47)1 leading to

(55) $\{x_{Th} \lessdot LL, f(x_{Th}) \lessdot LU\}$

The only resolution possibility for (55)1 is (11)1 and we obtain

(56) $\{f(x_{Th}) \lessdot LU\}$

Finally for (56)1 the only applicable step is resolution with (21)1 and we result in the empty clause

(57) $\square$

We needed 6 resolution steps to refute the clause set and the initial branching rate of the search tree is 8 (with a SOS strategy). We constructed 9 clauses to explore the search space where 4 clauses are deleted by subsumption. In [WO90] we refuted the problem after 15 steps. Our new approach is faster because it uses sorted unification instead of adding negative declarations which are deleted by resolution.

The "Lion and the Unicorn" example is non trivial for unsorted theorem provers. An unsorted theorem prover needs 25 resolution steps to refute the clause set and the initial branching rate is 17 (with a SOS strategy). Especially if we increase the nesting depth of the f function in the theorem,

$$\{\neg Lies(lion, x_D, f(f(x_D))), \neg Lies(uni, x_D, f(f(x_D)))\}$$

the search space can become arbitrarily complex in unsorted logic whereas our sorted calculus has no problem, because the effect of the nesting depth on the complexity of the unification problem can be bound by some polynomial. For example, if we use the above theorem, the first unification problem is

$$\Gamma_0 \quad = \quad \{lion = lion, x_D = x_{LL}, y_D = f(f(x_D))\}$$

After eliminating the first equation with rule 5, application of rule 2 using the second equation and using the sorted rule 10 and rule 3 we again get seven problems:

$$\begin{array}{rcl}
\Gamma_{011} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{Mo} = f(x_{LL})\} \\
\Gamma_{012} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{Tu} = f(x_{LL})\} \\
\Gamma_{013} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{We} = f(x_{LL})\} \\
\Gamma_{014} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{Th} = f(x_{LL})\} \\
\Gamma_{015} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{Fr} = f(x_{LL})\} \\
\Gamma_{016} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{Sa} = f(x_{LL})\} \\
\Gamma_{017} & = & \{x_D = x_{LL}, y_D = f(f(x_{LL})), x_{Su} = f(x_{LL})\}
\end{array}$$

Again four problems can be deleted applying Algorithm 7.3.1, namely Γ_{013} to Γ_{016}. The other problems can be solved in the same way than before. Thus the additional nesting depth of the f does not cause additional unification problems. Therefore the effect of the nesting to the complexity of the unification problem can be bound polynomially.

In contrast the increased nesting of the f function causes an exploding search space for an unsorted theorem prover. We gave the examples to OTTER (version 2.0, binary resolution) and got the following statistics.

Nesting Depth	Clauses generated
1	64600
2	181486
3	1367151

That means OTTER generated 1367151 clauses in order to prove the theorem clause

$$\{\neg Lies(lion, x_D, f(f(f(x_D)))), \neg Lies(uni, x_D, f(f(f(x_D))))\}$$

and was not able to succeed with nesting depth 4 in reasonable time. We tested OTTER also with UR-resolution and gave the prover the clauses (1) to (7) as initial SOS. UR-resolution performed much better:

Nesting Depth	Clauses generated
1	1664
2	2943
3	4761
4	6971
5	9627
6	12675

8.3 Condensed Detachment

We solve the problem using a set of support strategy. The problem is not a hard problem, e.g. OTTER [McC90b, McC90a] can solve the problem in less than one second. But we will show that in the sorted formalization the problem can be solved without any search. Here is the final clause set from Section 7.1.1.

$$\begin{array}{ll}
(1) & \{i(x_P, y_\top))\not\leftarrow P, y_\top\leftarrow P\} \\
(2) & \{i(i(x_\top, y_\top), i(i(y_\top, z_\top), i(x_\top, z_\top)))\leftarrow P\} \\
(3) & \{i(i(n(x_\top), x_\top), x_\top)\leftarrow P\} \\
(4) & \{i(x_\top, i(n(x_\top), y_\top))\leftarrow P\} \\
(5) & \{i(a, a)\not\leftarrow P\}
\end{array}$$

The initial set of support consists of the clause $\{i(a, a)\not\leftarrow P\}$ only. The set $\mathbf{L}^{\mathbf{C}}_{\leftarrow} = \{i(i(x_\top, y_\top), i(i(y_\top, z_\top), i(x_\top, z_\top)))\leftarrow P,\ i(i(n(x_\top), x_\top), x_\top)\leftarrow P,\ i(x_\top, i(n(x_\top), y_\top))\leftarrow P\}$ and it contains no conditioned declarations. Therefore we skip the additional notations needed to represent conditioned objects. The only applicable step is resolution with (1)2, because unification with one of the literals (2)1, (3)1 or (4)1 results in a function symbol clash. The corresponding unification problem is

$$\Gamma_0 \quad = \quad \{y_\top = i(a, a)\}$$

which is still in solved form, as $i(a, a)\in\mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{\leftarrow},\top}$. Thus we add the new clause

$$(6) \quad \{i(x_P, i(a, a))\not\leftarrow P\}$$

to the set of support. The new clause cannot be resolved with clauses (2) and (4) because of a function clash and unification with the literal (3)1 results in the unification problem

$$\Gamma_1 \quad = \quad \{x_P = i(n(x_\top), x_\top), x_\top = i(a, a)\}$$

which can not be well sorted solved, because $i(n(i(a, a)), i(a, a))\notin\mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{\leftarrow},P}$. Again the only possible step is resolution with clause (1) yielding

(7) $\{i(x_P, i(x'_P, i(a,a)))\not< P\}$

Trying to build a resolvent between (7) and (3) or (4) again leads to unification problems which cannot be well sorted solved. Resolution with clause (2) gives the unification problem

$$\Gamma_2 \quad = \quad \{x_P = i(x_\top, y_\top), x'_P = i(y_\top, z_\top), x_\top = a, z_\top = a\}$$

Applying rule 2 of the unification algorithm using the equations $x_\top = a$ and $z_\top = a$ twice, we obtain

$$\Gamma_{21} \quad = \quad \{x_P = i(a, y_\top), x'_P = i(y_\top, a), x_\top = a, z_\top = a\}$$

We don't have $i(a, y_\top) \in \mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{<},P}$ or $i(y_\top, a) \in \mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{<},P}$. Application of rule 10 to the equation $x_P = i(a, y_\top)$ using the declaration $i(x_\top, i(n(x_\top), y_\top)) < P$ yields

$$\begin{aligned}\Gamma_{211} \quad = \quad & \{x_P = i(a, y_\top), x'_P = i(y_\top, a), x_\top = a, z_\top = a,\\ & y_\top = i(n(x'_\top), y'_\top), x'_\top = a\}\end{aligned}$$

Following rule 2 we use the new equations $y_\top = i(n(x'_\top), y'_\top)$ and $x'_\top = a$ as substitutions and result in

$$\begin{aligned}\Gamma_{2111} \quad = \quad & \{x_P = i(a, i(n(a), y'_\top)), x'_P = i(i(n(a), y'_\top), a),\\ & x_\top = a, z_\top = a, y_\top = i(n(a), y'_\top), x'_\top = a\}\end{aligned}$$

The first equation is now well sorted solved, i.e. $i(a, i(n(a), y'_\top)) \in \mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{<},P}$. But for the second equation we still have $i(i(n(a), y'_\top), a) \notin \mathbf{T}_{\mathbf{L}^{\mathbf{C}}_{<},P}$. Applying rule 10 using the declaration $i(i(n(x_\top), x_\top), x_\top) < P$ solves this last problem.

$$\begin{aligned}\Gamma_{21111} \quad = \quad & \{x_P = i(a, i(n(a), y'_\top)), x'_P = i(i(n(a), y'_\top), a),\\ & x_\top = a, z_\top = a, y_\top = i(n(a), y'_\top), x'_\top = a,\\ & i(i(n(a), y'_\top), a) = i(i(n(x''_\top), x''_\top), x''_\top)\}\end{aligned}$$

After applying the rules 5, 3 and 2 to the equation $i(i(n(a), y'_\top), a) = i(i(n(x''_\top), x''_\top), x''_\top)$ and it's descendants we finish in the solved problem

$$\begin{aligned}\Gamma_{211111} \quad = \quad & \{x_P = i(a, i(n(a), a)), x'_P = i(i(n(a), a), a),\\ & x_\top = a, z_\top = a, y_\top = i(n(a), a), x'_\top = a,\\ & y'_\top = a, x''_\top = a)\}\end{aligned}$$

and derive

(8) $\Box$

Note that the solution of the unification problem Γ_2 is unique and computationally simple. Comparing our solution to the proof found by OTTER (version 2.0, binary resolution), we needed only 3 steps to refute the clause set, while OTTER made 5 steps. The missing 2 steps occur in the solution of the unification problem Γ_2. The most important difference is that the sorted proof was found without any search, i.e. we did not generate useless resolvents. OTTER produced 60 clauses to find the empty clause. Here is OTTER's solution to the unsorted formalization:

(1) $\{\neg P(i(x,y)), \neg P(x), P(y)\}$

(2) $\{P(i(i(x,y), i(i(y,z), i(x,z))))\}$

(3) $\{P(i(i(n(x),x),x))\}$

(4) $\{P(i(x, i(n(x),y)))\}$

(5) $\{\neg P(i(a,a))\}$

(6) $\{\neg P(i(x, i(a,a))), \neg P(x)\}$
Resolution between (5)1 and (1)2

(7) $\{\neg P(x), \neg P(i(x', i(x, i(a,a)))), \neg P(x')\}$
Resolution between (6)1 and (1)2

(8) $\{\neg P(i(x,a)), \neg P(i(a,x))\}$
Resolution between (7)2 and (2)1

(9) $\{\neg P(i(a, i(n(a),a)))\}$
Resolution between (8)1 and (3)1

(10) $\square$
Resolution between (9)1 and (4)1

Chapter 9

Conclusion

We have presented a sound and complete calculus for the logic $\mathcal{L}_\mathcal{S}$. This logic $\mathcal{L}_\mathcal{S}$ is currently the most general sorted logic known as it allows sort declarations to be used arbitrarily as ordinary atoms. Nevertheless deductions in the calculus $\mathcal{CL}_\mathcal{S}$ are efficient and the calculus is a conservative extension of existing approaches.

All known sorted calculi [Wal87, SS89, Fri89, BHPS89, Coh87] allow only declarations of the kind $S \sqsubseteq T$ (expressing that S is a subsort of T) or $t\colon S$ (expressing that the term t has sort S). Translating these declarations into $\mathcal{L}_\mathcal{S}$ results in the unit clauses $x_S \lessdot T$ and $t \lessdot S$, respectively. Hence the sort information for these logics which is used in the unification algorithm corresponds to a set of unit clauses containing declarations in $\mathcal{L}_\mathcal{S}$. As there are only unit clauses after the translation we have $\mathbf{L}^{\mathbf{U}}_{\lessdot} = \mathbf{L}^{\mathbf{C}}_{\lessdot}$. Hence, all chosen declarations have an empty condition part. For declarations with empty condition part the definition of conditioned well sorted terms simplifies to the definition of well sorted terms (see Section 1.2.1 and Definition 6.1.2). Therefore the new notion of well sortedness is exactly the same notion as in the existing logics. Thus the unification algorithm using conditioned well sortedness collapses to the known unification algorithms using well sortedness and the new calculus has exactly the same behaviour as the simpler sorted calculi. Here we have the very nice feature that the calculus $\mathcal{CL}_\mathcal{S}$ for the general logic $\mathcal{L}_\mathcal{S}$ is as efficient as special calculi designed for sublanguages.

For clause sets containing declarations in non unit clauses the calculus must be compared to unsorted resolution as all approaches known so far can not exploit conditioned declarations for sorted reasoning. In Chapter 8 we showed that the $\mathcal{CL}_\mathcal{S}$ calculus improves efficiency significantly, if one place predicates (sorts) are involved in the reasoning process.

An open theoretical question is the incorporation of equality into $\mathcal{L}_\mathcal{S}$. It is well know that this is in general not without problems [SS89]. But we believe that our main idea, i.e. not to separate the information about sorts from the rest of the formulas is a good basis to overcome these problems.

Bibliography

[AB70] R. Anderson and W.W. Bledsoe. A linear format for resolution with merging and a new technique for establishing completeness. *Journal of the ACM*, 17:525–534, July 1970.

[AO83] G. Antoniou and H.J. Ohlbach. Terminator. In A. Bundy, editor, *Proc. of 8th International Joint Conference on Artificial Intelligence, IJCAI-83*, pages 916–919, 1983.

[BG90] L. Bachmair and H. Ganzinger. On restrictions of ordered paramodulation with simplification. In *10th International Conference on Automated Deduction*, pages 427–441. Springer Verlag, 1990.

[BHPS89] C. Beierle, U. Hedstueck, U. Pletat, and J. Siekmann. An order sorted predicate logic with closely coupled taxonomic information. LILOG-Report 86, IBM Germany, Scientific Center, 1989. To appear in Artificial Intelligence.

[BLS87] W. Bibel, R. Letz, and J. Schumann. Bottom-up enhancements of deductive systems. In *Conference on Artificial Intelligence and Information - Control Systems of Robots 1987*. North-Holland, 1987.

[BS85] R.J. Brachmann and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Bue91] H.J. Buerckert. *Constraint resolution.* Lecture Notes in Artificial Intelligence. Springer Verlag, 1991.

[CL73] C.L. Chang and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science and Applied Mathematics. Academic Press, 1973.

[Coh87] A. Cohn. A more expressive formulation of many sorted logic. *Journal of Automated Reasoning*, 3(2):113–200, 1987.

[Fri89] A.M. Frisch. A general framework for sorted deduction: Fundamental results on hybrid reasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 126–136, May 1989.

[Gal86] J.H. Gallier. *Logic for Computer Science*. Harper & Row, 1986.

[KN86] D. Kapur and P. Narendran. Np-completeness of the set unification and matching problems. In *8th International Conference on Automated Deduction*, pages 489–495. Springer Verlag, 1986.

[LO80] E. Lusk and R. Overbeek. Data structures and control architectures for the implementation of theorem proving programs. In *5th International Conference on Automated Deduction*, pages 232–249. Springer Verlag, 1980.

[Lov78] D. Loveland. *Automated Theorem Proving: A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, 1978.

[McC90a] W. McCune. Otter 2.0. In *10th International Conference on Automated Deduction*, pages 663–664. Springer Verlag, 1990.

[McC90b] W. McCune. Otter 2.0 users guide. Report ANL-90 9, Argonne National Laboratory, March 1990.

[Mer77] D. Meredith. In memoriam carew arthur meredith (1904-1976). *Notre Dame Journal of Formal Logic*, 18(513-516), 1977.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Programming Languages and Systems*, 4(2):258–282, 1982.

[MOW76] J.D. McCharen, R. Overbeek, and L. Wos. Complexity and related enhancements for automated theorem-proving programs. *Computers and Mathematics with Applications*, 2:1–16, 1976.

[MW92] W. McCune and L. Wos. Experiments in automated deduction with condensed detachment. In *11th International Conference on Automated Deduction*. Springer Verlag, 1992.

[Ohl90] H.J. Ohlbach. Abstraction tree indexing for terms. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 479–484. Pitman Publishing, London, August 1990.

[OS91] H.J. Ohlbach and J.H. Siekmann. The markgraf karl refutation procedure. In J.L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 41–112. MIT Press, 1991.

[OSS85] H.J. Ohlbach and M. Schmidt-Schauss. The lion and the unicorn. *Journal of Automated Reasoning*, 1(3):327–332, 1985.

[PW78] M. Paterson and M. Wegman. Linear unification. *Journal of the Computers and Systems*, 16, 1978.

[Rob65] J.A. Robinson. A machine -oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Sie89] J. Siekmann. Unification theory. *Journal of Symbolic Computation, Special Issue on Unification*, 7:207–274, 1989.

[Smu78] R. Smullyan. *What is the name of this book ?* Prentice-Hall, 1978.

[Soc90] R. Socher. *Simplification and Reduction for Automated Theorem Proving.* PhD thesis, Universitaet Kaiserslautern, 1990. Partially published in JAR.

[Soc91] R. Socher. Optimizing the clausal normal form transformation. *Journal of Automated Reasoning*, 7(3):325–336, 1991.

[SS89] M. Schmidt-Schauss. *Computational aspects of an order sorted logic with term declarations.* Lecture Notes in Artificial Intelligence. Springer Verlag, 1989.

[Sti85] M. Stickel. Theory resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.

[Sti86] M. Stickel. Schubert's steamroller problem: Formulations and solutions. *Journal of Automated Reasoning*, 2(1), 1986.

[Wal84] C. Walther. A mechanical solution of schubert's steamroller by many-sorted resolution. In *Proceedings of the 4th AAAI*, pages 330–334, 1984. revised version in Artificial Intelligence (26)2:217-224, 1985.

[Wal87] C. Walther. *A Many-sorted Calculus based on Resolution and Paramodulation.* Research Notes in Artificial Intelligence. Pitman Ltd., 1987.

[WO90] C. Weidenbach and H.J. Ohlbach. A resolution calculus with dynamic sort structures and partial functions. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 688–693. Pitman Publishing, London, August 1990.

[WRC65] L. Wos, G.A. Robinson, and D.F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12(4):536–541, 1965.