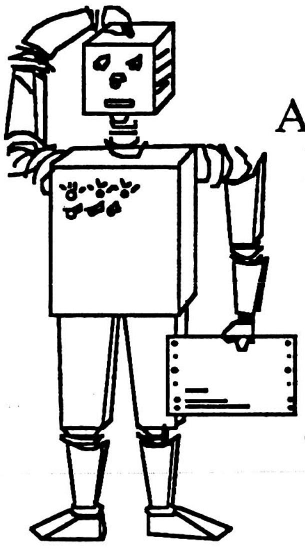
A Resolution Calculus with Dynamic Sort Structures and Partial Functions

Christoph Weidenbach

SEKI Report SR-89-23



A Resolution Calculus with Dynamic Sort Structures and Partial Functions

Christoph Weidenbach
FB Informatik
Universität Kaiserslautern
email: weidenb@informatik.uni-kl.de

Abstract: A many sorted logic is presented, which supports partial functions as well as potentially empty sorts and the dynamic derivation of sort information. In this so called DSPF-logic[1] (Dynamic Sorts and Partial Functions), the specification of the sort structure and the declarations for constant and function symbols are part of the formulae themselves. Therefore sort information needs not necessarily be known from the beginning but can be deduced from other information. Nevertheless the resolution calculus which is presented is almost as restrictive as in logics with static sort information.

[1] This work was supported by the "Sonderforschungsbereich 314", Artificial Intelligence, of the German Science Foundation (DFG)

1. Introduction

Many-sorted logics are becoming more and more important in Artificial Intelligence. The main reason for this development is the realization that a special treatment of taxonomic information leads to a more natural encoding of this knowledge and therefore to more natural deductions. In these logics the taxonomic information is represented by sorts and declarations. Sorts denote sets of objects with similar properties and declarations describe the relations between objects and sorts as well as the domain-range relation of functions.

The first logics which solved the problem of dealing with taxonomic information with a fully developed resolution and paramodulation calculus, e.g. [Walther 87, Schmidt-Schauß 89] integrate the sort information into the signature, thus fully separate it from the rest of the information. This separation into taxonomic and other knowledge leads to more compact formulae and, using sorted unification algorithms, to a smaller search space. Since, however, some sort declarations result in undecidable unification problems, not every sort information can be represented this way. Furthermore it is not possible to handle "dynamic" taxonomic information, i.e. information which is not known from the beginning, but has to be deduced. It turned out that in some application areas of order sorted logic, for example natural language processing, the taxonomic information is not completely known from the beginning and for this reason the static approach is not adequate. To overcome this problem new logics like Cohn's LLAMA [Cohn 87] or the logic used in the LILOG project [Beierle *et al.* 89], have been invented. In addition to the sort part in the signature, these logics allow sort predicates to occur in the formulae. Special deduction rules guarantee consistency between the statically represented and dynamically derived taxonomic information. But it is still not clear, how the dynamically represented sort information can be used to reduce the search space.

In our approach we pull down the barrier between static and dynamic taxonomic knowledge and put the whole information into the fromulae by using the special relations $\in$ (membership) and $\sqsubseteq$ (subset), without loosing the advantages of sort information during deduction.

In order to give a flavor of DSPF-logic and to distinguish it from unsorted logic as well as from other sorted logics, consider the following small example. We have rational and real numbers and all rationals are reals. A function 'div' maps two rational numbers to a rational number, provided that it's second argument is not zero. Furthermore there are two constants 'a' and 'b' of type rational and 'b' is not zero.

An unsorted formulation looks as follows:

1) $\forall x\ \mathrm{Rat}(x) \Rightarrow \mathrm{Real}(x)$ 2) $\forall x,y\ \mathrm{Rat}(x) \land \mathrm{Rat}(y) \land y \neq 0 \Rightarrow \mathrm{Rat}(\mathrm{div}(x,y))$

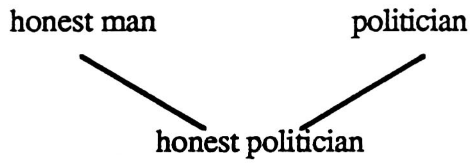
3) $\mathrm{Rat}(a)$ 4) $\mathrm{Rat}(b)$ 5) $b \neq 0$

From this axioms we can deduce the infinitely many facts Rat(div(a,b)), Rat(div(div(a,b),b)),...,Real(div(a,b)),... In most sorted logics, a sorted axiomatization of this situation is not possible at all, because the second axiom is a conditioned function declaration. In DSPF-logic, the corresponding axiomatization is:

1) $\forall x_{Rat}\ x_{Rat} \in Real$ (another formulation for $Rat \sqsubseteq Real$)
2) $\forall x_{Rat}, y_{Rat}\ y_{Rat} \neq 0 \Rightarrow div(x_{Rat}, y_{Rat}) \in Rat$
3) $a \in Rat$ 4) $b \in Rat$ 5) $b \neq 0$

Here only a single deduction is possible, namely 6) $\forall x_{Rat}\ div(x_{Rat}, b) \in Rat$, which seems reasonable. Nevertheless we can prove all theorems in the DSPF-logic, for example $div(a,b) \in Real$. With 1) and the negated theorem $div(a,b) \notin Real$ we obtain 7) $div(a,b) \notin Rat$, where unification of x_{Rat} and $div(a,b)$ generates the condition $div(a,b) \in Real$ whose negation becomes a residue literal in the result. 7) and 6) now yields $a \notin Rat$ and finally using 3) we derive the empty clause. This example shows that the application of sort information is still driven by unification and therefore very tightly controlled.

The unification mechanism which generates conditions of the form *term∈sort*, supports the incorporation of two more concepts into DSPF-logic, namely sorts denoting possibly empty sets and "dynamic" partial functions. For example the following sort hierarchy can be declared



without knowing that there is a honest politician at all. Since quantification over an empty set yields a true statement, the usual instantiation rule is not sound. For example from

$$\forall x_{honest\text{-}politician}\ loved(x_{honest\text{-}politician}, people)$$

we can not deduce loved(Gorbi,people), because if there is no honest politician at all, the quantified statement is true, but nevertheless the instantiated formula is false. In DSPF-logic we would generate the conditioned instance

$$Gorbi \in honest\text{-}politician \Rightarrow loved(Gorbi, people)$$

which is only satisfiable if there is at least one honest politician, namely Gorbi. With exactly the same mechanism we can handle partial functions. For example an instantiation of the above formula with leader(GDR) yields

$$leader(GDR) \in honest\text{-}politician \Rightarrow loved(leader(GDR), people)$$

If the function leader would not be defined for 'GDR', the condition would be false and therefore the implication would be true.

The literals with the negated ∈-predicate can be seen as an invitation to prove a membership relation. For example $a \notin S$ can be proved with a corresponding $a \in S$ literal. For the special sort Ω which denotes all objects, however, a literal $f(c) \notin \Omega$, which is true when f is not defined for c, can also be proved using an arbitrary positive literal P(...f(c)...), where f(c) occurs as a subterm. The reason is that positive literals can only be satisfied when all their arguments are defined. Thus, $f(c) \notin \Omega$ (f(c) is not defined) and

P(...f(c)...) (f(c) is defined) are contradictionary and can trigger a resolution operation. A literal of the form $c \notin \Omega$ is contradictionary, because constants denote always existing objects.

Unification with conditioned instantiation and the special resolution rule between *term*$\notin \Omega$-literals and positive literals are the only extensions of the resolution calculus in DSPF-logic. In the subsequent chapters the logic and the calculus will be presented in more detail.

2 Syntax

The syntax of the DSPF-Logic is very similar to the syntax of unsorted first order logic, because terms, atoms and formulae are generated without considering sorts. Only the two special relations $\in$ and $\subseteq$ are added and variables are indexed with the sort they range over.

2.1 Signature

2.1.1 Definition: Signature

A signature $\Sigma := (\mathbf{S}, \mathbf{V}, \mathbf{F}, \mathbf{P})$ consists of the following disjoint sets: $\mathbf{S}$ is a finite set of sort symbols, the fixed symbol Ω for "any" is always in $\mathbf{S}$. Sort symbols are S, T. $\mathbf{V}$ is a countably infinite set of variable symbols. For variables we write x, y, z. $\mathbf{F}$ is a countably infinite set of function symbols, $\mathbf{F}$ is divided into the sets of n-place function symbols $\mathbf{F}_n$.We use the symbols f, g, h for functions and a, b, c for constants (0-place function symbols). $\mathbf{P}$ is a finite set of predicate symbols divided into the sets of n-place predicate symbols $\mathbf{P}_n$, $n \in \mathbb{N}_0$. Predicates are named by P, Q. Additionally Σ consists of a function $S{:}\mathbf{V} \to \mathbf{S}$ such that for every sort $T \in \mathbf{S}$, there exist countably infinitely many variables $x \in \mathbf{V}$, with $S(x) = T$. We say x has sort T in this case and denote this by x_T.

2.1.2 Definition: Special Symbols

The following symbols are available: the logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists$, the auxiliary symbols "(", ")", "," and the special relations $\subseteq$ (subset), $\in$ (is-element), which are used to represent information about sorts. At the meta level, we use italic versions of the subset and is-element symbol.

2.1.3 Assumption: Non-empty Universe

We always assume that there is at least one constant symbol in Σ. This assumption implies that the universe of discourse is not empty and this appears to be natural for us.

2.2 Terms, Atoms, and Formulae

2.2.1 Definition: Terms

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

i) $x \in \mathbf{T}_\Sigma$, if $x \in \mathbf{V}$

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

Let V(t) denote the 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 with $V(t) = \emptyset$ is called <u>ground</u>.

2.2.2 Definition: Atoms

If $P \in \mathbf{P}_n$ and $t_1,\ldots,t_n$ are terms then $P(t_1,\ldots,t_n)$ is an <u>atom</u>. If S and T are sort symbols and t is a term, then $t \in S$ and $S \subseteq T$ are <u>atoms</u>.

2.2.3 Definition: 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.

2.2.4 Definition: 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 variables occurring in it. We use two notations for clauses, the logical notation, e.g. $C = L_1 \vee L_2$ and the set oriented notation $C = \{L_1,L_2\}$. The empty clause is denoted by $\{\}$.

2.2.5 Definition: Formulae

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

2.2.6 Definition: Substitutions

A substitution σ is a total function $\sigma: V \to T_\Sigma$, such that the set $\{x \in V \mid \sigma(x) \neq x\}$ is finite. Let $DOM(\sigma) := \{x \in V \mid \sigma(x) \neq x\}$. 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 $\{x_1 \mapsto t_1,\dots,x_n \mapsto t_n\}$, where $DOM(\sigma) = \{x_1,\dots,x_n\}$.We can extend the application of σ to T_Σ by $\sigma(t) := \sigma(t)$, if $t \in V$ and $\sigma(t) := f(\sigma(t_1),\dots,\sigma(t_n))$, if $t = f(t_1,\dots,t_n)$. The same way the application of σ can be extended to other objects, e.g. literals, clauses or sets of such objects. Let $COD(\sigma) := \sigma(DOM(\sigma))$ and $I(\sigma) := V(COD(\sigma))$. A substitution σ is called ground if $I(\sigma) = \emptyset$.

The composition of two substitutions $\sigma = \{x_1 \mapsto t_1,\dots,x_n \mapsto t_n\}$ and $\tau = \{y_1 \mapsto s_1,\dots,y_m \mapsto s_m\}$ can be computed by $\sigma\tau = \{y_1 \mapsto \sigma(s_1),\dots,y_m \mapsto \sigma(s_m)\} \cup \{x_i \mapsto t_i \mid x_i \in (DOM(\sigma) - DOM(\tau))\}$. A substitution σ is called idempotent if $\sigma\sigma = \sigma$. Note that a substitution σ is idempotent iff $DOM(\sigma) \cap I(\sigma) = \emptyset$ [Herold 83]. With $\sigma\backslash x$ we denote a substitution that is equal to σ, except that it maps x to x.

2.2.7 Definition: Subformula, Scope, Positive Component

Let $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$ be formulae. $\mathcal{F}$ 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 $\wedge$, a disjunction $\vee$, a negation $\neg$, an implication $\Rightarrow$, an equivalence $\Leftrightarrow$) iff $\mathcal{F}$ is a subformula of $\mathcal{G}$ (or $\mathcal{H}$) in $\forall x_S\ \mathcal{G}$ ($\exists x_S\ \mathcal{G}$, $\mathcal{G} \wedge \mathcal{H}$, $\mathcal{G} \vee \mathcal{H}$, $\neg\mathcal{G}$, $\mathcal{G} \Rightarrow \mathcal{H}$, $\mathcal{G} \Leftrightarrow \mathcal{H}$). An occurrence of a subformula $\mathcal{F}$ of 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.

Note that $\mathcal{F}$ is not a positive component in $\neg(\neg\mathcal{F})$. This does not seem to be natural, but we only use this definition for formulae in negation normal form, where all negation symbols stand directly in front of the predicate symbols.

2.2.8 Definition: free, bound variables

For a formula $\mathcal{F}$, the set $FV(\mathcal{F})$ of free variables of $\mathcal{F}$ is defined by:

i) $FV(P(t_1,\ldots,t_n)) = V(t_1) \cup \ldots \cup V(t_n)$ for a n-place predicate P
ii) $FV(t \in S) = V(t)$ and $FV(S \sqsubseteq T) = \emptyset$ for the special relations $\in$ and $\sqsubseteq$.
iii) $FV(\neg\mathcal{F}) = FV(\mathcal{F})$
iv) $FV(\mathcal{F} * \mathcal{G}) = FV(\mathcal{F}) \cup FV(\mathcal{G})$, where $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
v) $FV(\forall x_T\, \mathcal{F}) = FV(\mathcal{F}) - \{x_T\}$
vi) $FV(\exists x_T\, \mathcal{F}) = FV(\mathcal{F}) - \{x_T\}$

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

i) $BV(P(t_1,\ldots,t_n)) = \emptyset$
ii) $BV(t \in S) = \emptyset$ and $BV(S \sqsubseteq T) = \emptyset$ for the special relations $\in$ and $\sqsubseteq$.
iii) $BV(\neg\mathcal{F}) = BV(\mathcal{F})$
iv) $BV(\mathcal{F} * \mathcal{G}) = BV(\mathcal{F}) \cup BV(\mathcal{G})$, where $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
v) $BV(\forall x_T\, \mathcal{F}) = BV(\mathcal{F}) \cup \{x_T\}$
vi) $BV(\exists x_T\, \mathcal{F}) = BV(\mathcal{F}) \cup \{x_T\}$

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 $x_T \in BV(\mathcal{F})$. This definition is adopted from [Gallier 86].

2.2.9 Assumption: Considering only Sentences

We only consider formulae without free variables. Such formulae are called sentences.

3 Semantics

In order to define the semantics for the DSPF-Logic, we have to assign a set of objects of the non-empty universe to every sort symbol, a partial function to every function symbol and we interpret predicates, the special relations and the logical connectives in the usual way. To get around the problems of partial functions, we use the often applied trick to introduce a new object "$\perp$", which denotes "undefined" and extend the functions using this object such that they become strict and total. If $\perp$ occurs at an argument position of an atom, the atom becomes false, i.e. an atom can never hold if an ill sorted term occurs in it. On the other hand the negation of an atom including an ill-sorted term always becomes true. Thus our semantics of the negation is like "If the negation of a statement is true, the statement is wrong or the statement does not make sense because it is ill-sorted".

3.1 Algebras

3.1.1 Definition: Σ-quasi-algebra

Let Σ be a signature. A <u>Σ-quasi-algebra</u> $\mathcal{A}$ consists of a non-empty carrier set A, a partial function $f_{\mathcal{A}}:A^n \rightarrow A$ with domain $\mathcal{D}(f_{\mathcal{A}})$ for every function symbol $f \in F_n$, a set $S_{\mathcal{A}} \subseteq A$ for every sort S, such that $A = \bigcup_{S \in \mathbf{S}} S_{\mathcal{A}}$.

3.1.2 Definition: Σ-algebra

Let Σ be a signature. Then a Σ-algebra $\mathcal{A}$ is defined as a Σ-quasi-algebra $\mathcal{A}$ with the following additions:

i) The carrier set $A^{\perp} := A \cup \{\perp\}$, where $\perp \notin A$.
ii) $\Omega_{\mathcal{A}} = A$
iii) For all constants $c \in F_0$, $c_{\mathcal{A}} \in A$.
iv) For all $f \in F_n$ $(n>0)$ and all $(a_1,\ldots, a_n) \in (A^{\perp})^n$, if $(a_1,\ldots,a_n) \notin \mathcal{D}(f_{\mathcal{A}})$ we extend $f_{\mathcal{A}}$ to $f_{\mathcal{A}}(a_1,\ldots,a_n) := \perp$. That means $f_{\mathcal{A}}$ is assumed to be a strict ω-extension.

3.1.3 Definition: Σ-assignment

Let $\mathcal{A}$ be a Σ-algebra. A <u>Σ-assignment</u> is a total mapping $\varphi:V \rightarrow A^{\perp}$ with $\varphi(x) \in S(x)_{\mathcal{A}}$ if $S(x)_{\mathcal{A}} \neq \emptyset$ and $\varphi(x) = \perp$ otherwise. The <u>homomorphic extension</u> $\varphi_h:T_{\Sigma} \rightarrow A^{\perp}$ of a Σ-assignment φ is defined as follows:

$$\varphi_h(x) := \varphi(x) \text{ for all } \Sigma\text{-variables } x$$
$$\varphi_h(f(t_1,\ldots,t_n)) := f_{\mathcal{A}}(\varphi_h(t_1),\ldots,\varphi_h(t_n)), \text{ for all } f \in F_n.$$

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

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

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

3.1.4 Auxiliary definitions

For $\forall x_{S_1} \dots \forall x_{S_n}$ we use the abbreviation $\forall \overline{x_S}$ and for $y_1, \dots, y_n$ we write $\overline{y}$. If we write $\mathcal{F}[x]$ for a formula $\mathcal{F}$ and a variable x, we want to emphasize that x occurs free in $\mathcal{F}$.

3.2 Structures and Interpretations

3.2.1 Definition: Σ-structure

Let Σ be a signature. A Σ-structure M is a Σ-algebra $\mathcal{A}$ which has additional denotations P_M for every predicate symbol $P \in \mathbf{P}_n$ and the special symbols $\in$ and $\subseteq$, such that:

i) P_M is a relation with $P_M \subseteq A^n$
ii) $\in_M$ is the element-of relation, i.e. $\in_M = \in$
iii) $\subseteq_M$ is the subset relation, i.e. $\subseteq_M = \subseteq$

3.2.2 Definition: Interpretation

Let M be a Σ-structure and $\varphi: T_\Sigma \to A^\perp$ a Σ-assignment. An interpretation is a pair $\mathfrak{I} = (M, \varphi)$ such that for every s, $t \in T_\Sigma$, S, $T \in \mathbf{S}$ and $P \in \mathbf{P}_n$

$\mathfrak{I}(t) = \varphi(t)$	
$\mathfrak{I} \models P(t_1, \dots, t_n)$	iff $(\varphi(t_1), \dots, \varphi(t_n)) \in P_M$.
$\mathfrak{I} \models s \in T$	iff $\varphi(s) \in T_{\mathcal{A}}$.
$\mathfrak{I} \models S \subseteq T$	iff $S_{\mathcal{A}} \subseteq T_{\mathcal{A}}$.
$\mathfrak{I} \models \neg \mathcal{F}$	iff not $\mathfrak{I} \models \mathcal{F}$
$\mathfrak{I} \models \mathcal{F} \wedge \mathcal{G}$	iff $\mathfrak{I} \models \mathcal{F}$ and $\mathfrak{I} \models \mathcal{G}$
$\mathfrak{I} \models \mathcal{F} \vee \mathcal{G}$	iff $\mathfrak{I} \models \mathcal{F}$ or $\mathfrak{I} \models \mathcal{G}$
$\mathfrak{I} \models \mathcal{F} \Rightarrow \mathcal{G}$	iff not $\mathfrak{I} \models \mathcal{F}$ or $\mathfrak{I} \models \mathcal{G}$
$\mathfrak{I} \models \mathcal{F} \Leftrightarrow \mathcal{G}$	iff $\mathfrak{I} \models (\mathcal{F} \Rightarrow \mathcal{G}) \wedge (\mathcal{G} \Rightarrow \mathcal{F})$
$\mathfrak{I} \models \forall x_S\ \mathcal{F}$	iff for all $a \in S_{\mathcal{A}}$, $(M, \varphi[\{x_S/a\}]) \models \mathcal{F}$
$\mathfrak{I} \models \exists x_S\ \mathcal{F}$	iff there exists an $a \in S_{\mathcal{A}}$, such that $(M, \varphi[\{x_S/a\}]) \models \mathcal{F}$.

3.2.3 Definition: Σ-model, satisfiable, unsatisfiable

Let $\mathcal{F}$ be a formula. An interpretation $\mathcal{M} = (M, \varphi)$ is a Σ-model for $\mathcal{F}$ if $\mathcal{M} \models \mathcal{F}$. Note that φ plays no role for sentences. A set F of formulae has a Σ-model $\mathcal{M}$ if for every formula $\mathcal{F} \in F$ $\mathcal{M} \models \mathcal{F}$. A formula is called satisfiable if it has a Σ-model. We call a formula unsatisfiable, if it has no Σ-model.

Our interpretation of atoms including ill-sorted terms leads to a semantics which is not symmetric with respect to the sign of predicates. If $\mathcal{F}$ is a satisfiable formula and we replace every occurrence of a predicate P in $\mathcal{F}$ by it's negation, the resulting formula $\mathcal{F}'$ may be unsatisfiable. For example the formula $\mathcal{F} = (\neg P(f(a)) \wedge f(a) \notin \Omega)$ is satisfiable in every interpretation with $\varphi(f(a)) = \perp$. But $\mathcal{F}' = (P(f(a)) \wedge f(a) \notin \Omega)$ is unsatisfiable because $f(a) \notin \Omega$ forces f(a) to be undefined and P(f(a)) is false if f(a) is undefined.

One can argue whether the negation of something undefined should yield a true statement. Our reason for choosing this meaning was the realization that it leads to a resolution calculus very near to the unsorted resolution calculus, allows an easy transformation to unsorted first order logic and together with the sort mechanism gives enough expressivity to avoid critical situations in practical applications.

3.3 Examples

3.3.1 Schubert's Steamroller

We shall use Schubert's Steamroller as an example throughout this paper. In 1978 Schubert raised the following challange problem:

> 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.

Let Σ be a signature including the following symbols: A, B, C, G, F, P, S, W are all sorts and the intended meaning for them is animals, birds, caterpillars, grains, foxes, plants, snails and wolves, respectively. We use two 2-place predicate symbols E(x,y) to express x likes to eat y and M(x,y) to denote that x is much smaller than y. The constants tweety, swallowtail, muesli, foxy, slimey and lupo are also available. Then the problem can be formalized as follows (for a discussion of the formalization and the two different versions of the theorem see [Stickel 86]):

1) $lupo \in W \wedge foxy \in F \wedge tweety \in B \wedge swallowtail \in C \wedge slimey \in S \wedge muesli \in G$
2) $W \subseteq A \wedge F \subseteq A \wedge B \subseteq A \wedge C \subseteq A \wedge S \subseteq A \wedge G \subseteq P$
3) $\forall x_A \forall x_P (E(x_A,x_P) \vee \forall y_A (M(y_A,x_A) \wedge \exists y_P E(y_A,y_P) \Rightarrow E(x_A,y_A)))$
4) $\forall x_C \forall x_B M(x_C,x_B)$ 5) $\forall x_S \forall x_B M(x_S,x_B)$
6) $\forall x_B \forall x_F M(x_B,x_F)$ 7) $\forall x_F \forall x_W M(x_F,x_W)$
8) $\forall x_W \forall x_F \neg E(x_W,x_F)$ 9) $\forall x_W \forall x_G \neg E(x_W,x_G)$
10) $\forall x_B \forall x_C E(x_B,x_C)$ 11) $\forall x_B \forall x_S \neg E(x_B,x_S)$
12) $\forall x_C \exists x_P E(x_C,x_P)$ 13) $\forall x_S \exists x_P E(x_S,x_P)$
14) $\exists x_A \exists y_A (E(x_A,y_A) \wedge \exists x_G E(y_A,x_G))$ Theorem Version 1
15) $\exists x_A \exists y_A (E(x_A,y_A) \wedge \forall x_G E(y_A,x_G))$ Theorem Version 2

3.3.2 The Lion and the Unicorn

Our second well known example is a puzzle named "The Lion and the Unicorn", that can be found in [Smullyan 78]. 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 [Ohlbach&Schmidt-Schauß 85]. 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 UL 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 monday, tuesday, wednesday, thursday, friday, saturday and sunday, the constants lion and unicorn and a 1-place function yesterday are needed. Furthermore we need a 3-place predicate LA(x,y,z) which is true if x says at day y that he lies at day z. Now we can give a complete set of formulae for this problem:

1) $\text{monday} \in MO \land \text{tuesday} \in TU \land \text{wednesday} \in WE \land \text{thursday} \in TH \land \text{friday} \in FR \land \text{saturday} \in SA \land \text{sunday} \in SU$
2) $MO \subseteq LL \land TU \subseteq LL \land WE \subseteq LL \land \forall x_{TH}\ x_{TH} \notin LL \land \forall x_{FR}\ x_{FR} \notin LL \land \forall x_{SA}\ x_{SA} \notin LL \land \forall x_{SU}\ x_{SU} \notin LL$
3) $TH \subseteq UL \land FR \subseteq UL \land SA \subseteq UL \land \forall x_{SU}\ x_{SU} \notin UL \land \forall x_{MO}\ x_{MO} \notin UL \land \forall x_{TU}\ x_{TU} \notin UL \land \forall x_{WE}\ x_{WE} \notin UL$
4) $MO \subseteq D \land TU \subseteq D \land WE \subseteq D \land TH \subseteq D \land FR \subseteq D \land SA \subseteq D \land SU \subseteq D \land LL \subseteq D \land UL \subseteq D$
5) $\text{lion} \in C \land \text{unicorn} \in C$
6) $\text{yesterday}(\text{monday}) \in SU \land \text{yesterday}(\text{tuesday}) \in MO$
7) $\text{yesterday}(\text{wednesday}) \in TU \land \text{yesterday}(\text{thursday}) \in WE$
8) $\text{yesterday}(\text{friday}) \in TH \land \text{yesterday}(\text{saturday}) \in FR \land \text{yesterday}(\text{sunday}) \in SA$
9) $\forall x_D\ \forall y_D\ (x_D \notin LL \land LA(\text{lion},x_D,y_D) \Rightarrow y_D \in LL)$
10) $\forall x_D\ \forall y_D\ (x_D \notin LL \land \neg LA(\text{lion},x_D,y_D) \Rightarrow y_D \notin LL)$
11) $\forall x_D\ \forall y_D\ (x_D \in LL \land LA(\text{lion},x_D,y_D) \Rightarrow y_D \notin LL)$
12) $\forall x_D\ \forall y_D\ (x_D \in LL \land \neg LA(\text{lion},x_D,y_D) \Rightarrow y_D \in LL)$
13) $\forall x_D\ \forall y_D\ (x_D \notin UL \land LA(\text{unicorn},x_D,y_D) \Rightarrow y_D \in UL)$
14) $\forall x_D\ \forall y_D\ (x_D \notin UL \land \neg LA(\text{unicorn},x_D,y_D) \Rightarrow y_D \notin UL)$
15) $\forall x_D\ \forall y_D\ (x_D \in UL \land LA(\text{unicorn},x_D,y_D) \Rightarrow y_D \notin UL)$
16) $\forall x_D\ \forall y_D\ (x_D \in UL \land \neg LA(\text{unicorn},x_D,y_D) \Rightarrow y_D \in UL)$
17) $\exists x_D\ (LA(\text{lion},x_D,\text{yesterday}(x_D)) \land LA(\text{unicorn},x_D,\text{yesterday}(x_D)))$ Theorem

Comparing the formalization given so far with the formalization in [Ohlbach&Schmidt-Schauß 85], we have replaced the MEMBER predicate by the $\in$ relation and we have tried to express as much as possible in terms of sorts. To this end we have doubled the axioms 9 - 12 and eliminated the relation lying-days. The reason is that in the DSPF-Logic presented in this paper, it is not possible to use a formula like

$$\forall x_D\ \forall y_D\ \forall x_C\ (x_D \notin \text{lying-days}(x_C) \land LA(x_C,x_D,y_D) \Rightarrow y_D \in \text{lying-days}(x_C))$$

because only sort constants are allowed on the right hand side of the $\in$ relation. Although the axioms 9 - 16 are necessary to define the LA predicate, they can also be interpreted as a dynamic description of the sorts UL (the unicorn's lying days) and LL (the lion's lying days) with respect to the statements of the two creatures. In the resolution proof a dynamically derived inconsistency for the searched day will be obtained using the sort information above.

3.4 Relativization

In this paragraph we show how to transform formulae of the DSPF-Logic in unsorted first order logic. This is useful because if we can prove that the semantic and syntactic notations of both logics are equivalent, results like for example the compactness theorem, from the well known first order logic can be applied to our approach. Therefore we prove the model theoretic part of the sort theorem, which states the equivalence of the logics. The transformation of the formulae will be done in two steps. First, the sorted formulae are translated in a form, where they only consist of variables of sort Ω and then use the standard method [Oberschelp 62, Walther 87] to provide a unary predicate for every sort symbol to reach the final unsorted form.

3.4.1 Lemma: Simple Properties

First we list some equivalences, which can be used to eliminate variables of a sort not equal to Ω and substitute the $\subseteq$ relation by the $\in$ relation.

$$\begin{array}{ll}
\mathfrak{I} \models \forall x_S\ \mathcal{F} & \text{iff } \mathfrak{I} \models \forall x_\Omega\ (x_\Omega \in S \Rightarrow \{x_S \mapsto x_\Omega\}\mathcal{F})\ x_\Omega \text{ not free in } \mathcal{F} \\
\mathfrak{I} \models \exists x_S\ \mathcal{F} & \text{iff } \mathfrak{I} \models \exists x_\Omega\ (x_\Omega \in S \wedge \{x_S \mapsto x_\Omega\}\mathcal{F})\ x_\Omega \text{ not free in } \mathcal{F} \\
\mathfrak{I} \models S \subseteq T & \text{iff } \mathfrak{I} \models \forall x_\Omega\ (x_\Omega \in S \Rightarrow x_\Omega \in T) \\
 & (\text{iff} \quad \mathfrak{I} \models \forall x_S\ x_S \in T)
\end{array}$$

Proof:

Can be easily seen by using the semantics for the special relations.■

The first two equivalences are correct regardless whether x_S occurs free within $\mathcal{F}$ or not. The first equivalence interpreted as a rewrite rule, gives the possibility to eliminate universally quantified variables, which do not occur free in the formula as for example $\forall x_S$ $P(a) \rightarrow \forall x_\Omega\ (x_\Omega \in S \Rightarrow P(a))$. Using a semantics which allows empty sorts, quantifiers, like $\forall x_S$ in the above example, can't be dropped. But to produce clause normal form, we have to eliminate the universal quantifiers and therefore to apply this equivalence as a rule.

If the three equivalences are applied from left to right to a formula, the resulting formula is very near to unsorted first order logic. This is because the variables range over the whole universe (the whole universe is always assigned to sort Ω) and there is only one special relation left.

3.4.2 Unsorted First Order Logic

We shall give a brief introduction to the syntax and semantics of unsorted first order logic. The first order signature $\Sigma := (\mathbb{V}, \mathbb{F}, \mathbb{P})$ consists of the following disjoint sets: $\mathbb{V}$ is a countably infinite set of variable symbols, $\mathbb{F}$ is a countably infinite set of function symbols and $\mathbb{P}$ a finite set of predicate symbols. Terms, atoms, literals and formulae are built in the same way as in the DSPF-Logic, except that the special relations $\in$ and $\subseteq$ don't exist. An interpretation $\mathfrak{I} = (\mathbb{M},\varphi)$ consists of a Σ-structure $\mathbb{M}$ and a Σ-assignment $\varphi:\mathbb{V}\to\mathbb{U}$ which can be extended as in Definition 3.1.3. The Σ-structure $\mathbb{M}$ consists of a non-empty universe $\mathbb{U}$, a total function $f_{\mathbb{U}}:\mathbb{U}^n\to\mathbb{U}$ for every n-place function symbol $f\in\mathbb{F}_n$ and a n-place relation $P_{\mathbb{M}}\subseteq\mathbb{U}^n$ for every predicate symbol $P\in\mathbb{P}_n$. For every $t\in T_\Sigma$ and $P\in\mathbb{P}_n$

$$\begin{array}{ll}
\mathfrak{I}(t) = \varphi(t) & \\
\mathfrak{I} \models P(t_1,\dots,t_n) & \text{iff } (\varphi(t_1),\dots,\varphi(t_n))\in P_{\mathbb{M}}. \\
\mathfrak{I} \models \neg\mathcal{F} & \text{iff not } \mathfrak{I}\models\mathcal{F} \\
\mathfrak{I} \models \mathcal{F}\wedge\mathcal{G} & \text{iff } \mathfrak{I}\models\mathcal{F} \text{ and } \mathfrak{I}\models\mathcal{G} \\
\mathfrak{I} \models \mathcal{F}\vee\mathcal{G} & \text{iff } \mathfrak{I}\models\mathcal{F} \text{ or } \mathfrak{I}\models\mathcal{G} \\
\mathfrak{I} \models \mathcal{F}\Rightarrow\mathcal{G} & \text{iff not } \mathfrak{I}\models\mathcal{F} \text{ or } \mathfrak{I}\models\mathcal{G} \\
\mathfrak{I} \models \mathcal{F}\Leftrightarrow\mathcal{G} & \text{iff } \mathfrak{I}\models(\mathcal{F}\Rightarrow\mathcal{G})\wedge(\mathcal{G}\Rightarrow\mathcal{F}) \\
\mathfrak{I} \models \forall x\,\mathcal{F} & \text{iff for all } a\in\mathbb{U},\ (\mathbb{M},\varphi[\{x/a\}])\models\mathcal{F} \\
\mathfrak{I} \models \exists x\,\mathcal{F} & \text{iff there exists an } a\in\mathbb{U}, \text{ such that } (\mathbb{M},\varphi[\{x/a\}])\models\mathcal{F}.
\end{array}$$

3.4.3 Transformation in Unsorted First Order Logic

We now present an algorithm Π, which transforms sentences of the DSPF-Logic in unsorted first order logic. Let $\mathcal{F}$ be a sentence of the DSPF-Logic.

Step1: Use the rules

i) $\forall x_S\, \mathcal{G} \to \forall x_\Omega\, (x_\Omega\in S \Rightarrow \{x_S\mapsto x_\Omega\}\,\mathcal{G})$ x_Ω not free in $\mathcal{G}$

ii) $\exists x_S\, \mathcal{G} \to \exists x_\Omega\, (x_\Omega\in S \wedge \{x_S\mapsto x_\Omega\}\,\mathcal{G})$ x_Ω not free in $\mathcal{G}$

iii) $S\subseteq T \to \forall x_\Omega\, (x_\Omega\in S \Rightarrow x_\Omega\in T)$

to eliminate the $\subseteq$ relation and all variables not having sort Ω. The first and second rule are applied exactly once to every subformula of $\mathcal{F}$ starting with a quantifier, e.g. the formula $\forall x_\Omega\, \mathcal{F}$ is also changed to $\forall x_\Omega\, (x_\Omega\in\Omega \Rightarrow \mathcal{F})$. We obtain a formula $\Pi_{\text{Step1}}(\mathcal{F})$ which is equivalent to $\mathcal{F}$ and still remains in the DSPF-Logic.

Step2: The signature for the corresponding first order formulae is given by

$\Sigma := (\mathbb{V}, \mathbb{F}, \mathbb{P} \cup \{\mathcal{S} \mid \mathcal{S} \text{ is a new 1-place predicate for every sort } S\in\mathbb{S}\})$

and apply the rule

i) $t\in S \to \mathcal{S}(t)$

to eliminate the $\in$ relation. After this step we can forget about the sorts of variables and have computed a sentence $\Pi_{\text{Step2}}(\mathcal{F})$ in unsorted first order logic.

Step3: Add the following formulae

i) $\forall x_1\dots\forall x_n\, (P(x_1,\dots,x_n) \Rightarrow (\Omega(x_1)\wedge\dots\wedge\Omega(x_n)))$ for every n-place predicate $P\in\mathbb{P}$.

ii) $\forall x_1 \ldots \forall x_n\ (\Omega(f(x_1,\ldots,x_n)) \Rightarrow (\Omega(x_1) \wedge \ldots \wedge \Omega(x_n)))$ for every n-place function symbol (n>0) occurring in $\mathcal{F}_2$.

iii) $\Omega(c)$ for every constant symbol c occurring in $\Pi_{Step2}(\mathcal{F})$, if there is no constant symbol in $\Pi_{Step2}(\mathcal{F})$, choose an arbitrary constant symbol $c \in F_0$ and add $\Omega(c)$.

as a conjunction to $\Pi_{Step2}(\mathcal{F})$. The part iii) guarantees that Ω is always a non-empty relation in every unsorted first order interpretation and the parts i) and ii) ensure that predicates and functions behave strict with respect to Ω. We obtain the final first order sentence $\Pi(\mathcal{F})$.

3.4.4 Theorem: Sort Theorem

Let $\mathcal{F}$ be a sentence of the DSPF-Logic. Then $\mathcal{F}$ has a Σ-model iff $\Pi(\mathcal{F})$ has a Σ-model.

Proof:

As we have mentioned in the algorithm Π, it suffices to show that $\Pi_{Step1}(\mathcal{F})$ has a Σ-model iff $\Pi(\mathcal{F})$ has a Σ-model, because $\Pi_{Step1}(\mathcal{F})$ is equivalent to $\mathcal{F}$.

"$\Rightarrow$" Let $\mathfrak{I} = (M,\varphi)$ be a Σ-model for $\Pi_{Step1}(\mathcal{F})$, i.e. $\mathfrak{I} \models \Pi_{Step1}(\mathcal{F})$. Then we construct a Σ-model $\mathfrak{I}$ as follows: $U = A^{\perp}$, functions and predicates are interpreted in M as they are interpreted in M and the added unary predicates S_i are given by $(S_i)_M = (S_i)_{\mathcal{A}}$. Obviously U is not empty and the added formulae of $\Pi(\mathcal{F})$ hold in $\mathfrak{I}$. Thus we have only to show by induction on the structure of formulae, that $\mathfrak{I} \models \Pi_{Step1}(\mathcal{F})$ implies $\mathfrak{I} \models \Pi_{Step2}(\mathcal{F})$. We prove only the non-trivial parts.

i) $\Pi_{Step1}(\mathcal{F}) = P(t_1,\ldots,t_n)$ for a n-place predicate symbol $P \in P_n$, then $\mathfrak{I} \models P(t_1,\ldots,t_n)$ implies $\mathfrak{I} \models P(t_1,\ldots,t_n)$, because the interpretation for P in $\mathfrak{I}$ is the same as in $\mathfrak{I}$.

ii) $\Pi_{Step1}(\mathcal{F}) = t \in S$, then $\mathfrak{I} \models t \in S$ implies $\mathfrak{I} \models S(t)$ by construction.

iii) $\Pi_{Step1}(\mathcal{F}) = \forall x_\Omega\ (x_\Omega \in S \Rightarrow \mathcal{G})$, then $\mathfrak{I} \models \forall x_\Omega\ (x_\Omega \in S \Rightarrow \mathcal{G})$ implies $\mathfrak{I} \models \forall x\ (S(x) \Rightarrow \mathcal{G})$, because U and $\Omega_{\mathcal{A}}$ are both not empty, case ii) holds for $x_\Omega \in S$ and the induction hypothesis for $\mathcal{G}$ holds.

The case that the topsymbol of $\Pi_{Step1}(\mathcal{F})$ is an existential quantifier is similar to this case.

"$\Leftarrow$" Let $\mathfrak{I} = (M,\varphi)$ be a Σ-model for $\Pi(\mathcal{F})$. Then we construct a Σ-model $\mathfrak{I}$ as follows: the Σ-quasi-algebra $\mathcal{A}$ is defined with carrier $A = \Omega_M$. A is not empty because of 3.4.3-Step3-iii). For every n-place function symbol $f \in F_n$, $(u_1,\ldots,u_n) \in A^n$ we define $f_{\mathcal{A}}(u_1,\ldots,u_n) = f_M(u_1,\ldots,u_n)$ if $f_M(u_1,\ldots,u_n) \in A$ and undefined otherwise. For every sort symbol $S \in \mathbf{S}$ the set S_M is assigned. The condition $S_{\mathcal{A}} \subseteq A$ holds because of 3.4.3-Step3-i). The Σ-algebra $\mathcal{A}$ is defined according to Definition 3.1.2. Clearly $\Omega_{\mathcal{A}} = A$ and for every constant c occurring in $\Pi_{Step1}(\mathcal{F})$ we have $c_{\mathcal{A}} \in A$. For all other constants an arbitrary element of A can be assigned. For every n-place predicate symbol $P \in P_n$ we assign the relation $P_M = \{(a_1,\ldots,a_n) \mid (a_1,\ldots,a_n) \in A^n \text{ and } (a_1,\ldots,a_n) \in P_M\}$. Finally we show by induction on the structure of formulae, that $\mathfrak{I} \models \Pi_{Step2}(\mathcal{F})$ implies $\mathfrak{I} \models \Pi_{Step1}(\mathcal{F})$.

i) $\Pi_{Step2}(\mathcal{F}) = P(t_1,\ldots,t_n)$ for a n-place predicate symbol $P \in P_n$, then $\mathfrak{I} \models P(t_1,\ldots,t_n)$ implies $\mathfrak{I} \models P(t_1,\ldots,t_n)$, because 2.4.3-Step3-ii) and the fact that $\Pi_{Step2}(\mathcal{F})$ is a sentence ensure that $\mathfrak{I}(t_i) \in A$ for every i and P_M is the restriction of P_M to A^n.

ii) $\Pi_{Step2}(\mathcal{F}) = \mathcal{S}(t)$, then $\mathfrak{I} \models \mathcal{S}(t)$ implies $\mathfrak{I} \models t \in S$ by construction and the above argument for the interpretation of t.

iii) $\Pi_{Step2}(\mathcal{F}) = \forall x\ (\mathcal{S}(x) \Rightarrow \mathcal{G})$, then $\mathfrak{I} \models \forall x\ (\mathcal{S}(x) \Rightarrow \mathcal{G})$ implies $\mathfrak{I} \models \forall x_\Omega\ (x_\Omega \in S \Rightarrow \mathcal{G})$, because $a \in S_M$ implies $a \in \Omega_{\mathcal{A}}$, condition ii) holds for $\mathcal{S}(x)$ and the induction hypothesis holds for $\mathcal{G}$.

The case that the topsymbol of $\Pi_{Step2}(\mathcal{F})$ is an existential quantifier is similar to this case.■

3.4.5 Corollary: Compactness Theorem

A set S of DSPF-sentences is satisfiable iff every finite subset T of S is satisfiable.

Proof:

Follows from the Sort Theorem and the fact that the Compactness Theorem holds for first order logic.■

4 Conjunctive Normal Form

4.1 Basic Definitions

4.1.1 Definition: Negation Normal Form, Extended Conjunctive Normal Form

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 extended conjunctive normal form, if $\mathcal{F}$ is in negation normal form and $\mathcal{F} = \mathcal{F}_1 \wedge ... \wedge \mathcal{F}_n$, where the $\mathcal{F}_i = ((Q_1\ x_1)...(Q_m\ x_m)\ M)$, M is a disjunction of literals and $Q_i \in \{\forall, \exists\}$.

4.1.2 Definition: Clause Normal Form

A formula $\mathcal{F}$ is in clause normal form, if $\mathcal{F}$ is in extended conjunctive normal form and contains no quantifiers. The variables occurring in $\mathcal{F}$ are assumed to be universally quantified.

If a formula $\mathcal{F}$ is in clause normal form, we sometimes use the set oriented notation instead of the logical notation for clauses.

4.2 Introduction

For many reasons the clause normal from can not be computed using the standard algorithms, as for instance [Chang&Lee 73, Loveland 78]. First, sets may be empty and therefore some transformation rules don't hold, e.g. the rule for extending the scope of a universal quantifier:

$$\forall x_S\ \mathcal{F}[x_S] \wedge \mathcal{G} \text{ is not equivalent to } \forall x_S\ (\mathcal{F}[x_S] \wedge \mathcal{G})$$

(Take an interpretation $\mathfrak{I}$ where $S_{\mathfrak{A}} = \emptyset$ and $\mathfrak{I} \not\models \mathcal{G}$. Then $\mathfrak{I} \models \forall x_S(\mathcal{F}[x_S] \wedge \mathcal{G})$ but $\mathfrak{I} \not\models \forall x_S\ \mathcal{F}[x_S] \wedge \mathcal{G}$.) Second, Skolemization doesn't work in the usual way, because information about the domain and range sorts of a Skolem function has to be provided. We have developed an algorithm, which takes the above cases into account.

4.3 The CNF Algorithm

The input of the algorithm is a DSPF-Logic sentence $\mathcal{F}$, which is transformed in the clause normal form $CS^{\mathcal{F}}$. We prove that $\mathcal{F}$ is satisfiable iff the clause set $CS^{\mathcal{F}}$ is satisfiable, which is a sufficient condition for a refutation calculus.

Step0: Use the rule

i) $S \subseteq T \rightarrow \forall x_S\ x_S \in T$

to eliminate the $\subseteq$ predicate. After this step we have only to consider one special relation $\in$ representing the whole sort information.

Step1: Use the rules

i) $F \Leftrightarrow G \rightarrow (F \Rightarrow G) \wedge (G \Rightarrow F)$

ii) $F \Rightarrow G \rightarrow \neg F \vee G$

to eliminate implications and equivalences.

Step2 : Use the rules

i) $\neg\forall x_S\ \mathcal{F} \to \exists x_S\ \neg\mathcal{F}$

ii) $\neg\exists x_S\ \mathcal{F} \to \forall x_S\ \neg\mathcal{F}$

iii) $\neg(\mathcal{F} \wedge \mathcal{G}) \to \neg\mathcal{F} \vee \neg\mathcal{G}$

iv) $\neg(\mathcal{F} \vee \mathcal{G}) \to \neg\mathcal{F} \wedge \neg\mathcal{G}$

v) $\neg(\neg\mathcal{F}) \to \mathcal{F}$

to move the negation sign inwards.

Step3 : Use the rules

i) $\forall x_S\ (\mathcal{F} \vee \mathcal{G}) \to (\forall x_S\ \mathcal{F}) \vee \mathcal{G}$ (x_S not free in $\mathcal{G}$)

ii) $\exists x_S\ (\mathcal{F} \wedge \mathcal{G}) \to (\exists x_S\ \mathcal{F}) \wedge \mathcal{G}$ (x_S not free in $\mathcal{G}$)

iii) $\forall x_S\ (\mathcal{F} \wedge \mathcal{G}) \to (\forall x_S\ \mathcal{F}) \wedge (\forall x_S\ \mathcal{G})$ (x_S free in $\mathcal{F}$ and $\mathcal{G}$)

iv) $\exists x_S\ (\mathcal{F} \vee \mathcal{G}) \to (\exists x_S\ \mathcal{F}) \vee (\exists x_S\ \mathcal{G})$ (x_S free in $\mathcal{F}$ and $\mathcal{G}$)

v) $\forall x_S\ \forall y_T\ (\mathcal{F} \vee \mathcal{G}) \to \forall y_T\ \forall x_S\ (\mathcal{F} \vee \mathcal{G})$ (y_T free in $\mathcal{F}$ and $\mathcal{G}$, but x_S not free in either $\mathcal{F}$ or $\mathcal{G}$)

vi) $\exists x_S\ \exists y_T\ (\mathcal{F} \wedge \mathcal{G}) \to \exists y_T\ \exists x_S\ (\mathcal{F} \wedge \mathcal{G})$ (x_S, y_T as in the rule v))

to move the quantifier inwards. These rules are not necessary for Skolemization, but are useful to get small Skolem functions.

Step4 : Rename all variables, so 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 actual existential quantifier. Let $\forall y_{S1},\ldots,\forall y_{Sn}$ be the universal quantifiers which have $\mathcal{F}$ in their scope. Then replace $\mathcal{F}$ by $\{x \mapsto f(y_{S1},\ldots,y_{Sn})\}\ \mathcal{G} \wedge f(y_{S1},\ldots,y_{Sn}) \in S$, where f is a new n-place Skolem function.

Step6 : Use the rules

i) $\mathcal{H} \vee (\mathcal{F} \wedge \mathcal{G}) \to (\mathcal{H} \vee \mathcal{F}) \wedge (\mathcal{H} \vee \mathcal{G})$

ii) $\forall x_S\ (\mathcal{F} \wedge \mathcal{G}) \to \forall x_S\ \mathcal{F} \wedge \forall x_S\ \mathcal{G}$

iii) $(\forall x_S\ \mathcal{F}) \vee \mathcal{G} \to \forall x_S\ (\mathcal{F} \vee \mathcal{G})$

and variable renaming to transform the formula in extended conjunctive normal form.

Step7 : We obtain a formula like $\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n$, where every $\mathcal{F}_i = \forall x_1 \ldots \forall x_m\ C_i$ and the C_i are clauses. Now use the rule $\forall x_S\ C \to \forall x_\Omega\ (C \cup \{x_\Omega \notin S\})$, where $x_S \notin V(C)$ to ensure that a variable occurs in a clause C iff it occurs in the universal quantifiers in front of C. This is correct only together with the assumption that we consider sentences for the transformation. After the above transformation we can drop the quantifiers and keep in mind that all variables are universally quantified over their sorts. So we have reached the desired clause normal form of the input formula.

4.4 Examples

4.4.1 Schubert's Steamroller

We shall only show the transformation of the negated theorem (version 2), because the transformation of the parts 1) - 13) is simple and straightforward.

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

$\rightarrow$ $\forall x_A\ \forall y_A\ (\neg E(x_A,y_A) \vee \exists x_G \neg E(y_A,x_G))$ using the steps 2-i), 2-ii) and 2-iii)

$\rightarrow$ $\forall y_A\ \forall x_A\ (\neg E(x_A,y_A) \vee \exists x_G \neg E(y_A,x_G))$ using step 3)-v)

$\rightarrow$ $\forall y_A\ (\forall x_A\ \neg E(x_A,y_A) \vee \exists x_G \neg E(y_A,x_G))$ using step 3)-i)

$\rightarrow$ $\forall y_A\ (\forall x_A\ \neg E(x_A,y_A) \vee (\neg E(y_A,f_A(y_A)) \wedge f_A(y_A) \in G))$ using step 5), there f_A is a new 1-place Skolem function and stands for the grain eaten by the animal y_A

$\rightarrow$ $\forall y_A\ ((\forall x_A\ \neg E(x_A,y_A) \vee \neg E(y_A,f_A(y_A))) \wedge (\forall z_A\ \neg E(z_A,y_A) \vee f_A(y_A) \in G))$ using step 6)-i) and variable renaming

$\rightarrow$ $\forall y_A\ (\forall x_A\ \neg E(x_A,y_A) \vee \neg E(y_A,f_A(y_A))) \wedge \forall u_A\ (\forall z_A\ \neg E(z_A,u_A) \vee f_A(u_A) \in G)$ using step 6)-ii) and variable renaming

$\rightarrow$ $\forall y_A\ \forall x_A\ (\neg E(x_A,y_A) \vee \neg E(y_A,f_A(y_A))) \wedge \forall u_A\ \forall z_A\ (\neg E(z_A,u_A) \vee f_A(u_A) \in G)$ using step 6)-iii)

$\rightarrow$ $\{\neg E(x_A,y_A),\neg E(y_A,f_A(y_A))\}$, $\{\neg E(z_A,u_A),f_A(u_A) \in G\}$ using step 7)

Finally we present all clauses obtained from the formulae 3.3.1-1) to 3.3.1-15):

1)	$\{lupo \in W\}$	2)	$\{foxy \in F\}$
3)	$\{tweety \in B\}$	4)	$\{swallowtail \in C\}$
5)	$\{slimey \in S\}$	6)	$\{muesli \in G\}$
7)	$\{x_W \in A\}$	8)	$\{x_F \in A\}$
9)	$\{x_B \in A\}$	10)	$\{x_C \in A\}$
11)	$\{x_S \in A\}$	12)	$\{x_G \in P\}$
13)	$\{E(x_A,x_P),\neg M(y_A,x_A),\neg E(y_A,y_P),E(x_A,y_A)\}$		
14)	$\{M(x_C,x_B)\}$	15)	$\{M(x_S,x_B)\}$
16)	$\{M(x_B,x_F)\}$	17)	$\{M(x_F,x_W)\}$
18)	$\{\neg E(x_W,x_F)\}$	19)	$\{\neg E(x_W,x_G)\}$
20)	$\{E(x_B,x_C)\}$	21)	$\{\neg E(x_B,x_S)\}$
22)	$\{E(x_C,f_C(x_C))\}$	23)	$\{f_C(y_C) \in P\}$
24)	$\{E(x_S,f_S(x_S))\}$	25)	$\{f_S(y_S) \in P\}$
26)	$\{\neg E(x_A,y_A),\neg E(y_A,x_G)\}$		Theorem version 1
26')	$\{\neg E(x_A,y_A),\neg E(y_A,f_A(y_A))\}$		Theorem version 2
27')	$\{\neg E(z_A,u_A),f_A(u_A) \in G\}$		Theorem version 2

4.4.2 The Lion and the Unicorn

Here are the resulting clauses for this example:

1)	$\{monday \in MO\}$	2)	$\{tuesday \in TU\}$
3)	$\{wednesday \in WE\}$	4)	$\{thursday \in TH\}$
5)	$\{friday \in FR\}$	6)	$\{saturday \in SA\}$
7)	$\{sunday \in SU\}$	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)	$\{x_{TH} \in UL\}$	16)	$\{x_{FR} \in UL\}$
17)	$\{x_{SA} \in UL\}$	18)	$\{x_{SU} \notin UL\}$

19) $\{x_{MO} \notin UL\}$
20) $\{x_{TU} \notin UL\}$
21) $\{x_{WE} \notin UL\}$
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) $\{x_{UL} \in D\}$
31) $\{lion \in C\}$
32) $\{unicorn \in C\}$
33) $\{yesterday(monday) \in SU\}$
34) $\{yesterday(tuesday) \in MO\}$
35) $\{yesterday(wednesday) \in TU\}$
36) $\{yesterday(thursday) \in WE\}$
37) $\{yesterday(friday) \in TH\}$
38) $\{yesterday(saturday) \in FR\}$
39) $\{yesterday(sunday) \in SA\}$
40) $\{x_D \in LL, \neg LA(lion,x_D,y_D), y_D \in LL\}$
41) $\{x_D \in LL, LA(lion,x_D,y_D), y_D \notin LL\}$
42) $\{x_D \notin LL, \neg LA(lion,x_D,y_D), y_D \notin LL\}$
43) $\{x_D \notin LL, LA(lion,x_D,y_D), y_D \in LL\}$
44) $\{x_D \in UL, \neg LA(unicorn,x_D,y_D), y_D \in UL\}$
45) $\{x_D \in UL, LA(unicorn,x_D,y_D), y_D \notin UL\}$
46) $\{x_D \notin UL, \neg LA(unicorn,x_D,y_D), y_D \notin UL\}$
47) $\{x_D \notin UL, LA(unicorn,x_D,y_D), y_D \in UL\}$
48) $\{\neg LA(lion,x_D,yesterday(x_D)), \neg LA(unicorn,x_D,yesterday(x_D))\}$ Theorem

4.5 Properties

We prove, that the usual properties of the CNF-algorithm holds also for the version for DSPF-Logic.

4.5.1 Lemma: 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 $\mathcal{M}$ for the set of formulae $\{\mathcal{G}, \mathcal{F} \Rightarrow \mathcal{F}'\}$, then $\mathcal{G}'$ has model $\mathcal{M}$.

Proof:

The proof is a direct extension of the proof in [Loveland 78].■

4.5.2 Theorem: Soundness and Completeness of the CNF-algorithm

Let $\mathcal{F}$ be a sentence and $\mathcal{F}'$ the formula produced by applying the CNF-algorithm to $\mathcal{F}$. Then $\mathcal{F}$ has a model $\mathcal{M}$ iff $\mathcal{F}'$ has a model $\mathcal{M}'$.

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 where the existential quantifier is eliminated; $\forall y_{S_1} \ldots \forall y_{S_n}$ be the universal quantifiers which have $\mathcal{G}$ in their scope and

$\mathcal{G}' := \{x \mapsto f(y_{S_1},\ldots,y_{S_n})\}\mathcal{H} \wedge f(y_{S_1},\ldots,y_{S_n}) \in S$ the formula obtained from $\mathcal{G}$ after eliminating the existential quantifier.

"$\Longrightarrow$" Let $\mathcal{M} = (M,\varphi)$ be a model for $\mathcal{F}$, $(S_i)_{\mathcal{A}} \neq \emptyset$ for every i and $(M,\varphi[\{y_{S_1}/a_1,\ldots,y_{S_n}/a_n\}]) \models \exists x_S\, \mathcal{H}$ for an arbitrary Σ-assignment $\varphi[\{y_{S_1}/a_1,\ldots,y_{S_n}/a_n\}]$ with $a_i \in (S_i)_{\mathcal{A}}$ for all i. If $(M,\varphi[\{y_{S_1}/a_1,\ldots,y_{S_n}/a_n\}]) \not\models \exists x_S\, \mathcal{H}$, then there is nothing to show. We will now extend $\mathcal{M}$ to a model $\mathcal{M}'$ which satisfies $\mathcal{F}'$. If $\mathcal{G}$ holds, there must be an $a \in S_{\mathcal{A}}$, such that $(M, \varphi[\{y_{S_1}/a_1,\ldots,y_{S_n}/a_n,x/a\}]) \models \mathcal{H}$. Now define $\mathcal{M}'$ like $\mathcal{M}$ except that $f_{\mathcal{A}}(a_1,\ldots,a_n) := a$ and apply Lemma 4.5.1.

"$\Longleftarrow$" Let $\mathcal{M}' = (M',\varphi)$ be a model for $\mathcal{F}'$, $(S_i)_{\mathcal{A}} \neq \emptyset$ for every i and $(M',\varphi[\{y_{S_1}/a_1,\ldots,y_{S_n}/a_n\}]) \models \mathcal{G}'$ for a Σ-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, $f_{\mathcal{A}}$ is defined on $(a_1,\ldots,a_n)$ and $f_{\mathcal{A}}(a_1,\ldots,a_n) \in S_{\mathcal{A}}$. Therefore $\mathcal{M}' \models \exists x_S\, \mathcal{H}$ holds and application of Lemma 4.5.1 proves the theorem.■

5 The Resolution Calculus

5.1 The Herbrand Theorem

So far, we are able to transform a DSPF-Logic sentence in clause normal form. We have proved, that the sentence is unsatisfiable iff the obtained clause set is unsatisfiable. Herbrand's theorem [Herbrand 30] states that for every unsatisfiable clause set, a finite set of ground instances exists, that is unsatisfiable. We show that this result also applies to the DSPF-Logic. To this end we first have to show how to instantiate clauses. Instantiation doesn't work in the usual way, because the well sortedness of a substitution is not decidable. Therefore we have to add some literals, which guarantee correctness of instantiation if the used substitution was not well sorted. The next step is to define the Herbrand set of ground clauses. We show that we can restrict the necessary ground substitutions, which generate the Herbrand clause set, to the so called possibly well sorted ground substitutions. They are our equivalent to the well sorted substitutions known from other sorted logics, e.g. [Schmidt-Schauß 89]. Finally we prove the Herbrand theorem for our Herbrand ground clause set.

As an abbreviation for $\mathfrak{I} \models \forall ys_1 \ldots \forall ys_n C$ for some interpretation $\mathfrak{I}$ and clause C with $V(C) = \{ys_1, \ldots, ys_n\}$ we write $\mathfrak{I} \models \forall(C)$.

5.1.1 Definition: Conditioned Instantiation of Clauses

Let $C = \{L_1, \ldots, L_n\}$ be a clause and σ a substitution. Then $\sigma{\downarrow}C := \{\sigma(L_1), \ldots, \sigma(L_n)\} \cup \{(t \notin T) \mid \text{there is an } x \in DOM(\sigma) \cap V(C) \text{ with } \sigma(x) = t \text{ and } S(x) = T\}$ is called a conditioned <u>instance</u> of C.

The literals of the form $(t \notin T)$ handle the case that σ is not well sorted, i.e. in some interpretation $\mathfrak{I}$, $\mathfrak{I}(t) \notin T_{\mathfrak{A}}$. So either for every variable $x \in DOM(\sigma)$, $\sigma(x) \in (S(x))_{\mathfrak{A}}$ or some of the added literals become true.

5.1.2 Lemma: Soundness of Conditioned Instantiation

For every interpretation $\mathfrak{I}$, substitution σ and clause C, $\mathfrak{I} \models \forall(C)$ implies $\mathfrak{I} \models \forall(\sigma{\downarrow}C)$.

<u>Proof:</u>

We prove the Lemma by induction on the number of variables in $DOM(\sigma)$:

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

$|DOM(\sigma)| = n + 1$: let $x \in DOM(\sigma)$ with $\sigma(x) = t$ and $S(x) = S$. If $x \notin V(C)$ then $\sigma{\downarrow}C = (\sigma \backslash x){\downarrow}C$ and the lemma is proven by the induction hypothesis. So assume $x \in V(C)$. If $S_{\mathfrak{A}} = \emptyset$ or $\mathfrak{I}(t) \notin S_{\mathfrak{A}}$ for the interpretation $\mathfrak{I}$, then $\mathfrak{I} \models (t \notin S)$ and because $(t \notin S) \in \sigma{\downarrow}C$ we conclude $\mathfrak{I} \models \forall(\sigma{\downarrow}C)$. If $S_{\mathfrak{A}} \neq \emptyset$ and $\mathfrak{I}(t) \in S_{\mathfrak{A}}$, then $\mathfrak{I} \models \forall(\sigma{\downarrow}C)$ because x is universally closed over S in $\forall(\sigma{\downarrow}C)$ and by the induction hypothesis $\mathfrak{I} \models \forall((\sigma \backslash x){\downarrow}C)$ holds.■

5.1.3 Lemma: Associativity of Conditioned Instantiation

Let σ, τ be two substitutions, with $I(\sigma) \subseteq V(C)$, σ is idempotent and let $C = \{L_1,...,L_n\}$ be a clause. Then $\tau\downarrow(\sigma\downarrow C) = (\tau\sigma)\downarrow C$.

Proof:

$(\tau\sigma)\downarrow C$

$= \{(\tau\sigma)L_1,...,(\tau\sigma)L_n\} \cup \{t \notin T \mid$ there is an $x \in DOM(\tau\sigma) \cap V(C)$ with $\tau\sigma(x) = t$ and $S(x) = T\}$

$= \{(\tau\sigma)L_1,...,(\tau\sigma)L_n\} \cup \{\tau(t \notin T) \mid$ there is a $x \in DOM(\sigma) \cap V(C)$ with $\sigma(x) = t$ and $S(x) = T\} \cup \{s \notin S \mid$ there is a $x \in (DOM(\tau) - DOM(\sigma)) \cap V(C)$ with $\tau(x) = s$ and $S(x) = S\}$

$= \{(\tau\sigma)L_1,...,(\tau\sigma)L_n\} \cup \{\tau(t \notin T) \mid$ there is a $x \in DOM(\sigma) \cap V(C)$ with $\sigma(x) = t$ and $S(x) = T\} \cup \{s \notin S \mid$ there is a $x \in DOM(\tau) \cap V(\sigma(C))$ with $\tau(x) = s$ and $S(x) = S\}$ because σ is idempotent and $I(\sigma) \subseteq V(C)$

$= \{(\tau\sigma)L_1,...,(\tau\sigma)L_n\} \cup \{\tau(t \notin T) \mid$ there is a $x \in DOM(\sigma) \cap V(C)$ with $\sigma(x) = t$ and $S(x) = T\} \cup \{s \notin S \mid$ there is a $x \in DOM(\tau) \cap V(\sigma\downarrow C)$ with $\tau(x) = s$ and $S(x) = S\}$

$= \tau\downarrow(\{\sigma L_1,...,\sigma L_n\} \cup \{t \notin T \mid$ there is a $x \in DOM(\sigma) \cap V(C)$ with $\sigma(x) = t$ and $S(x) = T\})$

$= \tau\downarrow(\sigma\downarrow C)$ ■

5.1.4 Definition: Ω-Closed

A clause set CS is called Ω-closed, if the following two conditions are satisfied:

i) For every constant c occurring in CS, the clause $\{c \in \Omega\}$ is in CS.

ii) There is at least one clause $\{a \in \Omega\}$ in CS, for a constant $a \in F_0$.

As we have mentioned in the introduction, the formula $c \notin \Omega$ is unsatisfiable for every constant symbol $c \in F_0$.The idea of Ω-closed clause sets is to omit a special reduction rule, which eliminates literals of the form $c \notin \Omega$. If a clause set is Ω-closed, this cases can be refuted with ordinary resolution. Hence from now on we always assume that clause sets are Ω-closed. If a clause set is finite, the corresponding Ω-closed clause set can be effectively computed.

5.1.5 Definition: Possibly Well Sorted Ground Terms

Let CS be a (Ω-closed) clause set and let L_{CS} be the set of all literals occurring in CS. Then the set of all possibly well sorted ground terms $T_{gr,S}$ of sort S is inductively defined as follows:

i) For all sorts $S \in \mathbf{S}$ $(S \neq \Omega)$ and all literals $(t \in S) \in L_{CS}$, if t is a ground term, then $t \in T_{gr,S}$.

ii) For all sorts $S \in \mathbf{S}$ $(S \neq \Omega)$ and all literals $(t \in S) \in L_{CS}$, if $V(t) = \{y_{S_1},...,y_{S_n}\}$ and there are ground terms $t_i \in T_{gr,S_i}$, then $\{y_{S_1} \mapsto t_1,...,y_{S_n} \mapsto t_n\}t \in T_{gr,S}$.

iii) For all positive literals $P(t_1,...,t_n) \in L_{CS}$ and every subterm t of the t_i, if $V(t) = \{y_{S_1},...,y_{S_n}\}$ and there are ground terms $s_i \in T_{gr,S_i}$, then $\{y_{S_1} \mapsto s_1,...,y_{S_n} \mapsto s_n\}t \in T_{gr,\Omega}$.

As we have mentioned in the introduction, it is not useful to define well sorted terms as in other approaches for sorted logics. Here the well sortedness of a term depends on the clause set it occurs in, because the sort information is encoded in the special $\in$ relation. What we can do is to define a superset of well sorted ground terms, the so called possibly well sorted terms, so that every ground term which is well sorted in every interpretation satisfying the clause set, is a member of the set of possibly well sorted terms.

5.1.6 Definition: Possibly Well Sorted Substitutions

Let CS be a clause set. A substitution $\sigma = \{y_{S_1} \mapsto t_1, \ldots, y_{S_n} \mapsto t_n\}$ is called <u>possibly well sorted</u> according to CS, if for every $y_{S_i} \in DOM(\sigma)$ the following conditions hold:

i) If t_i is a variable with $S(t_i) = U$, then $\mathbf{T}_{gr,U} \cap \mathbf{T}_{gr,S_i} \neq \emptyset$.

ii) If $t_i = f(s_1, \ldots, s_n)$, then there is a possibly well sorted ground substitution λ with $\lambda t_i \in \mathbf{T}_{gr,S_i}$.

Especially if σ is ground the second condition can be changed to $t_i \in \mathbf{T}_{gr,S_i}$ for every $y_{S_i} \in DOM(\sigma)$. The tests of condition i) and ii) are both undecidable in general [Schmidt-Schauß 89]. For the decidable subcases (for most "natural" examples both conditions are decidable in polynomial time), however, restricting the unifiers to possibly well sorted substitutions significantly decreases the search space. In the sequel "possibly well sorted" is abbreviated with pws.

5.1.7 Definition: Herbrand Set of Clauses

Let $\mathcal{F}$ be a sentence, $CS^{\mathcal{F}}$ the clause set obtained by applying the normalization algorithm to $\mathcal{F}$. Then the <u>Herbrand clause set</u> $CS_H^{\mathcal{F}}$ is defined as follows:

$CS_H^{\mathcal{F}} := \{\sigma{\downarrow}C \mid C \in CS^{\mathcal{F}}, \sigma \text{ is ground, pws and } DOM(\sigma) = V(C)\}$

5.1.8 Definition: Herbrand Interpretation

An <u>HΣ-interpretation</u> is a Ω-closed set $M\Sigma$ of ground literals satisfying the following conditions:

i) There are no complementary literals in $M\Sigma$.

ii) There are no literals L_1 and L_2 in $M\Sigma$, such that L_1 is positive, contains a term t and $L_2 = (t \notin \Omega)$.

A HΣ-model of a clause set $CS_H^{\mathcal{F}}$, is an HΣ-interpretation $M\Sigma$ such that for every clause $C_H \in CS_H^{\mathcal{F}}$: $C_H \cap M\Sigma \neq \emptyset$. A H$\Sigma$-model $M\Sigma$ of a clause set $CS_H^{\mathcal{F}}$ is called <u>minimal</u>, if $M\Sigma = \bigcup_{C_H \in CS_H^{\mathcal{F}}} (C_H \cap M\Sigma)$. Note that a set of ground clauses CS has a HΣ-model iff it has a minimal one. If $M\Sigma$ is a HΣ-model for a clause set CS, we write $M\Sigma \models CS$ or $M\Sigma \models C$ for a clause $C \in CS$.

5.1.9 Lemma: Compatibility of Σ-Models and HΣ-Models

Let $\mathcal{F}$ be a formula, $CS^{\mathcal{F}}$ and $CS_H^{\mathcal{F}}$ the above defined clause sets. Then $CS^{\mathcal{F}}$ has a Σ-model $\mathcal{M}$ iff $CS_H^{\mathcal{F}}$ has a HΣ-model $M\Sigma$.

Proof:

"$\Rightarrow$" Let $\mathcal{M} = (M,\varphi)$ be a Σ-model for $CS^{\mathcal{F}}$. Then $M\Sigma := \{L \mid L \in C_H, C_H \in CS_H^{\mathcal{F}}$ and $\mathcal{M} \models L\}$. $M\Sigma$ is an $H\Sigma$-interpretation. Suppose there is a clause $C_H \in CS_H^{\mathcal{F}}$ with $C_H \cap M\Sigma = \emptyset$. We know that C_H is an instantiation of a clause $C \in CS^{\mathcal{F}}$. That means $C = \{L_1,\ldots,L_n\}$ and there exists a possibly well sorted ground substitution $\sigma = \{ys_1 \mapsto t_1,\ldots,ys_n \mapsto t_n\}$ with $DOM(\sigma) = V(C)$ and $\sigma{\downarrow}C = C_H$. If $\mathcal{M}(t_i) \notin (S_i)_{\mathcal{A}}$ for some i then $\mathcal{M} \models t_i \notin S_i$ and this contradicts $C_H \cap M\Sigma = \emptyset$. So $\mathcal{M}(t_i) \in (S_i)_{\mathcal{A}}$ for all i. Therefore if φ is an arbitrary Σ-assignment, then $(M,\varphi[\{ys_1/\mathcal{M}(t_1),\ldots,ys_n/\mathcal{M}(t_n)\}]) \not\models C$. But C is universally closed over the ys_i, hence this contradicts the assumption that $\mathcal{M}$ is a Σ-model for $CS^{\mathcal{F}}$.

"$\Leftarrow$" Let $M\Sigma$ be a minimal $H\Sigma$-model for $CS_H^{\mathcal{F}}$. We construct a Σ-model as follows: we restrict F_0 to the constants occurring in $CS^{\mathcal{F}}$. F_0 is not empty because $CS^{\mathcal{F}}$ is Ω-closed. The Σ-quasi-algebra $\mathcal{A}$ is defined with carrier $A := \Omega_{\mathcal{A}} := \{t \mid$ there is a positive literal $L \in M\Sigma$ which contains t as a subterm$\}$, for every sort $S \in \mathbf{S}$ $(S \neq \Omega)$ a set $S_{\mathcal{A}} := \{t \mid (t \in S) \in M\Sigma\}$ is assigned and for every $f \in \mathbf{F}_n$ $f_{\mathcal{A}}(t_1,\ldots,t_n) := f(t_1,\ldots,t_n)$ if $f(t_1,\ldots,t_n) \in A$ and is undefined otherwise. A is not empty because because $CS^{\mathcal{F}}$ is Ω-closed. Additionally $\Omega_{\mathcal{A}} \subseteq \mathbf{T}_{gr,\Omega}$ holds. Now $\mathcal{A}$ is extended to a Σ-algebra by defining $A^{\perp} := A \cup \{\perp\}$, where $\perp \notin A$ and for all $f \in \mathbf{F}_n$ $(n>0)$ and all $(a_1,\ldots,a_n) \in (A^{\perp})^n$, if $(t_1,\ldots,t_n) \notin \mathcal{D}(f_{\mathcal{A}})$ $f_{\mathcal{A}}(t_1,\ldots,t_n) := \perp$. Clearly the conditions $\Omega_{\mathcal{A}} = A$ and for every constant $c \in F_0$, $c_{\mathcal{A}} \in A$ hold. Next we define a Σ-structure M by adding $P_M := \{(t_1,\ldots,t_n) \mid P(t_1,\ldots,t_n) \in M\Sigma\}$ for every $P \in \mathbf{P}_n$. What we have defined so far, is a kind of initial model for $CS^{\mathcal{F}}$ 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 $\mathcal{M} = (M,\varphi)$ is a Σ-model for $CS^{\mathcal{F}}$. Assume $\mathcal{M}$ is not a Σ-model for $CS^{\mathcal{F}}$. Then there is a clause $C \in CS^{\mathcal{F}}$ with $\mathcal{M} \not\models \forall(C)$. Let $ys_1,\ldots,ys_n$ be the variables occurring in C. Because of $\mathcal{M} \not\models \forall(C)$, $(S_i)_{\mathcal{A}} \neq \emptyset$ for every i and there is a Σ-assignment $\varphi[\{ys_1/t_1,\ldots,ys_n/t_n\}]$ with $t_i \in (S_i)_{\mathcal{A}}$ and $(M,\varphi[\{ys_1/t_1,\ldots,ys_n/t_n\}]) \not\models C$. Assume $\{ys_1 \mapsto t_1,\ldots,ys_n \mapsto t_n\}$ is not a pws ground substitution. Then there is a variable ys_i with $t_i \notin \mathbf{T}_{gr,S_i}$ and we have to distinguish two cases. If $S_i \neq \Omega$, then $M\Sigma$ can not be minimal because there must be a literal $(t_i \in S_i)$, which is in $M\Sigma$ but not in $CS_H^{\mathcal{F}}$. If $S_i = \Omega$, we have a contradiction against the definition of A. Thus there is a clause $C_H \in CS_H^{\mathcal{F}}$ with $C_H = \{ys_1 \mapsto t_1,\ldots,ys_n \mapsto t_n\}{\downarrow}C$ and $M\Sigma \models C_H$. Suppose one of the literals $(t_i \notin S_i) \in M\Sigma$. If $S_i \neq \Omega$ or $t_i \in F_0$, this contradicts $t_i \in (S_i)_{\mathcal{A}}$ because $M\Sigma$ does not contain complementary literals. If $S_i = \Omega$, we have a contradiction against the Definition 5.1.8-ii). Suppose a normal literal $L \in C_H$ is in $M\Sigma$, but then we have $(M,\varphi[\{ys_1/t_1,\ldots,ys_n/t_n\}]) \models L$ and hence $(M,\varphi[\{ys_1/t_1,\ldots,ys_n/t_n\}]) \models C$ contradicting the assumption that $(M,\varphi[\{ys_1/t_1,\ldots,ys_n/t_n\}]) \not\models C$.■

5.1.10 Theorem: Herbrand Theorem

Let $\mathcal{F}$ be a formula, $CS^{\mathcal{F}}$ and $CS_H^{\mathcal{F}}$ the above defined clause sets. Then $CS^{\mathcal{F}}$ is unsatisfiable iff there is a finite subset T of $CS_H^{\mathcal{F}}$ which is unsatisfiable.

Proof:

"⇒" We show this part by contradiction. Assume $CS^{\mathcal{F}}$ is unsatisfiable and every finite subset T of $CS^{\mathcal{F}}_{H}$ is satisfiable. Then because of the Corollary 3.4.5, $CS^{\mathcal{F}}_{H}$ is satisfiable and with Lemma 5.1.9 $CS^{\mathcal{F}}$ is satisfiable, which contradicts our assumption.

"⇐" Let T be the unsatisfiable finite subset of $CS^{\mathcal{F}}_{H}$. Then $CS^{\mathcal{F}}_{H}$ has no $H\Sigma$-model and by Lemma 5.1.9 $CS^{\mathcal{F}}$ has no Σ-model and is therefore unsatisfiable.■

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 rule for the DSPF-Logic and the resolution rule for unsorted first order logic, is the additional complementarity of two literals L_1 and L_2, where $L_1 = (t \notin \Omega)$ and L_2 is a positive literal containing t as a subterm.

5.2.1 Definition: Complementarity

Two literals L_1 and L_2 are called complementary, if one of the following conditions is satisfied:

i) L_1 and L_2 are equal as atoms but L_1 is positive and L_2 is negative or vice versa.
ii) $L_1 = (t \notin \Omega)$ and L_2 is a positive literal containing t as a subterm.

5.2.2 Definition: Ground Resolution

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 $R_g(C_1,C_2, L_1,L_2) := (C_1-\{L_1\}) \cup (C_2-\{L_2\})$ is a ground resolvent of C_1 and C_2.

5.2.3 Lemma: Ground Resolution is 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 $\mathfrak{I}$ with $\mathfrak{I} \models C_1$ and $\mathfrak{I} \models C_2$, $\mathfrak{I} \models R_g(C_1,C_2,L_1,L_2)$ holds.

We shall prove this Lemma later for the general case, where we deal with arbitrary clauses.■

5.2.4 Lemma: 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.

Proof:

We will prove the following equivalent conjecture: a set CS of ground unit clauses has a Σ-model iff CS does'nt contain complementary literals.

"⇒" If CS has a model, then by Lemma 5.1.9 and the fact that CS is Ω-closed, CS has a $H\Sigma$-model and therefore CS does'nt contain complementary literals.

"$\Leftarrow$" If CS does'nt contain complementary literals, then CS can be interpreted as a HΣ-model and using Lemma 5.1.9 we conclude that CS has a Σ-model.■

5.2.5 Lemma: Ground Resolution is Complete

Let CS be an unsatisfiable set of ground clauses. Then there exists a derivation of the empty clause using ground resolution as defined in Definition 5.2.2.

Proof:

The theorem is proved by induction on the k-parameter [Anderson&Bledsoe 70], $k(CS) := \Sigma\{(|C| - 1)| C \in CS\}$, where $|C|$ is the number of literals in the clause C.

If $k(CS) = 0$, then by Lemma 5.2.4 CS contains two complementary literals. Hence one resolution 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 two resolution proofs can be combined to a resolution proof of the empty clause in CS.■

5.3 Unification

After proving the sound- and completeness on the ground level, we shall now show how to lift this results to 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 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 $P(x_S),P(f(x_T))$ using the ground substitution $\sigma = \{x_S \mapsto f(a), x_T \mapsto a\}$. Clearly $P(x_S)$ and $P(f(x_T))$ 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 logic, we give here only a brief introduction to unification. There are a lot of efficient unification procedures known, e.g. [Paterson&Wegman 78, Martelli&Montanari 79]. For simplicity we will present a rule based version of the Robinson [Robinson 65] unification procedure following [Martelli&Montanari 82].

5.3.1 Definition: Unifiable, Unifier, mgu, Unifcation problem, Solved unification problem

Two objects (terms, literals) o_1 and o_2 are called unifiable, iff there exists a substitution σ such that $\sigma(o_1) = \sigma(o_2)$. In this case the substitution σ is called a unifier of o_1 and o_2. A unifier σ of two objects o_1 and o_2 is called an mgu (most general unifier), iff for every unifier λ of o_1 and o_2 there exists a substitution τ, such that $\tau\sigma = \lambda$. If $L_1 = P(t_1,...,t_n)$ and $L_2 = P(s_1,...,s_n)$ are two literals with no variables in common, which are to be unified, then $\Gamma = \{t_1=s_1\&...\&t_n=s_n\}$ is called the unification problem for L_1 and L_2. A substitution σ solves a unification problem $\Gamma = \{t_1=s_1\&...\&t_n=s_n\}$, iff $\sigma(t_1)=\sigma(s_1),...,\sigma(t_n)=\sigma(s_n)$. A unification problem Γ is called solved, iff $\Gamma = \{x_1=u_1\&...\&x_m=u_m\}$ where the x_i are all variables, $x_i \notin V(u_j)$ and $x_i \neq x_j$ for all i and j and the corresponding substitution $\sigma = \{x_1 \mapsto u_1,...,x_m \mapsto u_m\}$ solves Γ.

5.3.2 A Unification Algorithm for Free Terms

If we speak of free terms, we mean that no equational theories are known for the function symbols. An overview for unification with non-free terms can be found in [Siekmann 89]. 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:

(1) $(x=x\&\Gamma) \rightarrow (\Gamma)$

(2) $(f(t_1,...,t_n)= f(s_1,...,s_n)\&\Gamma) \rightarrow (t_1=s_1\&...\&t_n=s_n\&\Gamma)$

(3) $(x=t\&\Gamma) \rightarrow (x=t\&\{x\mapsto t\}\Gamma)$ if x is a variable, t a non-variable term and $x \notin V(t)$, $x \in V(\Gamma)$

(4) $(t=x\&\Gamma) \rightarrow (x=t\&\Gamma)$ if x is a variable and t a non-variable term

(5) $(f(t_1,...,t_n)= g(s_1,...,s_n)\&\Gamma) \rightarrow$ STOP.FAIL if $f \neq g$

(6) $(x=t\&\Gamma) \rightarrow$ STOP.FAIL if $x \in V(t)$

If $\Gamma = \{t_1=s_1\&...\&t_n=s_n\}$ is a unification problem, then the unification algorithm always terminates on Γ. If the algorithm stops with failure, then there is no substitution σ solving Γ. If the algorithm stops without failure, then Γ is solved and the corresponding substitution σ is an mgu of the pairs $(t_1,s_1),...,(t_n,s_n)$.

5.3.3 Remark: Properties of Unifiers

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

5.3.4 Lemma

Let $L_1,..., L_n$ be some atoms, σ a ground substitution with $\sigma(L_1) =...= \sigma(L_n)$ 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.■

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 syntactically equality of literals is not a priori given, but has to be produced by unification. Another problem is that applying a ground substitution to a clause may cause some mergings of literals, e.g. applying the ground substitution $\sigma = \{x_S \mapsto a, x_T \mapsto a\}$ to the clause $\{x_S \notin T, x_T \notin T\}$ results in the ground clause $\{a \notin T, a \notin S\}$, where the literals

$x_S \notin T, x_T \notin T$ and the instantiation literal $a \notin T$ have merged. This merging has also to be simulated on the general level and is called factorization.

5.4.1 Definition: Factor

If $C = \{L_1,...,L_n\}$ is a clause and σ is a possibly well sorted substitution with $\sigma(L_{i_1}) =...= \sigma(L_{i_k})$ $(1 \leq i_k \leq n)$, then $F(C) := \sigma{\downarrow}C$ is called a <u>factor</u> of C.

5.4.2 Lemma: Factorization is Sound

If $C = \{L_1,...,L_n\}$ is a clause, $F(C)$ a factor of C, then for every interpretation $\mathfrak{I}$ with $\mathfrak{I} \models \forall(C)$, $\mathfrak{I} \models \forall(F(C))$ holds.

<u>Proof:</u>

Follows from Lemma 5.1.2.■

5.4.3 Lemma: Factor Lifting

If $C = \{L_1,...,L_n\}$ is a clause, σ a pws ground substitution with $\sigma(L_{i_1}) =...= \sigma(L_{i_k})$ $(1 \leq i_j \leq n)$, then there exists a factor $F(C)$ with $\sigma{\downarrow}C = \sigma{\downarrow}F(C)$.

<u>Proof:</u>

Choose the mgu λ of the $L_{i_1},..., L_{i_k}$ as a factor substitution and apply Lemma 5.1.3 and Lemma 5.3.4. Note that λ is pws, because σ is pws.■

5.4.4 Definition: General Resolution Rule

Let C_1 and C_2 be two clauses with no variables in common, $F(C_1)$ and $F(C_2)$ two factors of C_1 and C_2, respectively, and $L_1 \in F(C_1)$, $L_2 \in F(C_2)$. If there exists a pws mgu σ of L_1 and L_2 such that $\sigma(L_1)$ and $\sigma(L_2)$ are two complementary literals, then $R(C_1,C_2,L_1,L_2) := (\sigma{\downarrow}F(C_1)-\{\sigma(L_1)\}) \cup (\sigma{\downarrow}F(C_2)-\{\sigma(L_2)\})$ is a <u>resolvent</u> of C_1 and C_2.

Our resolution rule can be viewed as a special instantiation of Stickel's theory resolution [Stickel 85].

5.4.5 Lemma: Resolution is Sound

Let $C_1 = \{L_1,...,L_n\}$ and $C_2 = \{K_1,...,K_m\}$ be two clauses with no variables in common, $F(C_1)$ and $F(C_2)$ the corresponding factors and $L \in F(C_1)$, $K \in F(C_2)$. If resolution using the literals L and K is possible, then for every interpretation $\mathfrak{I}$ with $\mathfrak{I} \models C_1$ and $\mathfrak{I} \models C_2$, $\mathfrak{I} \models R(C_1,C_2,L,K)$ holds.

<u>Proof:</u>

Let σ be the pws mgu, such that $\sigma(L)$ and $\sigma(K)$ are complementary. Clearly for every interpretation $\mathfrak{I}$, $\mathfrak{I} \models \sigma(L)$ and $\mathfrak{I} \models \sigma(K)$ do not hold both. Thus together with Lemma 5.4.2 and Lemma 5.1.2 now yield the desired result.■

5.4.6 Lemma: Distributivity of Conditioned Instantiation

Let $C_1 = \{L_1,...,L_n\}$ and $C_2 = \{K_1,...,K_m\}$ be two clauses with no variables in common, let σ be a ground substitution such that for two literals L_i and K_j, $\sigma(L_i)$ and $\sigma(K_j)$

are complementary and let δ be the mgu of L_i and K_j. If for every $L\in C_1$, $\sigma(L) = \sigma(L_i)$ implies $L = L_i$ and the corresponding condition for C_2 holds (we call this the "distinct literal condition"), then

$\sigma{\downarrow}((\delta{\downarrow}C_1\text{-}\{\delta(L_i)\}) \cup (\delta{\downarrow}C_2\text{-}\{\delta(K_j)\})) = (\sigma{\downarrow}(\delta{\downarrow}C_1)\text{-}\{\sigma\delta(L_i)\}) \cup (\sigma{\downarrow}(\delta{\downarrow}C_2)\text{-}\{\sigma\delta(K_j)\})$

Proof:

Let $\sigma{\downarrow}R := \sigma{\downarrow}((\delta{\downarrow}C_1\text{-}\{\delta(L_i)\}) \cup (\delta{\downarrow}C_2\text{-}\{\delta(K_j)\}))$ and $R_{\downarrow\sigma} := (\sigma{\downarrow}(\delta{\downarrow}C_1)\text{-}\{\sigma\delta(L_i)\}) \cup (\sigma{\downarrow}(\delta{\downarrow}C_2)\text{-}\{\sigma\delta(K_j)\})$.

"$\subseteq$" Let $L\in\sigma{\downarrow}R$. We have to distinguish two cases:
i) $L = (t\notin S)$ is stemming from a binding $\{x_S \mapsto t\}\subseteq\sigma$. This implies that $x_S\notin DOM(\delta)$ and therefore $x_S\in V(\delta{\downarrow}C_1)$ or $x_S\in V(\delta{\downarrow}C_2)$ and x_S is not eliminated by computing the set difference of $(\delta{\downarrow}C_1\text{-}\{\delta(L_i)\})$ and $(\delta{\downarrow}C_2\text{-}\{\delta(K_j)\})$. Together with the "distinct literal condition" follows $L\in R_{\downarrow\sigma}$.
ii) $L = \sigma\delta(L')$ and wlog. $L'\in C_1$. For the reason that $L\in\sigma{\downarrow}R$, $L\neq L_i$, the "distinct literal condition" again leads to $L\in(\sigma{\downarrow}(\delta{\downarrow}C_1)\text{-}\{\sigma\delta(L_i)\})$ and therefore $L\in R_{\downarrow\sigma}$.

"$\supseteq$" Let $L\in R_{\downarrow\sigma}$ and wlog. $L\in(\sigma{\downarrow}(\delta{\downarrow}C_1)\text{-}\{\sigma\delta(L_i)\})$. Now we have to distinguish three cases:
i) $L = (t\notin S)$ is stemming from a binding $\{x_S\mapsto t\}\subseteq\sigma$. This case is analogous to the first case above.
ii) $L = \sigma(t\notin S)$ is stemming from a binding $\{x_S\mapsto t\}\subseteq\delta$. Then clearly $L\in\sigma{\downarrow}R$.
iii) $L = \sigma\delta(L')$ and $L'\in C_1$. Then $L\neq L_i$ and the "distinct literal condition" leads to $L\in\sigma{\downarrow}R$.■

5.4.7 Lemma: Resolvent Lifting

If $C_1 = \{L_1,\ldots,L_n\}$ and $C_2 = \{K_1,\ldots,K_m\}$ are two clauses with no variables in common, σ is a pws ground substitution such that for two literals L_i and K_j, $\sigma(L_i)$ and $\sigma(K_j)$ are complementary, then there exists a resolvent $R(C_1,C_2,L_i,K_j)$ such that $R_g(\sigma{\downarrow}C_1,\sigma{\downarrow}C_2,\sigma(L_i),\sigma(K_j)) = \sigma{\downarrow}R(C_1,C_2,L_i,K_j)$.

Proof:

Let $\{L_{i_1},\ldots,L_{i_k}\}\subseteq C_1$be a set of literals with L_i stemming from L_{i_1}, $\sigma(L_{i_1}) =\ldots= \sigma(L_{i_k})$ and $\{K_{j_1},\ldots,K_{j_h}\}$the corresponding set for C_2. Let λ_1 be the mgu of $\{L_{i_1},\ldots,L_{i_k}\}$, λ_2 the mgu of the $\{K_{j_1},\ldots,K_{j_h}\}$ and $\lambda = \lambda_1 \cup \lambda_2$, $F(C_1) = \lambda{\downarrow}C_1$, $F(C_2) = \lambda{\downarrow}C_2$ and δ the mgu of $\lambda(L_i)$ and $\lambda(K_j)$. The unifiers λ_1, λ_2, δ are possibly well sorted because σ is possibly well sorted.

$\sigma{\downarrow}R(C_1,C_2,L_i,K_j)$
$= \sigma{\downarrow}((\delta{\downarrow}F(C_1)\text{-}\{\delta(L_i)\}) \cup (\delta{\downarrow}F(C_2)\text{-}\{\delta(K_j)\}))$
$= \sigma{\downarrow}((\delta{\downarrow}(\lambda{\downarrow}C_1)\text{-}\{\delta\lambda(L_{i_1})\}) \cup (\delta{\downarrow}(\lambda{\downarrow}C_2)\text{-}\{\delta\lambda(K_{j_1})\}))$
$= (\sigma{\downarrow}(\delta{\downarrow}(\lambda{\downarrow}C_1))\text{-}\{\sigma(L_i)\}) \cup (\sigma{\downarrow}(\delta{\downarrow}(\lambda{\downarrow}C_2))\text{-}\{\sigma(K_j)\})$ by Lemma 5.4.6
$= (((\sigma\delta){\downarrow}(\lambda{\downarrow}C_1))\text{-}\{\sigma(L_i)\}) \cup (((\sigma\delta){\downarrow}(\lambda{\downarrow}C_2))\text{-}\{\sigma(K_j)\})$ by Lemma 5.1.3
$= ((\sigma{\downarrow}(\lambda{\downarrow}C_1))\text{-}\{\sigma(L_i)\}) \cup ((\sigma{\downarrow}(\lambda{\downarrow}C_2))\text{-}\{\sigma(K_j)\})$ by Lemma 5.3.4
$= (\sigma{\downarrow}C_1\text{-}\{\sigma(L_i)\}) \cup (\sigma{\downarrow}C_2\text{-}\{\sigma(K_j)\})$ by Lemma 5.4.3
$= R_g(\sigma{\downarrow}C_1,\sigma{\downarrow}C_2,\sigma(L_i),\sigma(K_j))$■

The statement of Lemma 5.4.7 is not strong enough to lift every ground resolution proof to a non-ground resolution proof, because one of the premises of the lemma, that both literals stem from a non-ground clause is not satisfied in general. The following example demonstrates this fact. Let

$$CS = \{\{a \in S\},\ \{P(x_S)\},\ \{\neg P(a)\}\}$$

be an unsatisfiable clause set and let

$$CS_g = \{\{a \in S\},\ \{P(a), a \notin S\},\ \{\neg P(a)\},\ \{a \in \Omega\}\}$$

be the corresponding unsatisfiable, Ω-closed set of ground clauses. On ground level, we can generate a resolvent using the clauses $\{a \in S\}$ and $\{P(a), a \notin S\}$, but there is no corresponding resolution step using two clauses in CS. If we resolve first between $\{P(x_S)\}$ and $\{\neg P(a)\}$ we get the missing literal $\{a \notin S\}$. Thus the conclusion is that lifting works, but that the ordering of the resolution steps is not arbitrary. Therefore not all, but only special ordered ground refutations can be lifted.

5.4.8 Lemma: Rearrangement of Ground Derivations

Let C be a clause derived from the ground clauses $C_1,...,C_n$ using ground resolution. If we mark some of the literals in the clauses $C_1,...,C_n$ we can always rearrange the derivation of C to a derivation of a clause C', such that the resolution steps using a marked literal all come after the steps eliminating unmarked literals and $C' \subseteq C$.

Proof:

For the proof we use a special representation of derivations. Every ground derivation using resolution can be represented as a binary tree, where the vertices are labelled with the used clauses and the edges denote the resolution steps. The root of the tree is labelled with the derived clause. In terms of the derivation tree, the Lemma states that all resolution steps using marked literals are nearer to the root than steps using unmarked literals. We prove the Lemma by induction on the number n of vertices in the tree.

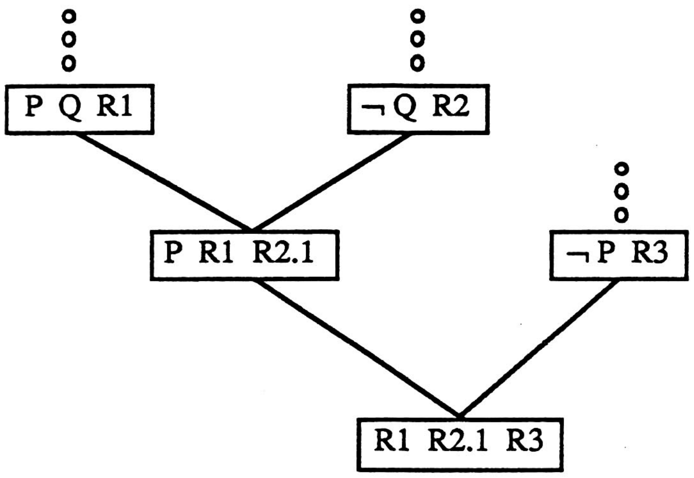
n=1 If there is only one vertex, then no resolution step is done and the Lemma trivially holds.

n=3 Then we have the following situation,

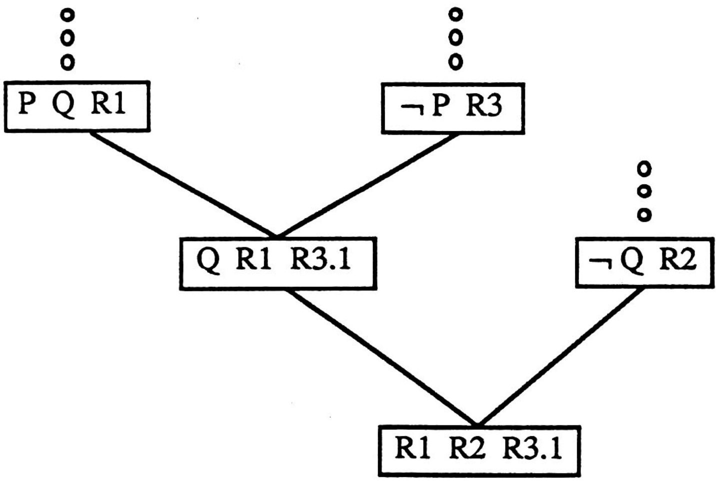
$$\begin{tikzcd}
\boxed{Q\ R1} \arrow[dr, dash] & & \boxed{\neg Q\ R2} \arrow[dl, dash] \\
& \boxed{R1\ R2} &
\end{tikzcd}$$

where Q is a literal R1, R2 are sets of literals and Q R1 is an abbreviation for the disjoint union $(\{Q\} \cup R1)$. The two clauses $(\{Q\} \cup R1)$ and $(\{\neg Q\} \cup R2)$, where Q and $\neg Q$ denote complementary literals, are used to build the resolvent $C = R1 \cup R2$. For this situation the lemma also holds, because there is only one resolution step.

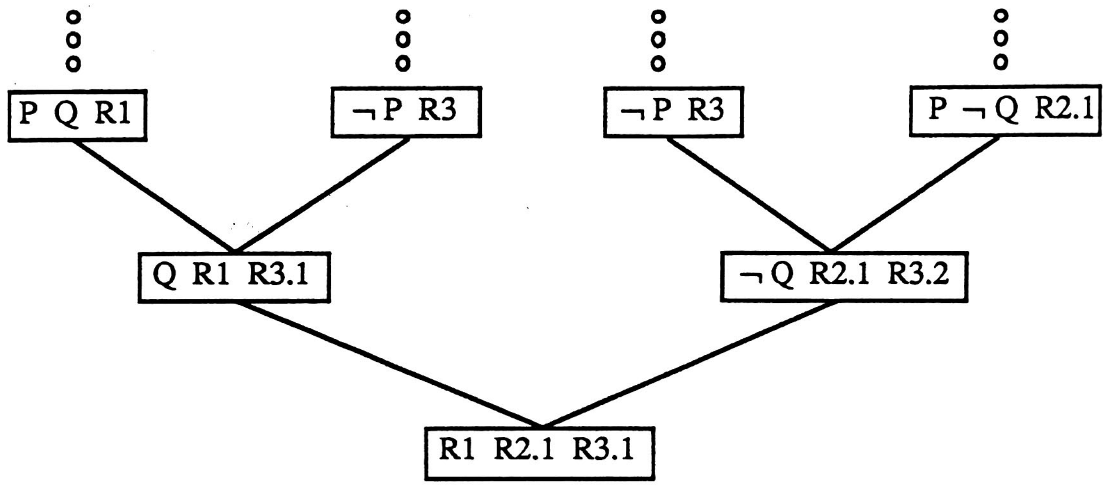
n>3 Then the root of the derivation looks as follows,



where the three circles at top of a clause stand for the subtree deriving the clause. Clause parts with additional numbers, e.g. R2.1, indicate that these parts may have changed because a literal may have been merged after the resolution step with one of the separately mentioned literals. But we will return to this problem later. If P or ¬P is marked or the derivation of (P R1 R2.1) and (¬P R3) does not contain marked literals, we have finished the proof by induction hypothesis. Thus P and ¬P are both nor marked and wlog. Q is marked. Now the idea is to rearrange the tree, such that the resolution step using Q is the closest step to the root. Assume that there was no merging between a separately mentioned literal and a clause rest during the last two resolution steps. Then we can alter the tree as follows



with R3 = R3.1, R2 = R2.1 and we have finished the proof by induction hypothesis. So let us have a closer look at the two critical mergings. First, if Q∈R3 then Q∈(R1 R2.1 R3) but Q∉(R1 R2 R3.1). Then the lemma holds because (R1 R2 R3.1)⊆(R1 R2.1 R3). Second, if P∈R2 then P∉(R1 R2.1 R3) but P∈(R1 R2 R3.1). In this case we have to add one more resolution step which is shown by the following diagram



and the proof is again completed by induction hypothesis.■

5.4.9 Theorem: Completeness of the Resolution Rule

Let CS be an unsatisfiable set of clauses. Then there exists a derivation of the empty clause using resolution.

Proof:

If CS is unsatisfiable, Theorem 5.1.10 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. We can transform this proof in a tree representation of Lemma 5.4.8 marking all literals of the form ($t \notin S$), which are used in the ground derivation, introduced by instantiation and have never merged with a literal not introduced by instantiation. Now we show by Noetherian induction that the ground proof can be lifted to a proof using resolution as defined in 5.4.4. As a measure we use pairs p of the form (n,m) with a lexicographical ordering, where n is the number of marked literals and m the number of vertices in the derivation tree.

p=(0,0) Then the empty clause is still in CS.

p=(n,m) We know that there is at least one step at the leaves of the derivation tree, such that the two ground literals used for resolution are direct instances of literals of clauses in CS. This step can be lifted using Lemma 5.4.7. Now we have to distinguish two cases. First, if this step doesn't affect the number of literals stemming only from instantiation, the derivation tree is still sorted after the resolution step, the new measure is (n,m-1) with $(n,m)>_{lex}(n,m-1)$ and the statement is proved by the induction hypothesis. Second, if the step decreases the number of literals stemming only from instantiation, we obtain the measure (n-k,m-1) with $k>0$. Hence we can rearrange the tree according to Lemma 5.4.8 obtaining a tree of size (n-k,m') with $m'<2^{m-1}$, because the rearrangement does not affect the number of the special literals, but the number of vertices in the refutation tree may grow exponentially. Nevertheless we have $(n,m)>_{lex}(n-k,m')$ and we can again finish the proof by applying the induction hypothesis.■

5.4.10 Definition: $\in$-Resolution

Let $R = R(C,D,L,K)$ be a resolvent using the clauses C, D and the literals $L \in F(C)$ and $K \in F(D)$. Then R is a resolvent generated by $\in$-Resolution if L does not contain $\in$ as predicate symbol, or all literals in C, D have the $\in$ symbol as predicate symbol.

5.4.11 Theorem: $\in$-Resolution is Sound and Complete

Let CS be an unsatisfiable set of clauses. Then there exists a correct derivation of the empty clause using $\in$-resolution.

Proof:

Follows from the fact that resolution is correct and the proof of Theorem 5.4.9.■

5.5 Examples

As we have not yet implemented the DSPF-calculus, all proofs are made by hand, but discussed in a way which takes the behaviour of a pure resolution theorem prover into consideration.

5.5.1 Schubert's Steamroller

Here is a proof of the theorem using the DSPF-calculus. We can omit the clauses stemming from the Ω-closing of the clause set, because they play no role in the proof. The sets of pws ground terms and the clauses are given as follows:

$T_{gr,A}$ = {lupo, foxy, tweety, swallowtail, slimey}
$T_{gr,B}$ = {tweety}
$T_{gr,C}$ = {swallowtail}
$T_{gr,F}$ = {foxy}
$T_{gr,W}$ = {lupo}
$T_{gr,S}$ = {slimey}
$T_{gr,G}$ = {muesli}
$T_{gr,P}$ = {muesli, f_C(swallowtail), f_S(slimey)}

1) $\{lupo \in W\}$
2) $\{foxy \in F\}$
3) $\{tweety \in B\}$
4) $\{swallowtail \in C\}$
5) $\{slimey \in S\}$
6) $\{muesli \in G\}$
7) $\{x_W \in A\}$
8) $\{x_F \in A\}$
9) $\{x_B \in A\}$
10) $\{x_C \in A\}$
11) $\{x_S \in A\}$
12) $\{x_G \in P\}$
13) $\{E(x_A,x_P),\neg M(y_A,x_A),\neg E(y_A,y_P),E(x_A,y_A)\}$
14) $\{M(x_C,x_B)\}$
15) $\{M(x_S,x_B)\}$
16) $\{M(x_B,x_F)\}$
17) $\{M(x_F,x_W)\}$
18) $\{\neg E(x_W,x_F)\}$
19) $\{\neg E(x_W,x_G)\}$
20) $\{E(x_B,x_C)\}$
21) $\{\neg E(x_B,x_S)\}$
22) $\{E(x_C,f_C(x_C))\}$
23) $\{f_C(y_C) \in P\}$
24) $\{E(x_S,f_S(x_S))\}$
25) $\{f_S(y_S) \in P\}$

26) $\{\neg E(x_A,y_A),\neg E(y_A,x_G)\}$ Theorem version 1

For the clauses we use the above numbering and we prove the first version of the theorem. Here is an example for the notation used to describe the derivation:

13,4&18,1 R $\{x_A\mapsto x_W,y_A\mapsto x_F\}$
    27: $\{E(x_W,x_P),\neg M(x_F,x_W),\neg E(x_F,y_P),x_W\notin A,x_F\notin A\}$

The notation 13,4&18,1 R means that the fourth literal of clause 13 and the first literal of clause 18 are used to generate a resolvent with pws mgu $\{x_A\mapsto x_W,y_A\mapsto x_F\}$. If we have generated a factor before resolution, we write R&F instead of R. The new resolvent $E(x_W,x_P) \vee \neg M(x_F,x_W) \vee \neg E(x_F,y_P) \vee x_W\notin A \vee x_F\notin A$ gets the number 27.

13,4&18,1 R $\{x_A\mapsto x_W,y_A\mapsto x_F\}$
    27: $\{E(x_W,x_P),\neg M(x_F,x_W),\neg E(x_F,y_P),x_W\notin A,x_F\notin A\}$

27,4&7,1 R $\{y_W\mapsto x_W\}$
    28: $\{E(x_W,x_P),\neg M(x_F,x_W),\neg E(x_F,y_P),x_W\notin W,x_F\notin A\}$
where we have renamed clause 7 with $\{x_W\mapsto y_W\}$

28,4&1,1 R $\{x_W\mapsto lupo\}$
    29: $\{E(lupo,x_P),\neg M(x_F,lupo),\neg E(x_F,y_P),x_F\notin A\}$

Note that we have required two steps to eliminate the literal $x_W\notin A$. To shorten the proof, for reasons of readability and because these steps are unique, we shall omit them from now on, but we shall mention the used clauses.

29,4&8,1&2,1 R $\{x_F\mapsto foxy\}$
    30: $\{E(lupo,x_P),\neg M(foxy,lupo),\neg E(foxy,y_P)\}$

30,2&17,1&1,1&2,1 R $\{x_F\mapsto foxy,x_W\mapsto lupo\}$
    31: $\{E(lupo,x_P),\neg E(foxy,y_P)\}$

31,1&19,1&1,1&12,1&6,1 R $\{x_P\mapsto muesli,x_W\mapsto lupo,x_G\mapsto muesli\}$
    32: $\{\neg E(foxy,y_P)\}$

32,1&13,1&8,1&2,1 R $\{x_P\mapsto y_P,x_A\mapsto foxy\}$
    33: $\{\neg M(y_A,foxy),\neg E(y_A,y_P),E(foxy,y_A),y_P\notin P\}$

33,1&16,1&2,1&9,1&3,1 R $\{x_B\mapsto tweety,x_F\mapsto foxy,y_A\mapsto tweety\}$
    34: $\{\neg E(tweety,y_P),E(foxy,tweety),y_P\notin P\}$

13,4&21,1&9,1&3,1&11,1&5,1 R $\{x_S\mapsto slimey,x_B\mapsto tweety,y_A\mapsto slimey,x_A\mapsto tweety\}$
    35: $\{E(tweety,x_P),\neg M(slimey,tweety),\neg E(slimey,y_P)\}$

35,3&24,1&5,1&25,1 R $\{x_S\mapsto slimey,y_P\mapsto f_S(slimey)\}$
    36: $\{E(tweety,x_P),\neg M(slimey,tweety)\}$

36,2&15,1&5,1&3,1 R $\{x_S\mapsto slimey,x_B\mapsto tweety\}$
    37: $\{E(tweety,x_P)\}$

34,2&26,1&8,1&2,1&9,1&3,1 R $\{x_A\mapsto foxy,y_A\mapsto tweety\}$
    38: $\{\neg E(tweety,y_P),\neg E(tweety,x_G),y_P\notin P\}$

38,1&37,1&12,1&6,1 R&F $\{x_P \mapsto muesli, y_P \mapsto muesli, x_G \mapsto muesli\}$

39: $\{\}$

For this example we can draw two conclusions. First, the derivation process is restricted as in Walther's logic, e.g. there are 12 possible initial resolution steps in both logics. This is the fact because every literal of the form $t \in S$ occurs in a unit clause. Therefore the set of possible well sorted ground terms is exactly the set of well sorted ground terms, i.e. for every interpretation $\mathfrak{I}$ satisfying the clause set, if $t \in \mathbf{T}_{gr,S}$ then $\mathfrak{I}(t) \in S_{\mathfrak{A}}$. Thus there is no difference in the restriction imposed on the resolution steps between our and Walther's formalization. So we have nothing lost in deduction power by putting the term and sort declarations in the formulae itself. Second, the proof becomes much longer because the additional literals stemming from instantiation have to be eliminated too. But the increased proof length has no effect on the search space, because the additional elimination steps are unique.

5.5.2 The Lion and the Unicorn

The clause set of the problem, the sets of pws ground terms and a refutation looks as follows:

1)	$\{monday \in MO\}$	2)	$\{tuesday \in TU\}$	3)	$\{wednesday \in WE\}$
4)	$\{thursday \in TH\}$	5)	$\{friday \in FR\}$	6)	$\{saturday \in SA\}$
7)	$\{sunday \in SU\}$	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)	$\{x_{TH} \in UL\}$
16)	$\{x_{FR} \in UL\}$	17)	$\{x_{SA} \in UL\}$	18)	$\{x_{SU} \notin UL\}$
19)	$\{x_{MO} \notin UL\}$	20)	$\{x_{TU} \notin UL\}$	21)	$\{x_{WE} \notin UL\}$
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)	$\{x_{UL} \in D\}$
31)	$\{lion \in C\}$	32)	$\{unicorn \in C\}$		

33) $\{yesterday(monday) \in SU\}$ 34) $\{yesterday(tuesday) \in MO\}$

35) $\{yesterday(wednesday) \in TU\}$ 36) $\{yesterday(thursday) \in WE\}$

37) $\{yesterday(friday) \in TH\}$ 38) $\{yesterday(saturday) \in FR\}$

39) $\{yesterday(sunday) \in SA\}$ 40) $\{x_D \in LL, \neg LA(lion, x_D, y_D), y_D \in LL\}$

41) $\{x_D \in LL, LA(lion, x_D, y_D), y_D \notin LL\}$

42) $\{x_D \notin LL, \neg LA(lion, x_D, y_D), y_D \notin LL\}$

43) $\{x_D \notin LL, LA(lion, x_D, y_D), y_D \in LL\}$

44) $\{x_D \in UL, \neg LA(unicorn, x_D, y_D), y_D \in UL\}$

45) $\{x_D \in UL, LA(unicorn, x_D, y_D), y_D \notin UL\}$

46) $\{x_D \notin UL, \neg LA(unicorn, x_D, y_D), y_D \notin UL\}$

47) $\{x_D \notin UL, LA(unicorn, x_D, y_D), y_D \in UL\}$

48) $\{\neg LA(lion, x_D, yesterday(x_D)), \neg LA(unicorn, x_D, yesterday(x_D))\}$ Theorem

$\mathbf{T}_{gr,MO} = \{monday, yesterday(tuesday)\}$

$T_{gr,TU}$ = {tuesday, yesterday(wednesday)}
$T_{gr,WE}$ = {wednesday, yesterday(thursday)}
$T_{gr,TH}$ = {thursday, yesterday(friday)}
$T_{gr,FR}$ = {friday, yesterday(saturday)}
$T_{gr,SA}$ = {saturday, yesterday(sunday)}
$T_{gr,SU}$ = {sunday, yesterday(monday)}
$T_{gr,D}$ = {monday, yesterday(tuesday), tuesday, yesterday(wednesday), wednesday, yesterday(thursday), ..., sunday, yesterday(monday)}
$T_{gr,LL}$ = $T_{gr,UL} = T_{gr,D}$
$T_{gr,C}$ = {lion, unicorn}

The sets of pws ground terms $T_{gr,LL}$ and $T_{gr,UL}$ include all days of the week because of the positive literals $y_D \in LL$ and $y_D \in UL$ in the clauses 40) to 47). Comparing the number of initial resolution possibilities, in our formulation there are 12, in the sorted formulation of [Ohlbach&Schmidt-Schauß 85] using 127 function declarations there are more than 40 and using the unsorted formulation there are more than 200 possible steps. The refutation is as follows:

48,1&41,2 R $\{z_D \mapsto x_D, y_D \mapsto \text{yesterday}(x_D)\}$
  49: $\{\neg LA(\text{unicorn}, x_D, \text{yesterday}(x_D)), x_D \in LL, \text{yesterday}(x_D) \notin LL, x_D \notin D, \text{yesterday}(x_D) \notin D\}$
  where 48 has been renamed with $\{x_D \mapsto z_D\}$

49,1&47,2 R $\{z_D \mapsto x_D, y_D \mapsto \text{yesterday}(x_D)\}$
  50: $\{x_D \in LL, \text{yesterday}(x_D) \notin LL, x_D \notin UL, \text{yesterday}(x_D) \in UL, x_D \notin D, \text{yesterday}(x_D) \notin D\}$
  where 49 has been renamed with $\{x_D \mapsto z_D\}$

At this point, the clause 50 denotes the day thursday. For example the first literal $x_D \in LL$ can only be resolved against the clauses 11) to 14), because the resolution steps using one of the clauses 49) to 47) are forbidden by $\in$-resolution. Every resolution step using one of the clauses 12) to 14) leads to a resolvent with pure literals and thus the following step is unique.

50,1&11,1 R $\{x_D \mapsto x_{TH}\}$
  51: $\{\text{yesterday}(x_{TH}) \notin LL, x_{TH} \notin UL, \text{yesterday}(x_{TH}) \in UL, x_{TH} \notin D, \text{yesterday}(x_{TH}) \notin D, x_{TH} \notin D\}$

From now all steps are deterministic because there is always only one possible resolution step for every literal. For example the first literal can only be resolved against clause 10) because the unifiers which are needed to generate a resolvent with one of the clauses 8) or 9) are not possibly well sorted.

51,1&10,1&4,1&25,1&36,1&24,1 R $\{x_{WE} \mapsto \text{yesterday(thursday)}, x_{TH} \mapsto \text{thursday}\}$
  52: $\{\text{thursday} \notin UL, \text{yesterday(thursday)} \in UL\}$

52,1&15,1&4,1 R $\{x_{TH} \mapsto \text{thursday}\}$

53: {yesterday(thursday)$\in$ UL}

53,1&21,1&36,1 R {$x_{WE} \mapsto$yesterday(thursday)}

54: {}

To sum up the behaviour of our calculus for this example, the derivation process is more restrictive than in the order sorted logics of [Walther 87, Schmidt-Schauß 89], because we can express more information in terms of sorts. The proof becomes much longer, because the additional literals introduced by instantiation have to be eliminated too. Since the additional elimination steps are unique, the increased proof length has no effect on the search space.

6 Conclusions and Future Work

We have presented a sound and complete resolution calculus for the DSPF-logic, a very expressive sorted logic which supports empty sorts, partial functions and arbitrary sort and term declarations. By putting all information about sorts into the formulae, we have gained a high degree of freedom for handling sort information. The restrictiveness of the calculus is not fully investigated so far, but if we transform the sort declarations of [Walther 87, Schmidt-Schauß 89] in our special relations, we obtain a formalization where the possibly well sorted ground terms of the DSPF-calculus correspond exactly to the well sorted ground terms in their logics. Therefore the question whether a substitution is possibly well sorted corresponds to the unification problem under sort declarations in their calculi and hence the DSPF-calculus is as restrictive as these calculi. Nevertheless the DSPF-calculus presented in this paper should be viewed as a theoretical basis for more hybrid and dynamic order sorted calculi.

Extensions for the DSPF-calculus are possible along two different lines. One possibility is to extend the expressiveness by allowing functions on sorts, special relations which are not handled as predicates (see the Lion and the Unicorn example), equality on sorts and terms, term rewriting or special declarations for finite sets. The other way is to improve the calculus in it's actual form by incorporating a dynamic translation of $\in$ unit clauses into special declarations which can be used directly by sorted unification algorithms.

Acknowledgments: I would like to thank Hans Jürgen Ohlbach for his intensive support in preparing this paper. Christoph Lingenfelder, Axel Präcklein and Rosa Ruggeri contributed a lot of good ideas and suggestions concerning the approach.

7 References

Anderson&Bledsoe 70	Anderson, R., Bledsoe, W.W., *A linear format for resolution with merging and a new technique for establishing completeness.* Journal of the ACM, vol. 17, July 1970, pp. 525-534
Chang&Lee 73	Chang, C.-L., Lee, R.C.-T., *Symbolic Logic and Mechanical Theorem Proving.* Computer Science and Applied Mathematics Series (Editor Werner Rheinboldt), Academic Press, New York, 1973.
Cohn 87	Cohn, A., *A More Expressive Formulation of Many Sorted Logic.* Journal of Automated Reasoning, vol. 3, No. 2, pp. 113-200, 1987.
Gallier 86	Gallier, J.H., *Logic for Computer Science,* Harper & Row, Publishers, New York, 1986.
Herbrand 30	Herbrand, J., *Recherches sur la theorie de la demonstration Travaux de la Societe des Sciences et des lettres de Varsovie, Classe III.* Science mathematique et physique, No 33, 1930. Also in "Logical Writings" (W.D. Goldfarb ed.), D. Reidel Publishing Company, 1971.
Herold 83	Herold, A., *Some Basic Notions of First-Order Unification Theory.* Interner Bericht 15/83, Inst. für Informatik I, Universität Kaiserslautern, 1983.
Loveland 78	Loveland, D., *Automated Theorem Proving: A Logical Basis.* Fundamental Studies in Computer Science, Vol. 6, North-Holland, New York, 1978.
Martelli&Montanari 79	Martelli, A., Montanari, U., *An Efficient Unification Algorithm.* Univ. of Pisa, Techn. report, 1979.
Martelli&Montanari 82	Martelli, A., Montanari, U., *An Efficient Unification Algorithm.* ACM Trans. Programming Languages and Systems 4, 2, pp. 258-282, 1982.
Oberschelp 62	Oberschelp, A., *Untersuchungen zur mehrsortigen Quantorenlogik.* Mathematische Annalen 145, 1962.
Ohlbach& Schmidt-Schauß 85	Ohlbach, H.J., Schmidt-Schauß, M., *The Lion and the Unicorn.* Journal of Automated Reasoning, Vol. 1, No. 3, pp. 327-332, 1985
Paterson&Wegman 78	Paterson, M., Wegman, M., *Linear Unification.* Journal of Comp. and Syst., 16, 1978.
Robinson 65	Robinson, J.A., *A Machine-Oriented Logic Based on the Resolution Principle.* J.ACM, Vol. 12, No. 1, pp. 23-41, 1965.

Schmidt-Schauß 89	Schmidt-Schauß, M., *Computational aspects of an order sorted logic with term declarations.* Lecture Notes in Artificial Intelligence, J. Siekmann(ed.), Springer Verlag, 1989.
Siekmann 89	Siekmann, J., *Unification Theory.* Journal of Symbolic Computation, Special Issue on Unification, C. Kirchner(ed.), vol 7, pp. 207-274, 1989.
Smullyan 78	Smullyan, R.,*What is the name of this book ?.* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
Stickel 85	Stickel, M.E., *Automated Deduction by Theory Resolution,* Journal of Automated Reasoning, Vol. 1, No. 4, pp. 333-356, 1985.
Stickel 86	Stickel, M.E., *Schubert's Steamroller Problem: Formulation and Solutions,* Journal of Automated Reasoning, Vol. 2, No. 1, pp. 89-101, 1986.
Walther 87	Walther, Ch., *A Many-sorted Calculus based on Resolution and Paramodulation.* Research Notes in Artificial Intelligence, Pitman Ltd., London 1987.

REPRESENTING AND ANALYZING CAUSAL, TEMPORAL, AND HIERARCHICAL RELATIONS OF DEVICES

Vom Fachbereich Informatik
der Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

von

Dipl.-Inform. Hans Voss

Berichterstatter:	Prof. Dr. Michael M. Richter Dr. habil. Werner Dilger
Dekan:	Prof. Dr. Otto Mayer

Tag der wissenschaftlichen Aussprache: 18. Dezember 1986

D 386

ABSTRACT: HIQUAL is a deep modeling language for the representation and analysis of techno-physical systems. It provides for object oriented modelings with highly independent models to be constructed according to the message passing paradigm. The same real world system may be represented at different levels of abstraction with explicit specifications of structural relations between neighboring levels. All models at all abstraction levels can be analyzed without the need to consider the deeper levels, that means abstraction hierarchies in HIQUAL are not definitional hierarchies as usually found in ordinary programming languages.

Communication between different models is interpreted as flow of material, forces, or information. Quantities capturing these changes may be continuous, thus only allowing for smooth transitions of successive values, or they may be digital with no such restrictions of value transitions.

We describe the semantics of a system of models as a set of temporally and causally related temporal intervals that are denoted by dynamic states and events of the models. Using this approach we obtain a uniform semantics for single models, for a system of horizontally connected models at the same level of abstraction, and for a system of vertically connected models of different levels. We demonstrate that our temporal approach is superior to other techniques involving global state semantics, because parallelism and other temporal aspects including temporal uncertainty are more naturally represented.

Since deep modeling as a subfield of Artificial Intelligence is quite new, the thesis presents a rather extended survey of the whole field.