Extending the Resolution Method with Sorts

Christoph Weidenbach*
Max-Planck-Institut fur Informatik
Im Stadtwald
6600 Saarbrucken, Germany
email: weidenb@mpi-sb.mpg.de

Abstract

In this paper I extend the standard first-order resolution method with special reasoning mechanisms for sorts. Sorts are unary predicates. Literals built from unary predicates are called sort literals. Negative sort literals can be compiled into restrictions of the relevant variables to sorts or can be deleted if they fulfill special conditions. Positive sort literals define the sort-theory. Sorted unification exploits the sort restrictions of variables with respect to the sort theory. As occurrences of sort literals are not restricted, it may be necessary to add additional literals to resolvents and factors and to dynamically change the sort theory used by sorted unification during the deduction process. The calculus I propose thus extends the standard resolution method with sorted unification, residue literals and a dynamic processing of the sort information. I show that this calculus generalizes and improves existing approaches to sorted reasoning. Finally, I give some applications to automated theorem proving and abduction.

1 Introduction

One promising approach for increasing the strength of automated reasoning systems is the integration of theories into the standard first-order resolution calculus. For special theories there are more efficient methods than standard resolution. One theory that has been investigated is the theory of unary predicates called "sorts", see e.g. the logics of Beierle et al. [Beierle *et al.,* 1992], Cohn [Cohn, 1992], Frisch [Frisch, 1991], Schmidt-SchauB [Schmidt-Schaufi, 1989], Walther [Walther, 1987], or Weidenbach et al. [Weidenbach and Ohlbach, 1990]. All these approaches offer special language constructs and reasoning facilities for sorts. They differ in the restrictions imposed on the sort theory and the way the sort theory is processed. This will be discussed in more detail in Section 4. In the following 1 will explain why the approach presented in this paper generalizes and improves existing results. What all approaches have in common is the incorporation of sorted reasoning in the unification algorithm. Here is an example for sorted resolution in comparison with standard resolution.

*This work was supported by the ESPRIT project 6471 MEDLAR of the European Community

The database A consisting of the clauses

Δ:	
(1)	$Man(peter)$
(2)	$Man(x) \Rightarrow Human(x)$
(3)	$Human(y) \Rightarrow Human(father(y))$
(4)	$Human(y) \wedge Human(z) \Rightarrow Love(y, z)$

can be represented in a sorted formalization by the database Δ' of clauses:

Δ':	
(1)	$Man(peter)$
(2)	$Human(x_{Man})$
(3)	$Human(father(y_{Human}))$
(4)	$Love(y_{Human}, z_{Human})$

An immediate difference between Δ and Δ' is that Δ' contains fewer literals. The negative sort literals (2)1, (3)1, (4)1 and (4)2 of Δ are replaced by restrictions on the variables. (In order to refer to clauses and literals the clause number and the position of the literal in the clause are used. Thus (4)1 refers to the literal $Human(y)$ of clause (4).) Second, there are infinitely many non redundant resolvents derivable from Δ (e.g. clause (3) with itself) whereas in Δ' no resolution step is possible at all. In order to derive the query $Love(peter, father(peter))$ from Δ and Δ' the clause

(5) $\neg Love(peter, father(peter))$

is added to the databases. Using Δ four resolution steps and one factorization step are needed, whereas using Δ', one single resolution step between (5)1 and (4)1 yields the empty clause. Sorted unification checks whether the terms *peter* and $father(peter)$ are of sort *Human*. This can be done by exploiting the sort theory consisting of the positive sort literals (1)1, (2)1, and (3)1 of Δ'. Thus third, the sorted approach succeeds with fewer inference steps. This simple example demonstrates the advantages of sorted reasoning compared to standard resolution.

In general, the sorted unification algorithm is more complex than standard unification. As compared to

standard resolution this has to be taken into account. For nearly all examples the behaviour of sorted unification is not harmful. In particular, for sort theories that are difficult the sorted unification process is much more efficient than standard resolution. This will be explained in Section 4. For Δ' the complexity of sorted unification is the same as for standard unification, Δ' is a simple database. It can be processed (after translation in the respective formalism) in the described way by all approaches mentioned above [Beierle et al., 1992; Cohn, 1992; Frisch, 1991; Schmidt-Schaufi, 1989; Walther, 1987; Weidenbach and Ohlbach, 1990]. For the next database Δ'' this is not the case. The unary predicates Man and Woman cannot be represented as a sort in the logics of Frisch, Schmidt-Schaufi, and Walther. They can be represented in the logics of Beierle et al., Cohn, and Weidenbach, but their calculi consist of more rules other than the usual resolution and factorization rule. In this paper I will show how any unary predicate can be processed as a sort just by modifying the standard resolution method. Now consider Δ'':

Δ'':	
(1)	Man(peter) ∨ Woman(peter)
(2)	Love(x_{Man} ,mary)
(3)	Love(ywoman, Paul)
(4)	¬Love(peter,paul)

From Δ'' the query Love(peter , mary) must be derivable. Adding the clause (5) ¬Love(peter, mary) to Δ'' offers only two resolution possibilities between (4)1 and (3)1 and between (5)1 and (2)1. For the two resolution steps sorted unification has to guarantee that peter is of sort Woman or peter is of sort Man, respectively. Clause (1) is valid if peter is of sort Man or peter is of sort Woman. Therefore at least one declaration of clause (1) has to be considered for sorted unification. For Δ', the sort theory consists of sort literals occurring in unit clauses. Now it becomes obvious that considering such sort theories is too restrictive to get a complete calculus. From every clause consisting of positive sort literals exactly one literal must be chosen for the sort theory (see also Theorem 8). For Δ'' if we choose (1)1, Man(peter) as the sort theory, from (5)1 and (2)1 we derive by resolution (6) Woman(peter). The literal Woman(peter) is the result of sorted unification, because (1)1 is not the only literal of clause (1). Clause (6) subsumes clause (1). The sort theory must be changed to the sort literal (6)1, Woman(peter). Then the empty clause is derived from (4)1 and (3)1.

The example demonstrates three important aspects of the resolution method extended with sorts:

- A notion of "conditional well sortedness" is needed, where the additional literals of declarations coming from non-unit clauses are collected. (See Section 2.2.)
- The declarations considered by sorted unification have to be changed dynamically during the deduction process. (See Theorem 8.)
- From each clause which consists of declarations only, at least one declaration must be chosen for sorted unification in order to obtain a complete calculus. (See Theorem 8.)

The next section introduces the new sorted resolution method. The method is applied to two examples in Section 3. Section 4 relates the new sorted resolution method to existing work in a more abstract way. The paper ends with a short summary in Section 5. All proofs are omitted and can be found in an internal report [Weidenbach, 1991].

2 The Sorted Resolution Method

The starting point of the new method is the standard resolution method for first-order logic. First, the standard syntax is extended with sorts. Then the notion of well sortedness is introduced. Finally, the modified resolution and factorization rule and the sorted unification procedure are presented.

2.1 Syntax and Semantics

Syntax: The standard signature Σ consisting of the sets V_Σ, P_Σ, F_Σ of variables, predicates, and functions respectively, is extended in the following way. Unary predicates are called *sorts*. S_Σ is the set of all sorts ($S_\Sigma \subseteq P_\Sigma$). The function $\mathcal{S}$, $\mathcal{S}: V_\Sigma \to S_\Sigma \cup \{\top\}$, where $\top \notin \Sigma$, maps variables to sorts such that for each sort $S \in S_\Sigma \cup \{\top\}$ there are infinitely many variables x with $\mathcal{S}(x) = S$. In examples (as in Δ', Δ'') it is indicated that a variable x has sort S by writing x_S. If not stated otherwise, variables not annotated with a sort have sort $\top$. Terms, literals, clauses, formulae and substitutions are defined in the usual (unsorted) way. Literals built from sorts are called *sort literals*. Positive sort literals are called *declarations*.

With $DOM(\sigma) := \{x \mid x\sigma \neq x\}$ we denote the finite domain of a substitution σ. Specific substitutions are described by their variable-term pairs, e.g. $\{x \mapsto a\}$ denotes the substitution x maps to a.

The function $\mathcal{V}$ maps terms, formulae and sets of such expressions to their variables.

Semantics: The semantics of sorted variables is given by the two relativization rules

$$\begin{aligned} \forall x\, \mathcal{F} &\to \forall y\, (S(y) \Rightarrow \mathcal{F}\{x \mapsto y\}) \\ \exists x\, \mathcal{F} &\to \exists y\, (S(y) \wedge \mathcal{F}\{x \mapsto y\}) \end{aligned}$$

where $\mathcal{S}(x) = S$, $S \neq \top$, $\mathcal{S}(y) = \top$ and y does not occur in the formula $\mathcal{F}$. Variables of sort $\top$ (the top sort) have the same semantics as standard variables. If the first relativization rule is applied to Δ', Δ is obtained. The database Δ is special case of a sorted database where all variables have sort $\top$.

2.2 Conditional Well Sorted Expressions

The processing of Δ'' showed the need for an extended notion of well-sortedness. A conditional well sorted expression consists of a well sorted expression in the usual sense [Schmidt-Schaufi, 1989; Frisch, 1991] plus a set of literals. Conditional expressions are written with a prime.

Definition 1 (Conditional Expressions)
A pair (L, C) is called a *conditional declaration* (*conditional term, conditional substitution*) if C is a finite set of literals and L a declaration (term, substitution).

The next step is to define what we mean by a sort theory. A *sort theory* $\mathcal{L}$ is a set of conditional declarations. The notion of well-sorted terms and sorted unification is defined with respect to a sort theory $\mathcal{L}$. The sort theory is always a part of the database (signature), although it is separated away and static in nearly all approaches known so far [Stickel, 1985; Walther, 1987; Schmidt-Schauß, 1989; Bürckert, 1991; Frisch, 1991; Beierle *et al.*, 1992; Cohn, 1992]. In this approach the sort theory is chosen dynamically from the database. In the course of generating new clauses it may happen that the sort theory changes (see Δ''). Nevertheless we can assume that $\mathcal{L}$ is always finite and all conditional declarations are variable disjoint. The following definitions are presented with respect to a fixed sort theory $\mathcal{L}$.

Definition 2 (Conditional Well Sorted Terms)
The set of conditional well sorted terms (abbreviated by cws. terms) $\mathcal{T}_S$ of sort S is recursively defined by:

- for every variable $x \in V_\Sigma$, $(x, \emptyset) \in \mathcal{T}_{\mathcal{S}(x)}$
- for every conditional declaration $(S(t), C) \in \mathcal{L}$, $(t, C) \in \mathcal{T}_S$
- for every conditional well sorted term $(t, C) \in \mathcal{T}_S$ $(S \neq \top)$, substitution $\sigma := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, with $(t_i, C_i) \in \mathcal{T}_{\mathcal{S}(x_i)}$ for all i, $(\bigcup_i C_i) \subseteq D$ for some finite set of literals D, $(t\sigma, C\sigma \cup D) \in \mathcal{T}_S$
- for every term t we have $(t, \emptyset) \in \mathcal{T}_\top$

We define $\mathcal{T}_S^\emptyset := \{(t, \emptyset) \mid (t, \emptyset) \in \mathcal{T}_S\}$. The relation $\sqsubseteq$ denotes the subsort relationship. If S and T are sorts, then we define $S \sqsubseteq T$ iff there exists a variable x with $\mathcal{S}(x) = S$ and $x \in \mathcal{T}_T$. A sort S is called *empty* if $\mathcal{T}_S$ does not contain a ground term. A conditional substitution $\sigma' = (\sigma, C)$ is called *conditional well sorted* if for every $x_i \in DOM(\sigma)$, $(x_i\sigma, C_i) \in \mathcal{T}_{\mathcal{S}(x_i)}$ for some C_i and $(\bigcup_i C_i) \subseteq C$. The composition of two cws. substitutions can be computed by $\tau'\sigma' := (\tau\sigma, K\sigma \cup C)$, where $\sigma' = (\sigma, C)$, $\tau' = (\tau, K)$. The result of the composition is again a cws. substitution. Thus the set of all cws. substitutions forms a monoid.

The relation $\sqsubseteq$ and the test whether a sort is empty can be decided in linear time with respect to the number of declarations in $\mathcal{L}$. The question whether a term t is included in $\mathcal{T}_S$ for some sort S is decidable in at most $O(n^2 * m)$ time where $m = |L|$ and n is the number of symbols in t [Weidenbach, 1993]. This is important because these relations are frequently needed in sorted unification and for the application of the resolution rule.

Example: Applying Definition 2 to Δ' with $\mathcal{L} = \{(1)1, (2)1, (3)1\}$, $\mathcal{T}_{Man} = \{(peter, \emptyset), (x_1, \emptyset), (x_2, \emptyset), \ldots\}$ where $\mathcal{S}(x_i) = Man$ and $\mathcal{T}_{Human} = \mathcal{T}_{Man} \cup \{(father^i(peter), \emptyset), (father^i(y_j), \emptyset), (y_1, \emptyset), (y_2, \emptyset), \ldots\}$ where $\mathcal{S}(y_i) = Human$ and $Man \sqsubseteq Human$, $Man \sqsubseteq \top$, $Human \sqsubseteq \top$.

Applying Definition 2 to Δ'' with $\mathcal{L} = \{(1)1\}$, $\mathcal{T}_{Man} = \{(x_1, \emptyset), (x_2, \emptyset), \ldots, (peter, \{Woman(peter)\})\}$ with $\mathcal{S}(x_i) = Man$. $\mathcal{T}_{Woman} = \{(y_i, \emptyset)\}$ with $\mathcal{S}(y_i) = Woman$. The sort *Woman* is empty. If we choose $\mathcal{L} = \{(1)2\}$ the sort *Man* would be empty and $\mathcal{T}_{Woman}$ contains the term $(peter, \{Man(peter)\})$. For a complete resolution calculus it suffices to choose exactly one of the literals (1)1 or (1)2.

The algorithm COMP compiles negative sort into variable restrictions. This is done with respect to the sort theory $\mathcal{L}$ consisting of all declarations which occur in a unit clause. COMP is a generalization of the sort generating algorithm SOGEN suggested by Schmidt-Schauß [Schmidt-Schauß, 1989].

Algorithm 3 (COMP) The input of the algorithm is a database of clauses Λ.

(1) Select all declarations occurring in unit clauses for $\mathcal{L}$.

(2) For every clause $C = \neg T(x) \vee C'$:
   (a) if $\mathcal{S}(x) \sqsubseteq T$ and $\mathcal{S}(x)$ is not empty or $x \in \mathcal{V}(C')$, delete $\neg T(x)$
   (b) if $T \sqsubseteq \mathcal{S}(x)$ and $x \in \mathcal{V}(C')$ replace C by $C'\{x \mapsto y_T\}$
   (c) if a new unit clause containing a declaration is derived, add the declaration to $\mathcal{L}$.

COMP is an efficient algorithm in at most $O(m^2)$ time where $m = |\Lambda|$ for some database Λ.

Example: Applying COMP to Δ results in Δ'. The only declaration occurring in a unit clause is $Man(peter)$, whence $\mathcal{L} = \{(1)1\}$ (Step 1 of COMP). By Definition 2 the sort *Man* is not empty and *Human* is empty. Step 2 of the algorithm is now applicable to (2)1 because $Man \sqsubseteq \top$. A new unit clause is derived and the declaration $Human(x_{Man})$ is added to $\mathcal{L}$. Now the sort *Human* is not empty and Step 2 is applicable to (3)1, (4)1, and (4)2 of Δ resulting in Δ'.

2.3 Sorted Resolution

For the definition of the sorted unification procedure GSOUP (General Sorted Unification Procedure) the standard notions of unification theory (see [Siekmann, 1989]) are used. The input of the unification procedure is a unification problem $\Gamma = \{s_1 = t_1, \ldots, s_n = t_n\}$. Γ is called solved if it has the form $\Gamma = \{x_1 = t_1, \ldots, x_n = t_n\}$, where $x_i \neq x_j$ for $i \neq j$, $x_i \notin \mathcal{V}(t_j)$ for all i and j, and $t_i \in \mathcal{T}_{\mathcal{S}(x_i)}$ for all i.

Definition 4 (GSOUP) The following five sorted rules (see Table 1) and the six standard rules Tautology, Decomposition, Application, Orientation, Clash, and Cycle (e.g. see [Siekmann, 1989]) are applied to the unification problem Γ until it is solved or the problem is found to be unsolvable.

In order to compute a cws. substitution from a solved unification problem, we have to do the following. Let $\Gamma = \{x_1 = t_1, \ldots, x_n = t_n\}$ be the solved unification problem, then $\sigma := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is the corresponding unifier. $\sigma' := (\sigma, C)$ is a cws. mgu if we have $(t_i, C_i) \in \mathcal{T}_{\mathcal{S}(x_i)}$ for all i and $C = \bigcup_i C_i$. From a solved unification problem it may be possible to compute several (but only finitely many) cws. mgu's.

Sorted Fail	$\frac{\{x = t\} \cup \Gamma}{\text{STOP.FAIL}}$ if there exists a variable $y \in \mathcal{V}(\{x = t\} \cup \Gamma)$ such that $\mathcal{S}(y)$ is empty or if $t \notin \mathcal{T}_{\mathcal{S}(x)}$ and none of the rules Subsort, Common Subsort, Weakening VT, and Weakening VV are applicable.
Subsort	$\frac{\{x = y\} \cup \Gamma}{\{y = x\} \cup \Gamma}$ if $x \in \mathcal{T}_{\mathcal{S}(y)}$ and $y \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$
Common Subsort	$\frac{\{x = y\} \cup \Gamma}{\{x = z\} \cup \{y = z\} \cup \Gamma}$ if $\mathcal{S}(x) = S$, $\mathcal{S}(y) = T$, $\mathcal{S}(z) = R$, $x \notin \mathcal{T}^{\emptyset}_{T}$ and $y \notin \mathcal{T}^{\emptyset}_{S}$ and R is a maximal sort with $R \sqsubseteq S$, $R \sqsubseteq T$
Weakening VT	$\frac{\{x = f(t_1, \ldots, t_n)\} \cup \Gamma}{\{x = f(t_1, \ldots, t_n)\} \cup \{t_1 = s_1, \ldots, t_n = s_n\} \cup \Gamma}$ if $\mathcal{S}(x) = S$, $x \notin \mathcal{V}(f(t_1, \ldots, t_n))$, $f(t_1, \ldots, t_n) \notin \mathcal{T}^{\emptyset}_{S}$ and there is a conditional declaration $(S'(f(s_1, \ldots, s_n)), C) \in \mathcal{L}$, $S' \sqsubseteq S$
Weakening VV	$\frac{\{x = y\} \cup \Gamma}{\{x = f(s_1, \ldots, s_n)\} \cup \{y = f(t_1, \ldots, t_n)\} \cup \{s_1 = t_1, \ldots, s_n = t_n\} \cup \Gamma}$ if $\mathcal{S}(x) = S$, $\mathcal{S}(y) = T$, $y \notin \mathcal{T}^{\emptyset}_{S}$ and $x \notin \mathcal{T}^{\emptyset}_{T}$ and there are conditional declarations $(S'(f(s_1, \ldots, s_n)), C) \in \mathcal{L}$, $S' \sqsubseteq S$ and $(T'(f(t_1, \ldots, t_n)), D) \in \mathcal{L}$, $T' \sqsubseteq T$

Table 1: The Sorted Rules of GSOUP

Lemma 5 (GSOUP is Sound and Complete) If σ' is a conditional substitution computed by the procedure, then σ' solves T and σ' is conditional well sorted. If γ' is a cws. ground substitution solving T, then there is a cws. substitution σ' computed by the unification procedure and a cws. substitution λ', such that $\sigma'\lambda' = \gamma'$ with respect to $\mathcal{V}(\Gamma)$. If in the rule Sorted Fail, the condition concerning empty sorts is erased, the unification procedure is complete with respect to all cws. substitutions.

Lemma 6 Sorted unification is undecidable and of unification type infinitary.

Lemma 6 is a worst case result. For restricted sort theories better results are known (see [Schmidt-SchauB, 1989; Uribe, 1992; Cohn, 1992; Weidenbach, 1993]). GSOUP can be implemented in a way such that it has the appropriate complexity properties for restricted sort theories.

For the resolution rule it is necessary to keep track of the sorts S_i occurring with the variables of the parent clauses but not with the variables of the resolvent. Every disappearing S_i must be non-empty and the corresponding conditions have to be added to the resolvent. For example, building a resolvent between $\neg Love(x_{Man}, mary)$ and $Love(x_{Man}, mary)$ by using $\mathcal{L} = \{(Man(peter), \{Woman(peter)\})\}$ results in the clause $Woman(peter)$. Note that the unification problem $\Gamma = \{x_{Man} = z_{Man}\}$ is solved, $(z_{Man}, \emptyset) \in \mathcal{T}_{Man}$, and the only cws. unifier is $(\{x_{Man} \mapsto z_{Man}\}, \emptyset)$.

Definition 7 (Resolution and Factorization) The rules are

$$\text{Resolution} \quad \frac{P(t_1, \ldots, t_n) \vee C_1 \quad \neg P(s_1, \ldots, s_n) \vee C_2}{C_1\sigma \vee C_2\sigma \vee D \vee E}$$

where $\sigma' = (\sigma, D)$ is a cws. mgu of $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$ and E is a set of literals (conditions) which guarantees the sorts attached to variables occurring in the codomain of σ but not in $C_1\sigma \vee C_2\sigma \vee D$ to be non-empty.

$$\text{Factorization} \quad \frac{P(t_1, \ldots, t_n) \vee P(s_1, \ldots, s_n) \vee C}{P(t_1, \ldots, t_n)\sigma \vee C\sigma \vee D}$$

where $\sigma' = (\sigma, D)$ is a cws. mgu of $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$.

The soundness of the rules follows immediately from their form and the soundness of the unification algorithm [Weidenbach, 1991]. The set E of non-emptiness conditions can be computed using Definition 2 for ground terms.

Theorem 8 (Completeness Theorem) Let Δ be a clause database. We choose $\mathcal{L} := \{(S_i(t_i), \{L_{i,1}, \ldots, L_{i,n_i}\})\}$ such that for each clause $C_i \in \Delta$ only containing declarations we choose exactly one declaration $S_i(t_i)$ with $C_i = S_i(t_i) \vee L_{i,1} \vee \ldots \vee L_{i,n_i}$.

If Δ is unsatisfiable there exists a derivation of the empty clause using resolution and factorization. The set $\mathcal{L}$ must be updated every time a new declaration clause is derived.

3 Applications

The first application is a puzzle called "The Lion and the Unicorn" which can be found in one of Smullyan's books [Smullyan, 1978] and was previously discussed by Ohlbach, Schmidt-SchauB, and Weidenbach [Ohlbach and Schmidt-Schaus8, 1985; Weidenbach and Ohlbach, 1990]. The lion and the unicorn are strange creatures which lie on certain days of the week and tell the truth on the other days. The lion lies on Mondays, Tuesdays and Wednesdays and the unicorn lies on Thursdays, Fridays and Saturdays. In order to solve the puzzle it must be figured out what day we have if they both make the statement "yesterday was one of my lying days".

In standard first-order logic the example is expressed by 47 clauses with 109 literals. After the application of COMP 47 clauses with 55 literals are left. The example is complicated because it contains recursive clauses and sort literals occur together with three place literals.The query is

$$\exists x_D\,(Lies(lion, x_D, f(x_D)) \wedge Lies(uni, x_D, f(x_D)))$$

which states that there is a day *x* when both the lion and the unicorn say that yesterday was one of the days they lie. The query can be further complicated by increasing the nesting depth of the function / (for "yesterday").

Using existing sorted approaches not all sorts can be processed by sorted unification due to the restrictions imposed on the sort theory. A consequence is that solving the puzzle with these approaches compares to solving the problem with the standard resolution method. We solved the problem on a Sparc ELC workstation with 16MB using OTTER 2.2 and STOP 0.9 (Sorted Theorem Prover- a first prototype implementation of the new resolution method with sorts.). The table shows the number of clauses generated and the time spent by the provers in order to derive the query depending on the nesting depth of the function / in the query:

Nesting Depth	OTTER		STOP	
	#clauses	#seconds	#clauses	#seconds
1	64600	600	22	6
2	181486	1800	18	9
3	1367151	31000	11	10
4	fail	fail	10	12
5	fail	fail	10	13

A second application concerns abduction. There are many applications for automated reasoning where abductive reasoning has to be applied. Of course, when using abduction one is interested in finite representations of answers. Demolombe and Farinas [Demolombe and Farinas, 1991] proposed an inference rule, called L-inference, which can be used to automatize abductive reasoning. The L-inference rule is a special resolution rule. I will show that extending this rule with sorts allows the generation of finite answers to abductive queries, where the standard rule without sorts computes infinite answers. Assume the query "Which assumptions guarantee that *Love{x,y)* holds ?" is asked to the database Δ (see Section 1). Applying the standard rule amounts to compute all resolvents between the clause

$$L(x, y) \vee \neg Love(x, y)$$

and the clauses in Δ. The literal *L{x,y)* is an extra literal introduced by Demolombe and Farinas method which is used to collect the instantiations made to the variables in the query literal. It is possible to derive infinitely many non-redundant clauses, e.g. clauses of the form

$$L(father^i(x), father^j(y)) \vee \neg Human(x) \vee \neg Human(y)$$

Applying L-inference to Δ' the only possible resolution step uses clause (4) and results in

$$L(y_{Human}, z_{Human})$$

Thus the sorted answer is that two objects love each other if they are humans. This is the natural answer. The example demonstrates that the resolution method with sorts terminates in more cases and derives natural answers.

4 Discussion

The approach of Schmidt-SchauB [Schmidt-SchauB, 1989] extends Walther's work [Walther, 1987]. The resolution method with sorts is an extension of Schmidt-SchauB's approach. If all declarations occur in unit clauses and all negative sort literals can be compiled by COMP then all conditional parts of cws. terms are empty. If in addition all sorts *S* are non-empty, the resolution method with sorts is the same as Schmidt-SchauB's order-sorted resolution calculus.

The framework presented by Frisch [Frisch, 1991] is more restricted than my approach. He separates the sort theory from the database. A sort theory is built from sort literals only and must be equivalent to a Horn theory. Sort literals are not allowed to occur with other literals in the database. As a consequence all sorts are a priori assumed to be non-empty. In addition, he does not answer how the sorted reasoning is to be performed, which in my approach is done by the notion of cws. terms and GSOUP.

The logic of Cohn [Cohn, 1992] also does not give an answer how sorted reasoning has to be performed (an oracle is assumed). He imposes no restrictions on the occurrence of sort literals, but does not incorporate declarations occurring together with other literals in the sorted reasoning process. This leads to a calculus which consists of more inference rules than the usual resolution and factoring rule. In addition, unifiers which are not well sorted (with respect to Definition 2) have to be considered for the inference rules also. Therefore, the resolution method with sorts is much more restrictive in the number of applicable inference steps.

The same arguments that hold for Cohn, apply to the work of Beierle et al. [Beierle *et* al., 1992]. Although he gives a fully developed calculus, his extended order-sorted unification algorithm is nothing else than unsorted unification plus the collection of a negative sort literal for each component of the unsorted unifier. Hence, his resolution method is also less restrictive in the number of applicable inference steps than the method proposed in this paper.

Compared to the work of Weidenbach and Ohlbach [Weidenbach and Ohlbach, 1990] I switched from a static processing of the sort theory to a dynamic one. As a consequence the number of declarations considered by the unification algorithm has been reduced significantly. The new method also needs fewer inference rules. Thus the method presented in this paper is more restrictive in the number of applicable inference steps than our own previous work.

The frameworks of Stickel [Stickel, 1985] and Biirckert [Biirckert, 1991] propose methods for integrating theories into the resolution method. The resolution method with sorts is not an instance of these frameworks, because it is assumed that the theory is static during the deduction process. The sort theory C changes during the deduction process. This allows for less inference rules and a more restricted calculus in the number of applicable inference steps.

Sorted unification in our resolution method is undecidable and of type infinitary (see Lemma 6). It is often argued that unification procedures having this property are not useful. But the result means that the sort theory processed by sorted unification has this properties in general. Hence the question is whether it is more efficient to process the theory by the unification procedure or by standard resolution. The notion of cws. terms prevents sorted unification from performing inference steps which are performed by standard resolution. In general there may be infinitely many such steps. Thus using GSOUP is much more efficient than weaker unification algorithms combined with resolution.

5 Summary

Every standard first-order database can be thought of as a database with sorts where all variables have sort "any" (T). The algorithm COMP can be used to compile negative sort literals into variable restrictions. If COMP can save sort literals, applying the resolution method with sorts is more efficient than standard resolution. If COMP cannot save literals (all variables have sort T) the resolution method with sorts derives exactly the same clauses than the standard resolution method.

The resolution method with sorts generalizes and improves existing approaches to sorted reasoning.

The resolution method with sorts terminates in more cases than the standard resolution method. This is useful for abductive reasoning, for example.

Acknowledgements

I would like to thank Alan M. Frisch, Renate Schmidt, and the reviewers for many helpful comments on this paper.

References

[Beierle et a/., 1992] C. Beierle, U. Hedstiick, U. Pletat, and J. Siekmann. An order-sorted logic for knowledge representation systems. Artificial Intelligence, 55:149-191, 1992.

[Biirckert, 1991] H.J. Biirckert. A Resolution Principle for a Logic with Restricted Quantifiers, volume 568 of Lecture Notes in Artificial Intelligence. Springer Verlag, 1991.

[Cohn, 1992] A.G. Cohn. A many sorted logic with possibly empty sorts. In 11th International Conference on Automated Deduction, CADE-11, LNCS 607, pages 633-647. Springer Verlag, 1992.

[Demolombe and Farinas, 1991] R. Demolombe and L. Farinas. An inference rule for hypothesis generation. In Proceedings of the Twelfth International Conference on Artificial Intelligence, pages 152-157. Morgan Kaufmann, 1991.

[Frisch, 1991] A.M. Frisch. The substitutional framework for sorted deduction: fundamental results on hybrid reasoning. Artificial Intelligence, 49:161-198, 1991.

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

[Schmidt-SchauB, 1989] M. Schmidt-SchauB. Computational aspects of an order sorted logic with term declarations, volume 395 of Lecture Notes in Artificial Intelligence. Springer Verlag, 1989.

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

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

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

[Uribe, 1992] T.E. Uribe. Sorted unification using set constraints. In 11th International Conference on Automated Deduction, CADE-11, LNCS 607, pages 163-177. Springer Verlag, 1992.

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

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

[Weidenbach, 1991] C. Weidenbach. A sorted logic using dynamic sorts. MPI-Report MPI-I-91-218, Max-Planck-Institut fur Informatik, Saarbriicken, December 1991.

[Weidenbach, 1993] C. Weidenbach. Unification in sort theories and its applications. MPI-Report MPI-1-93-211, Max-Planck-Institut fur Informatik, Saarbriicken, March 1993.