Sorted Unification and its Applications to Automated Theorem Proving

(Extended Abstract)

Christoph Weidenbach
Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
email: weidenb@mpi-sb.mpg.de
phone: +49-681-9325221
fax: +49-681-9325299

June 21, 1996

1 Introduction

The task of unification is always the search for a substitution σ for a unification problem $t \approx s$ such that $t\sigma = s\sigma$. For example, the standard[1] unification problem

$$x \approx f(y)$$

has the obvious standard solution $\sigma = \{x/f(y)\}$. In the presence of sort theories[2], we attach sorts to the variables and require, in addition to standard unification, that the assignments of σ respect the sort restrictions. For example, consider the sort theory[3]

$$\mathcal{L} = \{S(f(g(x_{\{S\}}))), T(f(y_{\{T\}})), T(g(y_{\{T\}})), S(a), T(a)\}$$

and a modification of the above unification problem where we have attached sorts to the variables.

$$x_{\{S,T\}} \approx f(y_{\{T\}})$$

In our approach, sorts are sets of monadic predicates. The denotation of the monadic predicates is fixed by the sort theory and sorts restrict the domain of variables to the intersection of the denotations of the monadic predicates. In general, sorted terms are schemas for infinitely many well-sorted instances, with respect to the sort theory. For

[1] With the term *standard* we refer to notions without sorts.

[2] Sort theories correspond to order-sorted signatures with arbitrary term declarations.

[3] Variables in sort theories are always implicitly universally quantified.

example, the denotations of the sorts $\{S\}$ and $\{T\}$ and their intersection $\{S,T\}$ with respect to $\mathcal{L}$ are[4]

$$\begin{aligned}
\mathcal{T}_{\{S\}} &= \{a, x_{\{S\}}, z_{\{S,T\}}, \ldots, [fg]^i(a), [fg]^i(x_{\{S\}}), [fg]^i(z_{\{S,T\}}), \ldots\} \\
\mathcal{T}_{\{T\}} &= \{a, y_{\{T\}}, z_{\{S,T\}}, \ldots, \{f,g\}^i(a), \{f,g\}^i(y_{\{T\}}), \{f,g\}^i(z_{\{S,T\}}), \ldots\} \\
\mathcal{T}_{\{S,T\}} &= \{a, z_{\{S,T\}}, \ldots, [fg]^i(a), [fg]^i(y_{\{S,T\}}), \ldots\}
\end{aligned}$$

where $\{f,g\}^i(t)$ is an abbreviation for all terms of the form $f(t), g(t), f(f(t)), f(g(t)), g(g(t)), g(f(t)), \ldots$ and $[fg]^i(t)$ is an abbreviation for all terms of the form $f(g(t))$, $f(g(f(g(t)))), \ldots$ Every set $\mathcal{T}_Q$ contains infinitely many variables of the respective subsorts. Now sorted unification of a problem $t \approx s$ computes a set of most general unifiers σ such that $t\sigma = s\sigma$ and the assignments of the unifier are well-sorted, e.g., if $x_{\{S,T\}}\sigma = t$ for a variable $x_{\{S,T\}}$, then we require $t \in \mathcal{T}_{\{S,T\}}$. For the above example, the standard unifier $\{x_{\{S,T\}}/f(y_{\{T\}})\}$ is not well-sorted, since $f(y_{\{T\}}) \notin \mathcal{T}_{\{S\}}$ with respect to $\mathcal{L}$. The variable $y_{\{T\}}$ has to be *weakened* (instantiated) in order to obtain a well-sorted unifier. The above unification problem is solved by the well-sorted mgu (most general unifier) $\sigma = \{x_{\{S,T\}}/f(g(z_{\{S,T\}})), y_{\{T\}}/g(z_{\{S,T\}})\}$.

Depending on the syntactic form of the terms in the sort theory, the complexity of sorted unification, i.e., the complexity of the problem whether a given unification problem has a solution, as well as the number of most general unifiers differs. In general, sorted unification is undecidable and produces infinitely many unifiers, but there are many decidable classes known (see Figure 3).

The application of sorted unification to automated theorem proving is the main motivation for our approach to sorted unification. Sorted unification has been applied both to the resolution [16] (superposition [1]) and the free variable tableau calculus [6, 17]. The sort information needs not to be given in advance, but can be extracted from the clauses. For example, using the relativization [10, 17] rules for sorted formulae we know that the standard clause

$$\neg T(x) \vee \neg S(x) \vee \neg R(x,y) \vee S(f(x))$$

is logically equivalent to the clause

$$\neg R(x_{\{S,T\}}, y) \vee S(f(x_{\{S,T\}}))$$

where sorts are attached to variables. Negative monadic literals with a variable as their argument code sort restrictions on the variable. The variable y has top-sort, i.e., y is interpreted like a standard variable, ranging over the whole domain. After the transformation, we basically have to replace standard unification with sorted unification resulting in the sorted extension of the respective calculus. The problem remains how the sort theory is extracted from the clause set. We have shown that for the free variable tableau calculus, it is sufficient to consider the rigid sort theory built from the monadic atoms on the branch of the tableau we want to close [17]. Here rigid means that the variables contained in the selected sort theory are not universally quantified, but free variables. For resolution based calculi [16, 19], there are several possibilities how the sort theory can be extracted. Either

[4] For a precise definition, see Definition 2.1

we dynamically choose from each clause consisting of positive monadic literals only, one positive literal for the sort theory [16] or, if we impose ordering restrictions on the resolution calculus, we dynamically choose all declarations that are maximal in their respective clauses [19]. The applications of sorted unification to tableau and resolution based calculi show that it is sufficient to consider sort theories of the form of $\mathcal{L}$ where sorts may denote the empty set. In addition to the unification problem, the use of sorted unification in resolution and tableau requires a test of whether a given sort is non-empty, i.e., the denotation of the sort contains at least one ground term. For all sort theories we investigated so far, the (un)decidability of sorted unification implies the (un)decidability of the empty sort test.

In the sequel we will introduce sorted unification, present complexity results for different sort theories (Figure 3) and we will finish the paper by presenting some automated theorem proving experiments (Figure 4) that justify our approach to sorted unification.

2 Sorted Unification

We use the usual notation for terms, substitutions, etc. The set of all terms is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$ or $\mathcal{T}$ as a shorthand and the set of all ground terms is denoted by $\mathcal{T}(\mathcal{F})$. Let $\mathcal{S}$ be a finite set of monadic predicate symbols, also called *base sorts*. Then a *sort* T is a finite subset of $\mathcal{S}$, $T \in 2^{\mathcal{S}}$, denoting the intersection of the contained base sorts. For variables we assume a function $sort\colon \mathcal{V} \to 2^{\mathcal{S}}$ that attaches sorts to variables such that for each sort $T \in 2^{\mathcal{S}}$ there are infinitely many variables x with $sort(x) = T$. For $\emptyset \in 2^{\mathcal{S}}$, we write $\top$, the top sort. We write S for $\{S\}$ if the context causes no confusion. Usually, variables are annotated with their sort. We write x_S if $sort(x) = S$.

Syntactic properties of terms will play an important role for the complexity of sorted unification: A term t is called *elementary* if t is either a variable or a constant or of the form $f(x_1,\ldots,x_n)$ where all x_i are different. A term t is called *weak-elementary* if t is either a variable or a constant or of the form $f(t_1,\ldots,t_n)$ where each t_i is either a variable[5] or a ground term. The term t is called *linear* if any variable occurs at most once in t. It is called *semi-linear* if for any variable x occurring more than once in t, there is a subterm $f(s_1,\ldots,s_n)$ of t such that all occurrences of x are in $f(s_1,\ldots,s_n)$, each s_i has at most one occurrence of x and whenever x occurs in s_i and s_j we have $s_i = s_j$. A term t is called *pseudo-linear* if all non-linear variable occurrences in t occur at the same depth.

An atom $S(t)$ is called a *declaration* if $S \in \mathcal{S}$. It is called a *subsort declaration* if t is a variable and a *term declaration* otherwise. It is called a *function declaration* if t is elementary. A *sort theory* $\mathcal{L}$ is a finite set of declarations where all variables are implicitly assumed to be universally quantified. All following notions and definitions refer to a fixed, finite sort theory $\mathcal{L}$. The *size* of a sort theory $\mathcal{L}$ is $size(\mathcal{L}) = \sum_{S(t)\in\mathcal{L}}(size(t) + \sum_{x\in vars(t)} |sort(x)|)$.

Depending on the declarations occurring in $\mathcal{L}$, the following sort theories are distinguished: $\mathcal{L}$ is called *elementary* (*weak-elementary*) if every term in a term declaration

[5] Multiple occurrences of the same variable are allowed.

in $\mathcal{L}$ is elementary (weak-elementary). $\mathcal{L}$ is called *linear* (*semi-linear*, *pseudo-linear*) if every term occurring in a term declaration is linear (semi-linear, pseudo-linear). $\mathcal{L}$ is called *simple* if for every function symbol f there is at most one declaration of the form $S(f(t_1, \ldots, t_n))$ in $\mathcal{L}$. $\mathcal{L}$ is called *rigid* if the variables in $\mathcal{L}$ are interpreted to be free variables. Rigid sort theories are of particular interest if sorted unification is applied to free variable tableau [17].

Definition 2.1 *The set of well-sorted terms $\mathcal{T}_S$ of sort S is recursively defined by:*

1. $x \in \mathcal{T}_S$ *if* $S \subseteq sort(x)$.
2. $t \in \mathcal{T}_S$ *if* S *is a base sort and* $S(t) \in \mathcal{L}$.
3. $t\sigma \in \mathcal{T}_S$ *if* $t \in \mathcal{T}_S$ *and* $x\sigma \in \mathcal{T}_{sort(x)}$ *for all* $x \in dom(\sigma)$.
4. $t \in \mathcal{T}_{S_1 \cup \ldots \cup S_n}$ *if* $t \in \mathcal{T}_{S_i}$ *for all* i.

Note that there are only declarations for base sorts in $\mathcal{L}$. Case 4 also includes the case where $S = \top$, i.e., $n = 0$; then we have $\mathcal{T}_\top = \mathcal{T}$. $\mathcal{T}(\mathcal{F})_S$ is the restriction of $\mathcal{T}_S$ to ground terms. A sort S is called *empty* if there is no well-sorted ground term $t \in \mathcal{T}_S$, or equivalently if $\mathcal{T}(\mathcal{F})_S = \emptyset$. We always have $\mathcal{T}(\mathcal{F})_S \subset \mathcal{T}_S$. The binary relation $\sqsubseteq$ denotes the subsort relationship. If S and T are sorts, then we define $S \sqsubseteq T$ iff there exists a variable x_S with $x_S \in \mathcal{T}_T$. Note that if there exists one variable $x_S \in \mathcal{T}_T$, there are infinitely many variables of sort S in $\mathcal{T}_T$. The relation $S \sqsubseteq T$ implies $\mathcal{T}_S \subseteq \mathcal{T}_T$, but the relation $\mathcal{T}(\mathcal{F})_S \subseteq \mathcal{T}(\mathcal{F})_T$ neither implies $S \sqsubseteq T$ nor $\mathcal{T}_S \subseteq \mathcal{T}_T$. A substitution σ is *well-sorted* if for every $x \in dom(\sigma)$, $x\sigma \in \mathcal{T}_{sort(x)}$. It can be polynomially decided (in $size(\mathcal{L}) + size(\sigma)$) whether a substitution σ is well-sorted.

Now we extend standard unification to sorted unification. A unification problem $\Gamma = (s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n)$ is called *solved*[6], iff all s_i are variables which don't occur elsewhere in Γ and $t_i \in \mathcal{T}_{sort(s_i)}$ for all i. The sorted unification algorithm consists of the three don't care non-deterministic sorted rules Sorted Fail, Subsort and Weakening (see Figure 2) and the rules Tautology, Decomposition, Orientation, Application, Clash and Cycle of standard tree oriented unification [12, 17, 5, 7] (Figure 1). These combined rule sets are applied to a unification problem Γ until it is solved or $\top$, $\bot$ is derived.

In rule Subsort the condition $sort(x) \not\subseteq sort(y)$ does not imply $sort(y) \not\sqsubseteq sort(x)$. However, if $sort(y) \sqsubseteq sort(x)$ then $\mathcal{T}_{sort(x)} = \mathcal{T}_{sort(x) \cup sort(y)}$. Note that declarations must be well-sorted renamed, before they are used by the Weakening rule.

Due to the rule Weakening, sorted unification does not terminate, in general. The Weakening rule introduces terms from $\mathcal{L}$ with fresh variables. Therefore, the number of variables and the multiset of all term depths in the unification problem may increase. In addition, the number of sorted unsolved equations may increase, too. In general, sorted unification, as well as the problem whether a sort is empty, is undecidable. The Weakening rule, together with the other rules can simulate arbitrary computational processes.

[6] For simplicity we choose a tree like solved form. In general, dag solved forms [17] are necessary to establish the complexity results we will present, because tree solved forms grow exponentially in size of their unsolved input problems, in general.

Tautology

$t \approx t \wedge \Gamma \rightarrow \Gamma$

Orientation

$t \approx x \wedge \Gamma \rightarrow x \approx t \wedge \Gamma$

if $t \notin \mathcal{V}$

Decomposition

$f(t_1, \ldots, t_n) \approx f(s_1, \ldots, s_n) \wedge \Gamma \rightarrow t_1 \approx s_1 \wedge \ldots \wedge t_n \approx s_n \wedge \Gamma$

Application

$x \approx t \wedge \Gamma \rightarrow x \approx t \wedge \Gamma\{x/t\}$

if $x \notin \mathit{vars}(t)$ and $x \in \mathit{vars}(\Gamma)$

Clash

$f(t_1, \ldots, t_n) \approx g(s_1, \ldots, s_m) \wedge \Gamma \rightarrow \bot$

if $f \neq g$

Cycle

$x \approx t \wedge \Gamma \rightarrow \bot$

if $x \in \mathit{vars}(t)$ and $t \notin \mathcal{V}$

Figure 1: The Rules of Tree Standard Unification

Sorted Fail

$x \approx f(t_1, \ldots, t_n) \wedge \Gamma \rightarrow \bot$

if $\top \not\sqsubseteq \mathit{sort}(x)$ and there is no sort $\{T_1, \ldots, T_m\} \sqsubseteq \mathit{sort}(x)$ with declarations $T_j(f(s_{j,1}, \ldots, s_{j,n})) \in \mathcal{L}$

Subsort

$x \approx y \wedge \Gamma \rightarrow x \approx z \wedge y \approx z \wedge \Gamma$

if $\mathit{sort}(x) \not\sqsubseteq \mathit{sort}(y)$, z new to Γ, and $\mathit{sort}(z) = \mathit{sort}(x) \cup \mathit{sort}(y)$

Weakening

$x \approx t \wedge \Gamma \rightarrow x \approx s_1 \wedge t \approx s_i \wedge \Gamma$

if $t \notin \mathcal{V}$, $t \notin \mathcal{T}_{\mathit{sort}(x)}$, $\{T_1, \ldots, T_m\} \sqsubseteq \mathit{sort}(x)$ and for each T_j there is a declaration $T_j(s_j) \in \mathcal{L}$ such that t and the s_j share the same top symbol

Figure 2: The Rules of Sorted Unification

Sort Theory	Complexity	Unification Type
Simple, Elementary	linear	unitary
Elementary	NP-complete	finitary
Weak-Elementary	NP-complete	finitary
Linear	NP-complete	finitary
Semi-linear	NP-complete	finitary
Simple, Weak-Elementary	linear	unitary
Pseudo-Linear	decidable	infinitary
Rigid	decidable	finitary
General	undecidable	infinitary

Figure 3: Complexity of Sorted Unification

It can be shown that for every unification problem Γ and finite sort theory $\mathcal{L}$, there exists a minimal, correct and complete set of unifiers and that sorted unification is correct and complete with respect to the usual free model semantics [15].

Concerning the complexity of sorted unification, the results listed in Figure 3 are known [10, 14, 15, 18]. Linear and semi-linear sort theories can be reduced to weak-elementary sort theories. Since a minimal, correct and complete set of unifiers always exists, the unification type [12] characterizes mainly the cardinality of such sets. A sort theory is of unification type *unitary* if for every unification problem there is at most one well-sorted mgu. A sort theory is of unification type *finitary* if for every unification problem there are at most finitely many well-sorted mgus. A sort theory is of unification *infinitary* if there exists a unification problem that has infinitely many mgus.

The complexity results for sorted unification are related to results obtained for bottom up tree automata. Comon [4] was the first who pointed out the relationship between sorted unification and regular tree languages. The ground terms induced by elementary sort theories are regular tree languages. The ground terms of weak-elementary sort theories can be described by tree automata with equality constraints on direct subterms, as shown by Bogaert and Tison [2]. Recently, from results on tree automata presented by Caron et al. [3], it can be shown that the empty sort problem is decidable if all declarations satisfy an ordering restriction. We assume a partial ordering on the sort symbols and require that whenever a declaration contains non-linear occurrences of variables, the sorts attached to the variables in the declaration are strictly below the sort symbol of the declaration itself. For example for the declaration $S(f(x_T, g(x_T)))$ we would require that T is strictly below S in the ordering. However, a detailed comparison between the two approaches is beyond the scope of this paper.

We demonstrate the rules of sorted unification on the unification problem Γ and pseudo-linear sort theory $\mathcal{L}$ presented below. Remember that the class of pseudo-linear sort theories properly includes the classes of elementary, weak-elementary, linear and semi-linear sort theories. For example, the pseudo-linear declaration $S(f(g(x_S), h(x_S)))$ is pseudo-linear but not semi-linear. Consider the pseudo-linear sort theory

$$\mathcal{L} = \{S(g(x_S)), S(h(x_S)), T(g(x_S)), S(f(g(x_S), h(x_S))), S(f(g(y_S), f(y_S, x_S)))\}$$

and the unification problem

$$\Gamma = (z_S \approx f(v_T, w_S))$$

Γ is not solved because $f(v_T, w_S) \notin \mathcal{T}_S$. The rule Weakening is applicable using the two declarations $S(f(g(x_S), h(x_S)))$ and $S(f(g(y_S), f(y_S, x_S)))$, respectively:

$$\begin{array}{lcl} \Gamma_1 & = & (z_S \approx f(g(x_S), h(x_S)) \wedge f(v_T, w_S) \approx f(g(x_S), h(x_S))) \\ \Delta_1 & = & (z_S \approx f(g(y_S), f(y_S, x_S)) \wedge f(v_T, w_S)) \approx f(g(y_S), f(y_S, x_S))) \end{array}$$

The two problems can be transformed into standard solved form using the rules of Figure 1:

$$\begin{array}{lcl} \Gamma_2 & = & (z_S \approx f(g(x_S), h(x_S)) \wedge v_T \approx g(x_S) \wedge w_S \approx h(x_S)) \\ \Delta_2 & = & (z_S \approx f(g(y_S), f(y_S, x_S)) \wedge v_T \approx g(y_S) \wedge w_S \approx f(y_S, x_S)) \end{array}$$

The problem Γ_2 is already sorted solved. The problem Δ_2 contains the unsolved pair $w_S \approx f(y_S, x_S)$ to which we apply Weakening using the declarations $S(f(g(x'_S), h(x'_S)))$ and $S(f(g(y'_S), f(y'_S, x'_S)))$, respectively. We only show the pairs resulting from the Weakening step.

$$\begin{array}{lcl} \Delta_{31} & = & (w_S \approx f(g(x'_S), h(x'_S)) \wedge f(y_S, x_S) \approx f(g(x'_S), h(x'_S))) \\ \Delta_{41} & = & (w_S \approx f(g(y'_S), f(y'_S, x'_S)) \wedge f(y_S, x_S) \approx f(g(y'_S), f(y'_S, x'_S))) \end{array}$$

After standard unification we get

$$\begin{array}{lcl} \Delta_{32} & = & (w_S \approx f(g(x'_S), h(x'_S)) \wedge y_S \approx g(x'_S) \wedge x_S \approx h(x'_S)) \\ \Delta_{42} & = & (w_S \approx f(g(y'_S), f(y'_S, x'_S)) \wedge y_S \approx g(y'_S) \wedge x_S \approx f(y'_S, x'_S)) \end{array}$$

Now Δ_{32} is sorted solved and Δ_{42} contains the sorted unsolved pair $x_S \approx f(y'_S, x'_S)$, a variant of the pair $w_S \approx f(y_S, x_S)$ in Δ_2. Hence, sorted unification runs through a "cycle" producing infinitely many mgus for the initial problem Γ:

$$\begin{array}{lcl} \sigma_1 & = & \{z_S/f(g(x_S), h(x_S)), v_T/g(x_S), w_S/h(x_S)\} \\ \sigma_2 & = & \{z_S/f(g(g(x'_S)), f(g(x'_S), h(x'_S))), v_T/g(g(x'_S)), w_S/f(g(x'_S), h(x'_S))\} \\ \sigma_3 & = & \{z_S/f(g(g(g(x''_S))), f(g(g(x''_S)), f(g(x''_S), h(x''_S)))), v_T/g(g(g(x''_S))), \\ & & \quad w_S/f(g(g(x''_S)), f(g(x''_S), h(x''_S)))\} \\ & & \vdots \end{array}$$

3 Conclusion

For the resolution (superposition) case, the results of the previous section are implemented in the SPASS theorem prover [19]. The calculus implemented in SPASS is the superposition calculus of Bachmair and Ganzinger [1] extended with the mechanisms for sorts presented in this paper. A description of the prover as well as the calculus is contained in the SPASS distribution that can be obtained from `ftp.mpi-sb.mpg.de` in the directory `pub/SPASS`.

Figure 4 presents some automated theorem proving experiments with SPASS. They are intended to demonstrate the advantages of extending standard calculi with sort mechanisms. The results have been computed on Sparc Ultra 1/170E computers with SPASS Version 0.45. The first column of Figure 4 names the problem, the next three columns show the results obtained by SPASS where the sort information contained in the problem is not considered by the special sort mechanisms and the final three columns present the results where the sort information is processed by sorted unification. The specific results are presented giving the time in seconds needed by SPASS to solve the problem, the number of inferred clauses and the number of kept clauses, after reduction. The entry ∞ means that no proof was found in a time limit of one hour. The examples are the following: P38 is Pelletier's [9] problem No 38, RED is a real world problem coming from the verification of an algorithm on hash tables, ALC is a satisfiable problem stated in the knowledge representation language ALC [11], LCL is the problem LCL050 taken from the TPTP [13] benchmark library, PL7 is a scalable challenge problem stated by David Plaisted and REUSE is a real world problem coming from formal software reuse. All problems, except LCL, are contained in the SPASS distribution (see above). For space limitations we cannot explain these problems in detail, but the following is important: (i) These problems are not special "sort problems", i.e., they have not been designed for our developed sort concept. The opposite is the case. These problems have either been stated to be general benchmark problems or come from real world applications. However, we only chose problems containing sort information. On problems without usable sort information, the behavior of SPASS with and without the sort mechanisms is the same. The problems RED, PL7 and REUSE contain equality. (ii) Even without using the sort mechanisms of SPASS, SPASS is a powerful theorem prover on these examples. For example, although OTTER [8] solves LCL faster than SPASS, OTTER cannot solve P38. (iii) The examples are not selected in favor of the sort mechanisms. They represent the typical behavior of SPASS on problems that contain sort information.

The results justify the incorporation of sorts mechanisms in calculi for automated theorem proving: Using the sort mechanisms, SPASS always needs less time and infers fewer clauses and also the number of kept clauses is smaller than the corresponding result without the sort mechanisms.

References

[1] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[2] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. In A. Finkel and M. Jantzen, editors, *Proc. of 9th Annual Symposium on Theoretical Aspects of Computer Science, STACS92*, volume 577 of *LNCS*, pages 161–171. Springer, 1992.

Problem	No Sort Mechanisms			Sort Mechanisms		
	Time	Inferred	Kept	Time	Inferred	Kept
P38	0.09	87	45	0.06	77	44
RED	20.10	3555	1829	10.14	2608	1362
ALC	687.97	62692	14452	44.63	5872	2134
LCL	∞	∞	∞	148.64	9678	4297
PL7	56.22	18315	1743	18.75	7997	1323
REUSE	1129.69	18077	8509	314.54	7711	5081

Figure 4: Experiments with SPASS

[3] Anne-Cécile Caron, Hubert Comon, Jean-Luc Coquidé, Max Dauchet, and Florent Jacquemard. Pumping, cleaning and symbolic constraints solving. In Serge Abiteboul and Eli Shamir, editors, *Automata Languages and Programming. 21st International Colloquium, ICALP'94*, volume 820 of *LNCS*, pages 436–447. Springer, 1994.

[4] Hubert Comon. Inductive proofs by specification transformations. In *Rewriting Techniques and Applications, RTA-89*, volume 355 of *LNCS*, pages 76–91. Springer, 1989.

[5] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier Science Publishers, 1990.

[6] Melvin Fitting. *First-Order Logic.* Texts and Monographs in Computer Science. Springer, 1990.

[7] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.L. Lassez and G. Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, 1991.

[8] William McCune. Otter 2.0. In *10th International Conference on Automated Deduction, CADE-10*, volume 449 of *LNCS*, pages 663–664. Springer, 1990.

[9] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986. Errata: *Journal of Automated Reasoning*, 4(2):235–236,1988.

[10] Manfred Schmidt-Schauß. *Computational aspects of an order sorted logic with term declarations*, volume 395 of *LNAI*. Springer, 1989.

[11] Manfred Schmidt-Schauß and Gerd Smolka. Attributive concept description with complements. *Artificial Intelligence*, 48:1–26, 1991.

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

[13] Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Twelfth International Conference on Automated Deduction, CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence, LNAI*, pages 252–266, Nancy, France, June 1994. Springer.

[14] Tomás E. Uribe. Sorted unification using set constraints. In *11th International Conference on Automated Deduction, CADE-11*, volume 607 of *LNCS*, pages 163–177. Springer, 1992.

[15] Christoph Weidenbach. Unification in sort theories and its applications. *Annals of Mathematics and Artificial Intelligence*. To appear.

[16] Christoph Weidenbach. Extending the resolution method with sorts. In *Proc. of 13th International Joint Conference on Artificial Intelligence, IJCAI-93*, pages 60–65. Morgan Kaufmann, 1993.

[17] Christoph Weidenbach. First-order tableaux with sorts. *Journal of the Interest Group in Pure and Applied Logics, IGPL*, 3(6):887–906, 1995.

[18] Christoph Weidenbach. Unification in Pseudo-Linear Sort Theories is Decidable. In *13th International Conference on Automated Deduction, CADE-13*, 1996.

[19] Christoph Weidenbach, Bernd Gaede, and Georg Rock. SPASS & FLOTTER, Version 0.42. In *13th International Conference on Automated Deduction, CADE-13*, 1996.