Contextual Rewriting

Christoph Weidenbach
Patrick Wischnewski

MPI–I–2009–RG1–002 March 2009

Authors' Addresses

Christoph Weidenbach
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken
Germany

Patrick Wischnewski
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken
Germany

Publication Notes

This report is a preliminary version of an article intended for publication elsewhere.

Abstract

Sophisticated reductions are of particular importance for progress in automated theorem proving. We consider the powerful reduction rule *Contextual Rewriting* in connection with the superposition calculus. If considered in its most general form the applicability of contextual rewriting is not decidable. We develop an instance of contextual rewriting where applicability becomes decidable while preserving a great deal of its simplification power. A sophisticated implementation of the rule in SPASS reveals its application potential. Our contextual rewriting instance is feasible in the sense that it can be executed on the overall TPTP resulting in a gain of solved problems and new solutions to a number of problems that could not be solved by theorem provers so far.

Keywords

Rewriting, Contextual Rewriting

Contents

1 Introduction **2**

2 Contextual Rewriting **6**

3 Implementation **12**
3.1 Integration of Unit and Non-Unit Rewriting 16
3.2 Fault Caching 18

4 Results **20**
4.1 Results on the TPTP 20
4.1.1 Integrated Unit and Non-Unit Rewriting 21
4.1.2 Fault Caching 21
4.2 Application to the Example from the Introduction 23

5 Summary **25**

1 Introduction

In the superposition context, first-order theorem proving with equality deals with the problem of showing unsatisfiability of a finite set N of clauses. This problem is well-known to be undecidable, in general. It is semi-decidable in the sense that superposition is refutationally complete. The superposition calculus is composed of inference and reduction rules. Inference rules generate new clauses from N whereas reduction rules delete clauses from N or transform them into simpler ones while deleting the ancestors. If, in particular, powerful reduction rules are available, decidability of certain subclasses of first-order logic can be shown and explored in practice [4, 16, 17, 12, 11]. Hence, sophisticated reductions are of particular importance for progress in automated theorem proving. In this paper the reduction rule *Contextual Rewriting* is considered in combination with the superposition calculus [2]. Contextual rewriting extends rewriting with unit equations to rewriting with full clauses containing a positive orientable equation. In order to apply such a clause for rewriting, all other literals of that clause have to be entailed by the context of the clause to be rewritten and potentially further clauses from a given clause set N. Hence, the name contextual rewriting.

For a first, simple example consider the two clauses

$$P(x) \to f(x) \approx x \qquad S(g(a)), a \approx b, P(b) \to R(f(a))$$

where we write clauses in implication form [29]. Now in order to rewrite $R(f(a))$ in the second clause to $R(a)$ using the equation $f(x) \approx x$ of the first clause with matcher $\sigma = \{x \mapsto a\}$, we have to show that $P(x)\sigma$ holds in the context of the second clause $S(g(a)), a \approx b, P(b)$, i.e., $\models S(g(a)), a \approx b, P(b) \to P(x)\sigma$. This obviously holds, so we can replace $S(g(a)), a \approx b, P(b) \to R(f(a))$ by $S(g(a)), a \approx b, P(b) \to R(a)$ via a contextual rewriting application of $P(x) \to f(x) \approx x$.

More general, contextual rewriting is the following rule:

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C = (\Gamma_2 \rightarrow \Delta_2)[u[s\sigma] \approx v]}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ (\Gamma_2 \rightarrow \Delta_2)[u[t\sigma] \approx v]\end{array}}$$

where $(\Gamma_2 \rightarrow \Delta_2)[u[s\sigma] \approx v]$ expresses that $u[s\sigma] \approx v$ is an atom occurring in Γ_2 or Δ_2 and u contains the subterm $s\sigma$. Contextual rewriting reduces the subterm $s\sigma$ of u to $t\sigma$ if, among ordering restrictions, the following conditions are satisfied

$$\begin{array}{c} N_C \models \Gamma_2 \rightarrow A \text{ for all } A \text{ in } \Gamma_1\sigma \\ N_C \models A \rightarrow \Delta_2 \text{ for all } A \text{ in } \Delta_1\sigma \end{array}$$

where N is the current clause set, $C, D \in N$, and N_C denotes the set of clauses from N smaller than C with respect to a reduction ordering $\prec$, total on ground terms. Reduction rules are labeled with an $\mathcal{R}$ and are meant to replace the clauses above the bar by the clauses below the bar. Both side conditions are undecidable, in general. Therefore, in order to make the rule applicable in practice, it must be instantiated such that eventually these two conditions become effective. This is the topic of this paper.

For a more sophisticated, further motivating example, consider the following clause set. It can be finitely saturated using contextual rewriting but not solely with less sophisticated reduction mechanisms such as unit rewriting or subsumption.

Let i, q, r, f be functions, a, b be constants and x_1, x_2, x_3, x_4, y_1 be variables and let $r \succ f \succ q \succ i \succ b \succ a \succ nil$ using the KBO with weight 1 for all function symbols and variables.

1: $\rightarrow q(nil) \approx b$

2: $i(x_1) \approx b, q(y_1) \approx b \rightarrow q(r(x_1, y_1)) \approx b$

3: $i(x_1) \approx b, q(y_1) \approx b \rightarrow q(f(x_1, y_1)) \approx a$

4: $i(x_1) \approx b, q(y_1) \approx b, i(x_3) \approx b \rightarrow$
$r(x_3, f(x_1, y_1)) \approx f(x_1, r(x_3, y_1))$

5: $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, b \approx a \rightarrow$
$y_1 \approx nil, q(f(x_1, f(x_2, r(x_3, y_1)))) \approx b$

If we apply superposition right between clause 4 and clause 5 on the term $q(f(x_1, f(x_2, r(x_3, y_1))))$ we obtain the clause

6: $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, i(x_4) \approx b, q(y_1) \approx b, q(f(x_4, y_1)) \approx b, b \approx a \rightarrow$
$f(x_4, y_1) \approx nil,\ q(f(x_1, f(x_2, f(x_4, r(x_3, y_1))))) \approx b$

which is larger (both in the ordering and the number of symbols) than clause 5. Applying superposition between clause 4 and clause 6 yields an even larger clause. Repeating the superposition inference between clause 4 and these clauses creates larger and larger clauses. All those clauses cannot be simplified by unit rewriting and are not redundant with respect to subsumption [29]. Hence, the exhaustive application of the superposition calculus does not terminate on this clause set. Furthermore, none of the reductions which have been implemented so far in SPASS and in any other system we are aware of, can reduce clause 5. However, with contextual rewriting we can reduce clause 5 using clause 3 to

$$7\colon i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, b \approx a \to y_1 \approx nil, a \approx b.$$

Clause 7 is a tautology and can be reduced to true. Then the set is saturated since no further superposition inference is possible. In order to apply contextual rewriting to clause 5 using clause 3 we have to verify the side conditions

$$N_C \models i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, b \approx a \to i(x_1) \approx b$$

and

$$N_C \models i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, b \approx a \to \\ q(f(x_2, r(x_3, y_1))) \approx b.$$

The first condition holds trivially and the latter follows from clause 3 and clause 2 which are both smaller than clause 5. For more details see Section 4.2. This example already shows that the class of clause sets that can be finitely saturated with contextual rewriting is strictly larger than the class of clause sets that can be finitely saturated by unit rewriting, non-unit rewriting or local contextual rewriting [29, 30].

Contextual rewriting was first implemented in the SATURATE system [20, 13] but never matured. It turned out to be very useful for a bunch of examples, but the rule has to be turned off in general, because often the provers does not return in reasonable time from even a single contextual rewriting application test. This is partly due to a straight forward naive implementation, compared to the techniques presented in our paper, and a more general setting where the ordering constraints of the rule are not a priori calculated but inherited through ordering constraints.

In this work we present an instance of contextual rewriting that is useful, e.g., it reduces the above clause set, its application is decidable and it is also feasible in practice. We tested our implementation on all problems of the *TPTP* library version 3.2.0 [27]. Compared to our first implementation of the rule [32] the results of this paper lead to a performance where we win significantly more problems on the overall TPTP than we lose while keeping the positive results on hard problems. In particular, we solve 6 problems from the TPTP that no other reported system could solve before. The gained performance is due to a tight incorporation of contextual rewriting with unit and non-unit rewriting and a new caching technique (Chapter 3).

The paper is now organized as follows. In Section 2 we develop our instance of contextual rewriting and present its implementation in SPASS in Section 3. The final section, Section 4, discusses experimental results, both on the TPTP and on the above example.

2 Contextual Rewriting

We consider first-order logic with equality using notation from [29]. We write clauses in the form $\Gamma \rightarrow \Delta$ where Γ and Δ are multi-sets of atoms. The atoms of Γ denote negative literals while the atoms of Δ denote the positive literals. A substitution σ is a mapping from the set of variables to the set of terms such that $x\sigma \neq x$ for only finitely many variables x. The reduction rules, in particular the contextual rewriting rule, are defined with respect to a well-founded reduction ordering $\prec$ on terms that is total on ground terms. This ordering is then lifted to literals and clauses in the usual way [29]. A term s is called *strictly maximal* in $\Gamma \rightarrow \Delta$ if there is no different occurrence of a term in $\Gamma \rightarrow \Delta$ that is greater or equal than s with respect to $\prec$.

Contextual rewriting is a sophisticated reduction rule originally introduced in [2] that generalizes unit rewriting and non-unit rewriting [29]. It is an instance of the standard redundancy notion of superposition. A clause C is called *redundant* in a clause set N if there exist clauses $C_1, \ldots, C_n \in N$ with $C_i \prec C$ for $i \in \{1, \ldots, n\}$, written $C_i \in N_C$, such that $C_1, \ldots, C_n \models C$. The clause C is implied by smaller clauses from N. This condition can actually be refined to grounding substitutions: C is redundant if for all grounding substitutions σ for C there are ground instances $C_i\sigma_i$ of clauses $C_i \in N$ such that $C_i\sigma_i \prec C\sigma$, written $C_i\sigma_i \in N_{C\sigma}$, and $C_1\sigma_1, \ldots, C_n\sigma_n \models C\sigma$. Reduction rules are marked with an $\mathcal{R}$ and their application replaces the clauses above the bar with the clauses below the bar.

Definition 1 (Contextual Rewriting [2]) *Let N be a clause set, $C, D \in N$, σ be a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C = \Gamma_2, u[s\sigma] \approx v \rightarrow \Delta_2}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ C' = \Gamma_2, u[t\sigma] \approx v \rightarrow \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C = \Gamma_2 \rightarrow \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ C' = \Gamma_2 \rightarrow \Delta_2, u[t\sigma] \approx v\end{array}}$$

where the following conditions are satisfied

1. $s\sigma \succ t\sigma$
2. $C \succ D\sigma$
3. $N_C \models \Gamma_2 \rightarrow A$ *for all* A *in* $\Gamma_1\sigma$
4. $N_C \models A \rightarrow \Delta_2$ *for all* A *in* $\Delta_1\sigma$

are called contextual rewriting.

Due to condition 1-1 and condition 1-2 we have $C' \prec C$ and $D\sigma \prec C$. Then from condition 1-3 and condition 1-4 we obtain that there exist clauses $C_1, \ldots, C_n \in N_C$ and $C_1, \ldots, C_n, C', D\sigma \models C$. Therefore, the clause C is redundant in $N \cup \{C'\}$ and can be eliminated. The rule is an instance of the abstract superposition redundancy notion.

The side conditions 1-3 and 1-4 having both the form $N_C \models \Gamma \rightarrow \Delta$ are undecidable, in general. There are two obstacles deciding the side conditions $N_C \models \Gamma \rightarrow \Delta$. First, there are infinitely possible grounding substitutions σ' for the clauses $\Gamma \rightarrow \Delta$ and C. Second, even for a given σ' there may be infinitely many ground substitutions δ with $C_i\delta \prec C\sigma'$, $C_i \in N$, e.g., if $\prec$ is the lexicographic path ordering (LPO). Therefore, in order to effectively decide the side conditions, in the following we will fix one σ' and restrict the number of considered substitutions δ to a finite number yielding a decidable instance of contextual rewriting.

First, $N_C \models \Gamma \rightarrow \Delta$ is equivalent to $N_C \cup \{\exists x_1, \ldots, x_n.\neg(\Gamma \rightarrow \Delta)\} \models \bot$ where the x_i are the variables of $\Gamma \rightarrow \Delta$. The existential quantifier can be eliminated by Skolemization yielding a Skolem substitution τ that maps any x_i to a new Skolem constant. Consequently, setting σ' to τ yields the instance $N_C \models (\Gamma \rightarrow \Delta)\tau$, where $(\Gamma \rightarrow \Delta)\tau$ is ground. Still there may exist infinitely many δ with $C_i\delta \prec C\tau$, $C_i \in N$. Furthermore, $C\tau$ may still contain variables as the literal $u[s\sigma] \approx v$ of C may contain variables that do not occur in Γ_2, Δ_2.

Therefore, we restrict δ to those grounding substitutions that map variables to terms only occurring in $C\tau$ or $D\sigma\tau$ where we additionally assume that τ is also grounding for C and $D\sigma$, i.e., it maps any variable occurring in C or $D\sigma$ to an arbitrary fresh Skolem constant. Let $N_{C\tau}^{D\sigma\tau}$ be the set of all ground instances of clauses from N smaller than $C\tau$ obtained by instantiation with ground terms from $D\sigma\tau, C\tau$. Then $N_{C\tau}^{D\sigma\tau}$ is finite and $N_{C\tau}^{D\sigma\tau} \subseteq N_{C\tau}$. Consequently, $N_{C\tau}^{D\sigma\tau} \models (\Gamma \rightarrow \Delta)\tau$ is a sufficient ground approximation of $N_C \models \Gamma \rightarrow \Delta$. Even though this is a decidable approximation of the original problem the set $N_{C\tau}^{D\sigma\tau}$ is exponentially larger than N, in general. In particular, the set typically already gets so large that an instantiation based theorem proving approach does not work out deciding $N_{C\tau}^{D\sigma\tau} \models (\Gamma \rightarrow \Delta)\tau$. For example, the rewriting step from the example in the introduction contains already more than 20 different ground terms out of the clause

$$i(c_1) \approx b, i(c_3) \approx b, i(c_2) \approx b, i(c_4) \approx b, q(c_5) \approx b, q(f(c_4, c_5)) \approx b, b \approx a \rightarrow f(c_4, c_5) \approx nil,\ q(f(c_1, f(c_2, f(c_4, r(c_3, c_5))))) \approx b$$

where the c_i are the freshly introduced Skolem constants. Recall that N is not the input clause set but the set of all clauses generated in the course of a saturation and can thus consists of several (hundred) thousand clauses. The side condition $N_{C\tau}^{D\sigma\tau}$ is typically tested several 10 thousand times for a problem with potential contextual rewriting applications, even with respect to the refinements that we will introduce in the sequel. Therefore, we represent $N_{C\tau}^{D\sigma\tau}$ implicitly by approximating $N_{C\tau}^{D\sigma\tau} \models (\Gamma \rightarrow \Delta)\tau$ by the application of a recursively defined redundancy redundancy called *ground subterm redundant*. A clause is ground subterm redundant, if it can be reduced to true by the reduction rules *tautology reduction*, *forward subsumption*, *obvious reduction* and a particular instance of contextual rewriting called *recursive contextual ground rewriting* defined below. Tautology reduction reduces syntactic and semantic tautologies to true whereas forward subsumption reduces subsumed clauses to true. Obvious reduction eliminates trivial literals [29]. Ground subterm redundancy is shown in Algorithm 1 and explained in detail in the next Chapter 3.

Ground subterm redundancy only applies to ground clauses. Therefore, the following definition introduces an instance of contextual rewriting only working on ground clauses. Further, it adapts contextual rewriting such that it implicitly only considers clauses from $N_{C\tau}^{D\sigma\tau}$. This is in particular guaranteed by condition 2-3 below that limits the clauses used for reductions to so called universally reductive clauses.

Definition 2 (Recursive Contextual Ground Rewriting) *If N is a clause set, $D \in N$, C' ground, σ a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C' = \Gamma_2, u[s\sigma] \approx v \rightarrow \Delta_2}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ \Gamma_2, u[t\sigma] \approx v \rightarrow \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C' = \Gamma_2 \rightarrow \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ \Gamma_2 \rightarrow \Delta_2, u[t\sigma] \approx v\end{array}}$$

where the following conditions are satisfied

1. $s\sigma$ *is a strictly maximal term in* $D\sigma$
2. $u[s\sigma] \approx v \succ s\sigma \approx t\sigma$
3. $vars(s) = vars(D)$
4. $(\Gamma_2 \rightarrow A)$ *is ground subterm redundant for all* A *in* $\Gamma_1\sigma$
5. $(A \rightarrow \Delta_2)$ *is ground subterm redundant for all* A *in* $\Delta_1\sigma$

are called recursive contextual ground rewriting.

Condition 2-1 and condition 2-2 ensure the ordering restrictions required by contextual rewriting. Condition 2-3 implies that $D\sigma$ is ground. A clause D meeting condition 2-1 and condition 2-3 is called *strongly universally reductive.* Condition 2-4 and condition 2-5 recursively apply the ground subterm redundancy criterion.

The ground subterm redundancy criterion is terminating since C' is reduced to a smaller ground clause. As a consequence, also the ground subterm redundancy procedure (Algorithm 1) is terminating. Eventually, our top level instance of contextual rewriting, the subterm contextual rewriting rule, becomes the below rule.

Definition 3 (Subterm Contextual Rewriting) *Let* N *be a clause set,* $C, D \in N$*,* σ *be a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C = \Gamma_2, u[s\sigma] \approx v \rightarrow \Delta_2}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ \Gamma_2, u[t\sigma] \approx v \rightarrow \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C = \Gamma_2 \rightarrow \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ \Gamma_2 \rightarrow \Delta_2, u[t\sigma] \approx v\end{array}}$$

where the following conditions are satisfied

1. $s\sigma \succ t\sigma$

2. $C \succ D\sigma$

3. τ *maps all variables from* $C, D\sigma$ *to fresh Skolem constants*

4. $(\Gamma_2 \rightarrow A)\tau$ *is ground subterm redundant for all* A *in* $\Gamma_1\sigma$

5. $(A \rightarrow \Delta_2)\tau$ *is ground subterm redundant for all* A *in* $\Delta_1\sigma$

are called subterm contextual rewriting.

Note that unit rewriting and non-unit rewriting [29] are also instances of the subterm contextual rewriting rule. Note further that the conditions for the subterm contextual rewriting rule are weaker compared to the recursive contextual ground rewriting rule: the right premise needs not to be ground and the equation $s \approx t$ needs not to be maximal in the first premise. Subterm contextual rewriting uses recursive contextual ground rewriting to effectively decide the side conditions.

In addition to the rewriting style, where subterms are replaced by simpler ones, the general idea of contextual rewriting can also be used to actually eliminate literals, resulting in a generalization of matching replacement resolution [29]. This variant then also uses negative literals for reductions.

Definition 4 (Subterm Contextual Literal Elimination) *Let* N *be a clause set,* $C, D \in N$*,* σ *be a substitution then the reductions*

$$\mathcal{R}\frac{D = \Gamma_1 \rightarrow \Delta_1, s \approx t \qquad C = \Gamma_2, u \approx v \rightarrow \Delta_2}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t\\ \Gamma_2 \rightarrow \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1, s \approx t \rightarrow \Delta_1 \qquad C = \Gamma_2 \rightarrow \Delta_2, u \approx v}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t\\ \Gamma_2 \rightarrow \Delta_2\end{array}}$$

where the following conditions are satisfied

1. $s\sigma = u$ *and* $t\sigma = v$

2. $C \succ D\sigma$

3. τ *maps all variables from* $C, D\sigma$ *to fresh Skolem constants*

4. $(\Gamma_2 \to A)\tau$ *is ground subterm redundant for all* A *in* $\Gamma_1\sigma$

5. $(A \to \Delta_2)\tau$ *is ground subterm redundant for all* A *in* $\Delta_1\sigma$

are called subterm contextual rewriting.

3 Implementation

The implementation of SPASS [29] focuses on a sophisticated reduction machinery. The SPASS main loop operates on two clause sets: WorkedOff and Usable. The WorkedOff set contains the clauses which have been processed and the Usable set contains the clauses which have to be considered for further inferences. When SPASS is started WorkedOff is empty and Usable contains all input clauses. Then in each iteration of the main loop SPASS chooses one clause C from Usable and moves it to WorkedOff. Then it computes all inferences of C with clauses of WorkedOff. Each inferenced clause C' is then fully interreduced using clauses of WorkedOff $\cup$ Usable. This process is called *forward reduction*. After that it reduces all clauses of WorkedOff and Usable by using C' which is called *backward reduction*. After having performed forward reduction and backward reduction, the clause sets WorkedOff and Usable are fully interreduced with respect to C. This is the well-known *Otter* loop that considers all clauses for reduction in contrast to a more lazy approach only considering WorkedOff clauses, called *Discount* loop.

The integration of contextual rewriting into the SPASS main loop consists of two steps. First, the search for appropriate contextual rewrite application candidates. This is analogous to the case of unit rewriting and non-unit rewriting. Finding appropriate rewrite candidates is realized in SPASS via substitution trees [29, 15]. The following shows the non-unit rewriting rule.

NON-UNIT REWRITING

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \qquad C = \Gamma_2, u[s\sigma] \approx v \to \Delta_2}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ C'' = \Gamma_2, u[t\sigma] \approx v \to \Delta_2\end{array}}$$

$$\mathcal{R}\frac{D = \Gamma_1 \to \Delta_1, s \approx t \qquad C = \Gamma_2 \to \Delta_2, u[s\sigma] \approx v}{\begin{array}{c}\Gamma_1 \to \Delta_1, s \approx t \\ C'' = \Gamma_2 \to \Delta_2, u[t\sigma] \approx v\end{array}}$$

where (i) $s \succ t$ and (ii) $\Gamma_1\sigma \subset \Gamma_2, \Delta_1\sigma \subset \Delta_2$

In order to rewrite a clause C the implementation of unit and non-unit rewriting tries to reduce each subterm $s\sigma$ of C. Therefore, for each subterm $s\sigma$ the procedure queries the substitution tree whether there exist a candidate term s. If there exists such a term then the substitution tree returns s together with the matcher σ. After retrieving the clause D of which term s is a subterm, the requirements (i) and (ii) are verified. If they are fulfilled then C is rewritten else the implementation queries the substitution tree for the next candidate term. The retrieval is realized in an iterative way because the first hit is actually already used for reduction.

The second step for integrating contextual rewriting into SPASS is to check the side conditions that require an effective implementation of the ground subterm redundancy check. First of all, it is too costly to explicitly compute the Skolem substitution τ for each clause $(\Gamma \rightarrow \Delta)\tau$ subject to the ground subterm redundancy criterion. Applying τ explicitly requires to allocate memory for the new constants, the resulting terms and the new clause and it requires additional computations to build the clause. Because of the recursive structure of the redundancy criterion this is not feasible. Therefore, our solution is to simply treat variables as constants in the implementation of the redundancy criterion.

In SPASS constants are function objects of arity zero. If the implementation of subterm contextual rewriting replaced the variables of the clause $\Gamma \rightarrow \Delta$ explicitly by fresh constants, then it would create for each variable a function object and insert it into the precedence with lowest precedence. Therefore, the term symbols of the new constants are ordered to each other as well as to the other term symbols. On the other hand variables are represented as integers in SPASS which implicitly orders them. Whenever we consider variables to be constants we assume them to have a lower precedence than any other non-variable symbol of the signature. As a consequence, if we adapt the ordering modules (KBO, RPOS) such that they treat variables in the above way, then our approach has the same properties with respect to ordering computation as creating constants explicitly.

If variables are interpreted as constants the standard procedure of SPASS for finding appropriate rewrite candidates remains unchanged. Let t be a term, u a constant, x a variable and I the term index containing all terms occurring in the clause set N. A generalization for the term $t[u]$ is a tuple (t', σ'), such that $t[t'\sigma'] = t[u]$ where t' is a term and σ' is a substitution. The lookup function for the retrieval of generalizations in the index I will return the same terms for $t[x]$ as for $t[u]$. In more detail, this means that for all generalizations (t_1, σ_1) of $t[u]$ in I there is a generalization (t_2, σ_2) of

```
GroundSubtermRedundant(CLAUSE C, CLAUSE SET N);
Rewritten=True;
while Rewritten do
    Rewritten=False;
    if IsEmpty(C) then return False;
    if IsTautology(C) then return True ;
    if ForwardSubsumption(C, N) then return True;
    if ObviousReduction(C) then Rewritten=True;
    if RecursiveContextualGroundRewriting(C, N) then
        Rewritten=True ;
end
return False
```

Algorithm 1: GroundSubtermRedundant

$t[x]$ in I with $t_1 = t_2$ and σ_1 is equal to σ_2 where all occurrences of u in the co-domain of σ_1 are replaced by x. Consequently, the lookup for appropriate rewrite candidates is independent of the interpretation of the variables.

In the following we present the implementation of the ground subterm redundancy check and verify that it works exactly like an implementation that creates Skolem constants explicitly.

The implementation is depicted in Algorithm 1 and uses tautology check, forward subsumption and obvious reductions from the reduction procedure of Spass. These are the procedures implemented in Spass except that they work with respect to the modified ordering procedures that interpret variables as constants. As explained above the retrieval of candidate terms of forward subsumption remains unchanged.

Algorithm 1 expects as input a clause C and a clause set N and reduces C with respect to N in the main loop. The reductions performed on clause C in Algorithm 2 change C destructively. IsEmpty(C) checks whether the given clause is the empty clause. IsTautology(C) checks if $\models C$. This is realized via a congruence closure algorithm testing whether a positive literal is implied by the negative literals.
ForwardSubsumption(C, N) checks whether C is already subsumed by clauses from N.

$$\mathcal{R}\frac{\Gamma_1 \to \Delta_1 \qquad \Gamma_2 \to \Delta_2}{\Gamma_1 \to \Delta_1}$$

where $\Gamma_1 \subset \Gamma_2$ and $\Delta_1 \subset \Delta_2$.

```
RecursiveContextualGroundRewriting(CLAUSE $C[u[u'] \approx v]$,
                                    CLAUSE SET $N$);
$G = general_{SDT}(N, u')$;
foreach $(s, \sigma) \in G$ do
    $Lits$ = LiteralsContainingTerm($s$);
    foreach $(s \approx t) \in Lits$ s.t. $s\sigma \succ t\sigma$ do
        $D$ = LiteralOwningClause($s \approx t$);
        if $(vars(s) \supset vars(D) \wedge$
            $u[s\sigma] \approx v \succ s\sigma \approx t\sigma \wedge$
            $s\sigma$ strictly maximal term in $D\sigma \wedge$
            $\forall A \in Ante(D\sigma)$ GroundSubtermRedundant($\Gamma \rightarrow A) \wedge$
            $\forall A \in Succ(D\sigma)$ GroundSubtermRedundant($A \rightarrow \Delta$))
        then
            return $C[u[t\sigma] \approx v]$;
        end
    end
end
```
Algorithm 2: RecursiveContextualGroundRewriting

ObviousReduction(C) implements the following rules

$$\mathcal{R}\frac{\Gamma \rightarrow \Delta, s \approx t, s \approx t}{\Gamma \rightarrow \Delta, s \approx t}$$

and

$$\mathcal{R}\frac{\Gamma, s \approx t, s \approx t \rightarrow \Delta}{\Gamma, s \approx t \rightarrow \Delta}$$

and

$$\mathcal{R}\frac{\Gamma \rightarrow \Delta, t \approx t}{\Gamma \rightarrow \Delta}$$

and

$$\mathcal{R}\frac{\Gamma, t \approx t \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

and

$$\mathcal{R}\frac{\Gamma, x \approx t \rightarrow \Delta}{\Gamma \rightarrow \Delta} \qquad \text{if} \quad x \notin (\Gamma \cup \Delta)$$

Further details can be found in the Spass Handbook [31].

RecursiveContextualGroundRewriting(C, N), depicted in Algorithm 2, implements recursive contextual ground rewriting. The variables occurring in C are interpreted as constants in the sense explained above. The call to $general_{SDT}(N, u')$ (line 2) returns the set of generalizations G from N of u' and the respective matcher σ. Then the procedure computes for each of the generalizations the literals and the clauses where they occur. The candidate clauses are then checked for the non-recursive side conditions of contextual rewriting. Because of the conditions $vars(s) = vars(D)$ (line 7) and $s\sigma$ is strictly maximal term in $D\sigma$ (line 8) the rewrite clause D is strongly universally reductive. This means that σ has all variables of D in its domain. The substitution σ replaces all variables of D by terms occurring in C. Therefore, $D\sigma$ contains only variables occurring in C which we assume to be constants. As a consequence, this procedure neither introduces any new Skolem constants nor does it change the precedence of them. Therefore, the ordering check (line 8) is implemented using the above-explained, modified ordering modules treating variables as constants. Additionally, building the subproblems (line 10 – line 11) does also not change the Skolem constants nor introduce new ones. For each of these subproblems RecursiveContextualGroundRewriting recursively calls GroundSubtermRedundant.

Interpreting variables as Skolem constants during the recursive application of GroundSubtermRedundant results exactly in the same behavior where explicitly new constant objects are introduced, but saves time and memory.

The implementation of subterm contextual rewriting and subterm contextual literal elimination is analogous to the implementation of recursive contextual ground rewriting. The difference is that the input clause C is not interpreted to be ground and the local side conditions (line 7 - line 9) are changed with respect to the definition of subterm contextual rewriting and subterm contextual literal elimination, respectively.

Algorithm 3 depicts the forward reduction procedure of Spass where subterm contextual rewriting is integrated. Note that the input clause C is destructively changed during the reductions.

3.1 Integration of Unit and Non-Unit Rewriting

Our first major improvement over [32] is the integration of unit and non-unit rewriting into subterm contextual rewriting. Considering the old Algorithm 3 standard rewriting (line 8), namely unit and non-unit rewriting,

```
1  ForwardReduction(CLAUSE $C$, CLAUSE SET $N$);
2  Rewritten=*True*;
3  while *Rewritten* do
4      Rewritten=*False*;
5      if IsTautology(C) then return *True*;
6      if ObviousReduction(C) then Rewritten=*True*;
7      if ForwardSubsumption($C, N$) then return *True*;
8      if Rewriting($C, N$) then Rewritten = *True*;
9      if SubtermContextualRewrting($C, N$) then
       Rewritten=*True*;
10 end
11 return ($C$)
```

Algorithm 3: ForwardReduction

was implemented independently from contextual rewriting (line 9) in the first implementation [32]. As a result our previous implementation searches the index structure for finding appropriate standard rewriting candidates and then searches the index again for finding candidates for contextual rewriting. In order to save queries to the index and side condition checks we nested standard rewriting into contextual rewriting resulting in the procedure depicted in Algorithm 4.

The procedure IsUnit (line 7) checks if the clause given as argument is a unit. If both C and $D\sigma$ are unit clauses then C can be rewritten. SubsumesBasic checkes if the literals of C except literal $u[u'] \approx v$ subsume all literals of $D\sigma$ except literal $s\sigma \approx t\sigma$. Analogously, we extended recursive contextual ground rewriting by unit and non-unit rewriting.

The integration of unit and non-unit rewriting into contextual rewriting potentially changes the proof search strategy because clauses are reduced in different order. Concerning Algorithm 3 rewriting (line 8) is performed on an input clause C before subterm contextual rewriting. The procedure implementing rewriting reduces the clause C using all clauses of N. When no further reduction with rewriting is possible then subterm contextual rewriting reduces C using N. After integrating standard rewriting into subterm contextual rewriting this is done in parallel. The clause set N is processed only once. Each time a candidate clause is retrieved the procedure checks if standard rewriting is possible. If it is not possible then it immediately checks whether contextual rewriting is possible. This potentially changes the proof search strategy, because the clauses are reduced in a different order.

```
SubtermContextualRewriting(CLAUSE $C[u[u'] \approx v]$,
                           CLAUSE SET $N$);
$G = general_{SDT}(N, u')$;
foreach $(s, \sigma) \in G$ do
    $Lits$ = LiteralsContainingTerm($s$);
    foreach $(s \approx t) \in Lits$ s.t. $s\sigma \succ t\sigma$ do
        $D$ = LiteralOwningClause($s \approx t$);
        if (IsUnit($C$) $\wedge$ IsUnit($D\sigma$)) then return $C[u[t\sigma] \approx v]$;
        else if SubsumesBasic($C, D\sigma$) then return $C[u[t\sigma] \approx v]$;
        else if ($u[s\sigma] \approx v \succ s\sigma \approx t\sigma$ $\wedge$
                $s\sigma$ strictly maximal term in $D\sigma$ $\wedge$
                $\forall A \in Ante(D\sigma)$
                        GroundSubtermRedundant($\Gamma \rightarrow A$) $\wedge$
                $\forall A \in Succ(D\sigma)$
                        GroundSubtermRedundant($A \rightarrow \Delta$))
        then
            return $C[u[t\sigma] \approx v]$;
        end
    end
end
```
Algorithm 4: SubtermContextualRewriting

3.2 Fault Caching

Testing a term whether it can be rewritten using contextual rewriting might cause to perform many procedure calls because of the mutual recursive structure of the side conditions. Memorizing terms that have been identified not to be reducible saves a lot of computation.

The cache itself is realized via a term indexing structure because this provides all the functionality for storing and querying terms that we need. Furthermore, Spass already provides this data structure via substitution trees as explained above. Substitution trees return for a query term a generalization together with the respective substitution. Let t be a term we want to check whether it is already in the fault cache. If the substitution tree returns a term t' and a substitution σ then $t'\sigma = t$. If σ only substitutes variables of t by variables then t is not subterm ground redundant if we consider the same context.

We modified the implementation of recursive contextual ground rewriting such that each time a term is considered for rewriting the algorithm first queries the cache. If the term is in the cache then this term is not considered for rewriting. If it is not in the cache and neither standard rewriting nor subterm contextual rewriting was possible then the algorithm inserts it into the cache. Once a term was included in the cache it remains there. We use a global cache for the implementation in order to avoid as much computation as possible.

This is an approximation because a term that is not reducible in the context of one clause might be reducible in the context of another clause. Further, for checking whether a term is in the fault cache, the implementation considers terms that are generalizations with respect to variable mappings. Because variables are interpreted as constants the cache might reject terms that are reducible with another ordering of the variables. Remember that variables are interpreted as Skolem constants. The fault cache is also compatible with splitting. If a term is inserted into the cache in a split branch that is not valid anymore, it does not produce wrong results because we have a purely negative cache which only excludes terms that could be possibly rewritten. As a consequence, we lose possible applications of a contextual rewriting step. If we also stored terms that can be reduced, this approach would not work because if a term could be identified to be reducible in a split branch then this term does not have to be reducible in another branch. In this case we need to cache clauses and have to consider backtracking updates by the splitting. Similarly, if a term is reducible in the context of one clause with contextual rewriting, this does not have to be the case in the context of another clause. However, the below results show that this heuristic performs well on practical instances.

4 Results

4.1 Results on the TPTP

In this chapter we evaluate the implementations of contextual rewriting in Spass, presented in the last chapter, by comparing it to the current standard configuration of Spass.

As test samples we used the *TPTP 3.2.0* [27] which is a library consisting of 8984 problems for testing automated theorem proving systems. As reference we run Spass version 3.1 that is version 3.0 extended by some bug fixes with default configuration. For the sample runs we integrated contextual rewriting and the respective improvements in this version of Spass. All sample runs were performed with Spass options set to *-RFRew=4 -RBRew=3 -RTaut=2*. This means that both subterm contextual rewriting and subterm contextual literal elimination are activated for forward rewriting, subterm contextual rewriting is activated for backward reductions and semantic tautology checks are activated. The hardware setup consisted of Opteron nodes running at speed of 2.4 GHz equipped with 4 GB RAM for each node. For the sample run as well as for the reference run we set a time limit of 300 seconds for each problem.

The problems of the TPTP are ranked from 0.00 to 1.00 indicating their difficulty. Basically, the value expresses how many of the current existing provers were able to solve a particular problem. This means that problems with rating 1.00 have not been solved by any prover so far. For further details we refer to [28]. We compare Spass containing our new improvements to the original Spass with respect to the different rankings of problems. All proofs that Spass with contextual rewriting could find additionally were checked by the Spass proof checker.

The results of running Spass containing our first implementation of contextual rewriting are depicted in Table 4.1. This version found 85 additional proofs and lost 143 proofs compared to the run without contextual rewriting if we consider all problems. If we consider problems with rank greater then

0.65 then Table 4.1 shows that contextual rewriting solves more problems than it loses. The higher the rank the better are the results of contextual rewriting compared to the reference version. If we consider problems with threshold greater than 0.9, SPASS with contextual rewriting found 21 additional proofs and lost none. It even could solve five additional problems.

threshold	lost	won
0.00	143	85
0.50	80	69
0.65	48	61
0.80	3	42
0.90	0	21
1.00	0	5

Table 4.1: Contextual rewriting, 300 seconds time limit

First we did not understand why contextual rewriting lost so many easy problems. Then by inspecting some of the lost problems we could identify two reasons. First, the actual proof found by the standard version of SPASS got lost through a contextual rewriting application. Second, a call to a contextual rewriting procedure took so long that SPASS did not finish within the time limit although the reference run terminated within milliseconds. Therefore, we improved our implementation of contextual rewriting according to the lines of Chapter 3.

4.1.1 Integrated Unit and Non-Unit Rewriting

The integration of unit and non-unit rewriting into the implementation of contextual rewriting improved the results significantly. Although more problems were lost than before more could be additionally solved. This improvement narrowed the gap between solved and lost problems. Table 4.2 depicts the results.

Considering all problems the version with integrated unit and non-unit rewriting found 152 additional proofs and lost 119. It still solves hard problems whereas it is able to solve more easy once.

4.1.2 Fault Caching

After additionally integrating the fault cache in the contextual rewriting procedure the results further improved as Table 4.3 depicts. SPASS found 132

threshold	lost	won
0.0	152	119
0.5	88	71
0.65	51	62
0.8	4	36
0.9	0	20
1.0	0	5

Table 4.2: Integrated Unit and Non-Unit Rewriting, 300 seconds time limit

additional problems whereas it only lost 67. This implementation improved much on the easy problems but could solve new difficult problems, too. Even one additional problem that has not been solved before.

threshold	lost	won
0.0	67	132
0.5	43	81
0.65	23	71
0.8	3	40
0.9	0	22
1.0	0	6

Table 4.3: Contextual Rewriting with Caching, 300 seconds time limit

The following table compares SPASS with contextual rewriting and all improvements with a time limit of 900 seconds to the reference run. We see that almost half of the 67 lost problems could be regained by increasing the time limit.

The six new problems that we could solve are all from the software model checking category of the TPTP. The problems are: SWC308+1, SWC329+1, SWC345+1, SWC342+1, SWC261+1, SWC335+1. This is not a surprise as contextual rewriting can for example employ conditional access function definitions for reduction. For example, a list element access function that first checks for emptiness is perfectly matched by our subterm contextual rewriting rule.

It was a surprise to us that the implementation of contextual rewriting did not improve on satisfiable problems. In the first implementation we even lost seven problems and did not terminate on problems where the reference version did. After the integration of unit and non-unit rewriting we even

threshold	lost	won
0.0	38	190
0.5	23	99
0.65	7	83
0.8	3	40
0.9	0	22
1.0	0	6

Table 4.4: Contextual Rewriting with Caching, 900 sec run time

lost eight problems. But also for satisfiability problems it turned out that the fault cache is useful because the version containing the fault cache only lost one problem. Running the version with caching and a time limit of 900 seconds still lost this particular problem but additionally terminated on three problems. We are wondering about that phenomenon because contextual rewriting is capable to increase the number of problems where SPASS can terminate on. An example for such a problem is the introductory example which is shown in more detail in the next section. The TPTP version 3.2.0 does not contain such problems.

4.2 Application to the Example from the Introduction

In this part we depict the application of approximated recursive contextual rewriting on the introductory example in detail. Therewith, we show that superposition together with our instance of contextual rewriting terminates on this example. SPASS with contextual rewriting is able to saturate the clause set from section 1 whereas SPASS without contextual rewriting is not. Recall that we can reduce clause 5 with clause 3 using contextual rewriting if the side conditions are fulfilled. The ground clauses

8 : $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, b \approx a \rightarrow i(x_1) \approx b$

9 : $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, b \approx a \rightarrow q(f(x_2, r(x_3, y_1))) \approx b$

must be entailed by clauses from N_C. Clause 8 is a tautology whereas clause 9 can be rewritten with clause 3 to

10: $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, a \approx b \rightarrow a \approx b$

using contextual rewriting if in addition the clauses

11: $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, a \approx b \rightarrow i(x_2) \approx b$

12: $i(x_1) \approx b, i(x_3) \approx b, i(x_2) \approx b, q(y_1) \approx b, a \approx b \rightarrow q(r(x_3, y_1)))) \approx b$

are also entailed by clauses from N_C. Clause 11 is a syntactic tautology and clause 12 is subsumed by clause 2.

5 Summary

In summary, contextual rewriting is costly but it helps solving difficult problems. Our current implementation improves significantly over our first suggestion [32]. It actually gains more problems than it loses and is able to solve many difficult problems. The SPASS version described in this paper and used for the experiments can be obtained from the SPASS homepage (`http://spass-prover.org/`) following the prototype link.

Bibliography

[1] L. Bachmair and H. Ganzinger. On restrictions of ordered paramodulation with simplification. In *10th International Conference on Automated Deduction, CADE-10*, volume 449 of *LNCS*, pages 427–441. Springer, 1990.

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

[3] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.

[4] L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In *Computational Logic and Proof Theory*, volume 713 of *LNCS*, pages 83–96, 1993.

[5] P. Balbiani. Equation solving in geometrical theories. In Dershowitz and Lindenstrauss [9], pages 31–50.

[6] P. Bernays and M. Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *Mathematische Annalen*, 99:342–372, 1928.

[7] C. Brinker. Geometrisches schliessen mit spass. Diplomarbeit, Universität des Saarlandes and Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2000. Supervisors: H. Ganzinger, C. Weidenbach.

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

[9] N. Dershowitz and N. Lindenstrauss, editors. *Conditional and Typed Rewriting Systems, 4th International Workshop, CTRS-94, Jerusalem, Israel, July 13-15, 1994, Proceedings*, volume 968 of *Lecture Notes in Computer Science*. Springer, 1995.

[10] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

[11] C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tamet. Resolution decision procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 25, pages 1791–1849. Elsevier, 2001.

[12] H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *LICS*, pages 295–304, 1999.

[13] H. Ganzinger and R. Nieuwenhuis. The saturate system 1994. http://www.mpi-sb.mpg.de/SATURATE/Saturate.html, 1994.

[14] H. Ganzinger, R. Nieuwenhuis, and P. Nivela. Fast term indexing with coded context trees. *J. Autom. Reasoning*, 32(2):103–120, 2004.

[15] P. Graf. *Term Indexing*, volume 1053 of *LNAI*. Springer-Verlag, 1995.

[16] U. Hustadt, R. A. Schmidt, and L. Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004.

[17] F. Jacquemard, C. Meyer, and C. Weidenbach. Unification in extensions of shallow equational theories. In *Proceedings of RTA-98*, volume 1379 of *LNCS*, pages 76–90. Springer, 1998.

[18] B. Konev, R. A. Schmidt, and S. Schulz, editors. *Proceedings of the First International Workshop on Practical Aspects of Automated Reasoning, Sydney, Australia, August 10-11, 2008*, volume 373 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[19] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier, 2001.

[20] P. Nivela and R. Nieuwenhuis. Saturation of first-order (constrained) clauses with the *Saturate* system. In C. Kirchner, editor, *Rewriting Techniques and Applications, 5th International Conference, RTA-93*, volume

690 of *Lecture Notes in Computer Science, LNCS*, pages 436–440, Montreal, Canada, June 16–18, 1993. Springer.

[21] G. E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computation*, 12(1):82–100, February 1983.

[22] I. V. Ramakrishnan, R. C. Sekar, and A. Voronkov. Term indexing. In Robinson and Voronkov [26], pages 1853–1964.

[23] A. Riazanov and A. Voronkov. Efficient instance retrieval with standard and relational path indexing. *Inf. Comput.*, 199(1-2):228–252, 2005.

[24] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 135–150, 1969.

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

[26] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[27] G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[28] G. Sutcliffe and C. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[29] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.

[30] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS version 2.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 275–279, Kopenhagen, Denmark, 2002. Springer.

[31] C. Weidenbach, R. Schmidt, and E. Keen. Spass handbook version 3.0. Contained in the documentation of SPASS Version 3.0, 2007.

[32] C. Weidenbach and P. Wischnewski. Contextual rewriting in spass. In Konev et al. [18].

.

MPI-I-2008-5-003	F.M. Suchanek, G. de Melo, A. Pease	Integrating Yago into the Suggested Upper Merged Ontology
MPI-I-2008-5-002	T. Neumann, G. Moerkotte	Single Phase Construction of Optimal DAG-structured QEPs
MPI-I-2008-5-001	F. Suchanek, G. Kasneci, M. Ramanath, M. Sozio, G. Weikum	STAR: Steiner Tree Approximation in Relationship-Graphs
MPI-I-2008-1-001	D. Ajwani	Characterizing the performance of Flash memory storage devices and its impact on algorithm design
MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for Finite Domains
MPI-I-2007-5-003	F.M. Suchanek, G. Kasneci, G. Weikum	Yago : A Large Ontology from Wikipedia and WordNet
MPI-I-2007-5-002	K. Berberich, S. Bedathur, T. Neumann, G. Weikum	A Time Machine for Text Search
MPI-I-2007-5-001	G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum	NAGA: Searching and Ranking Knowledge
MPI-I-2007-4-008	J. Gall, T. Brox, B. Rosenhahn, H. Seidel	Global Stochastic Optimization for Robust and Accurate Human Motion Capture
MPI-I-2007-4-007	R. Herzog, V. Havran, K. Myszkowski, H. Seidel	Global Illumination using Photon Ray Splatting
MPI-I-2007-4-006	C. Dyken, G. Ziegler, C. Theobalt, H. Seidel	GPU Marching Cubes on Shader Model 3.0 and 4.0
MPI-I-2007-4-005	T. Schultz, J. Weickert, H. Seidel	A Higher-Order Structure Tensor
MPI-I-2007-4-004	C. Stoll	A Volumetric Approach to Interactive Shape Editing
MPI-I-2007-4-003	R. Bargmann, V. Blanz, H. Seidel	A Nonlinear Viseme Model for Triphone-Based Speech Synthesis
MPI-I-2007-4-002	T. Langer, H. Seidel	Construction of Smooth Maps with Mean Value Coordinates
MPI-I-2007-4-001	J. Gall, B. Rosenhahn, H. Seidel	Clustered Stochastic Optimization for Object Recognition and Pose Estimation
MPI-I-2007-2-001	A. Podelski, S. Wagner	A Method and a Tool for Automatic Veriication of Region Stability for Hybrid Systems
MPI-I-2007-1-002	E. Althaus, S. Canzar	A Lagrangian relaxation approach for the multiple sequence alignment problem
MPI-I-2007-1-001	E. Berberich, L. Kettner	Linear-Time Reordering in a Sweep-line Algorithm for Algebraic Curves Intersecting in a Common Point
MPI-I-2006-5-006	G. Kasnec, F.M. Suchanek, G. Weikum	Yago - A Core of Semantic Knowledge
MPI-I-2006-5-005	R. Angelova, S. Siersdorfer	A Neighborhood-Based Approach for Clustering of Linked Document Collections
MPI-I-2006-5-004	F. Suchanek, G. Ifrim, G. Weikum	Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents
MPI-I-2006-5-003	V. Scholz, M. Magnor	Garment Texture Editing in Monocular Video Sequences based on Color-Coded Printing Patterns