SPASS Version 3.5

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski

Max-Planck-Institut für Informatik, Campus E1 4
D-66123 Saarbrücken
spass@mpi-inf.mpg.de

Abstract. SPASS is an automated theorem prover for full first-order logic with equality and a number of non-classical logics. This system description provides an overview of our recent developments in SPASS 3.5 including subterm contextual rewriting, improved split backtracking, a significantly faster FLOTTER implementation with additional control flags, completely symmetric implementation of forward and backward redundancy criteria, faster parsing with improved support for big files, faster and extended sort module, and support for include commands in input files. Finally, SPASS 3.5 can now parse files in TPTP syntax, comes with a new converter tptp2dfg and is distributed under a BSD style license.

1 Introduction

SPASS is an automated theorem prover for full first-order logic with equality and a number of non-classical logics. This system description provides an overview of our recent developments in SPASS version 3.5 compared to version 3.0 [WSH+07].

One important change in the use of SPASS when moving from SPASS version 3.0 to SPASS version 3.5 is the change from a GPL to a BSD style license. We have been asked by several companies for this change and now eventually did it. Actually, it took some effort as it required the (re)implementation of several SPASS modules as we were so far relying on some code distributed under a GPL license, for example the command line parser up to version 3.0. Starting from SPASS version 3.5 it will be distributed under a "non-virulent" BSD style license.

The most important enhancements based on advances in theory are subterm contextual rewriting (Section 2) and improved split backtracking (Section 3).

Important enhancements based on further (re)implementations are a significantly faster FLOTTER procedure with more control on reduction during CNF translation (Section 4), faster parsing with support for big files, a faster and extended sort module, support for include commands in SPASS input files, support for the TPTP input problem syntax and a new tool tptp2dfg (Section 5).

In the subsequent sections we will explain our enhancements in more detail and also provide experimental data. All experiments were carried out on the TPTP version 3.2.0 [SS98] and performed on Opteron 2.4GHz computers running Linux with a 300 second time limit for each problem.

R.A. Schmidt (Ed.): CADE 2009, LNAI 5663, pp. 140–145, 2009.

2 Subterm Contextual Rewriting

Contextual rewriting is a powerful reduction rule generalizing standard unit equational rewriting to equational rewriting under "conditions". For 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 [Wei01]. 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 $P(x)\sigma$ is entailed by 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 $P(x) \to f(x) \approx x$. This step can not be performed by standard unit equational rewriting. Clauses of the form $\Gamma \to \Delta, s \approx t$ occur, e.g., often in problems from software verification where the execution of a function call represented by the term s may depend on some conditions represented in Γ, Δ. Then contextual rewriting simulates conditional execution. In case a clause set N is saturated, for all clauses of the above form $s \approx t$ is strictly maximal in the clause, s is strictly larger than t and s contains all variables occurring in Γ, Δ, and t, then contextual rewriting can be used to effectively decide validity of any ground clause with respect to the ordering based minimal model of N [GS92]. Translated to the above software verification scenario it means that in such a setting contextual rewriting effectively computes any function from N.

Contextual rewriting [BG94] was first implemented in the SATURATE system [GN94] 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 prover does not return in reasonable time from even a single contextual rewriting application test. Compared to this work, we have instantiated the contextual rewriting rule to subterm contextual rewriting and have implemented it in a much more sophisticated way. Our new rule is robust in the sense that invoking this rule in SPASS on the overall TPTP [SS98] results in an overall gain of solved problems.

The reduction rule *Subterm Contextual Rewriting* [WW08] is defined as follows. As already said, we write clauses in implication notation $\Gamma \to \Delta$ denoting that the conjunction of all atoms in Γ implies the disjunction of all atoms in Δ. As usual $\prec$ is a reduction ordering total on ground terms. Let N be a clause set, $C, D \in N$, σ be a substitution then the reductions

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

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

where the following conditions are satisfied (i) $s\sigma \succ t\sigma$, (ii) $C \succ D\sigma$, (iii) τ maps all variables from $C, D\sigma$ to fresh Skolem constants, (iv) $(\Gamma_2 \rightarrow A)\tau$ is ground subterm redundant in N for all A in $\Gamma_1\sigma$, (v) $(A \rightarrow \Delta_2)\tau$ is ground subterm redundant in N for all A in $\Delta_1\sigma$, are subterm contextual rewriting reductions.

A ground clause $\Gamma \rightarrow \Delta$ is *ground subterm redundant* in N if there are ground instances $C_1\sigma_1, \ldots, C_n\sigma_n$ of clauses from N such that all codomain terms of the σ_i are subterms of $\Gamma \rightarrow \Delta$, the clauses $(C_i\sigma_i) \prec (\Gamma \rightarrow \Delta)$ for all i, and $C_1\sigma_1, \ldots, C_n\sigma_n \models \Gamma \rightarrow \Delta$.

Subterm contextual rewriting is an instance of the general contextual rewriting rule. In particular, the recursive conditions (iv) and (v) are tested with respect to ground clauses of the form $\Gamma \rightarrow \Delta$ and only smaller ground clauses $C_i\sigma_i$. The smaller ground clauses $C_i\sigma_i$, considered for the test, are build by instantiating clauses from N only with ground terms from $\Gamma \rightarrow \Delta$. We actually implemented these conditions by a recursive call to the reduction machinery of SPASS including subterm contextual rewriting where we simply treat the variables in $(\Gamma_2 \rightarrow A)$, $(A \rightarrow \Delta_2)$ as fresh Skolem constants. This has the important advantage that the side condition clauses don't need to be explicitly instantiated (by τ). On the other hand we needed to extend our ordering computation algorithms to deal with particular variables as constants. Further details can be found in [WW08].

Even under these restrictions, our first implementation of the subterm contextual rewriting rule lost more examples on the overall TPTP than it won. A careful inspection of several runs showed that in particular a lot of time is wasted by testing the same failing terms for contextual rewriting again and again. Therefore, we decided to introduce *fault caching* as a form of (negative) dynamic programming. Whenever subterm contextual rewriting is tested for applicability on some term $s\sigma$ and the test fails, we store $s\sigma$ in an index. Then all subsequent tests first look into the index and only if the top term to be reduced is not contained, the conditions for the rule are evaluated. Of course, this is an approximation of the rule as some potential applications are lost. However, it turned out that at least with respect to the TPTP, SPASS with subterm contextual rewriting with fault caching solves more examples than SPASS without.

Subterm contextual rewriting can be controlled by the forward and backward rewriting flags `-RFRew` and `-RBRew` where starting from a value of `3` the rule is activated. Further details can be found in the SPASS man page.

Comparing SPASS version 3.0 with SPASS version 3.5 with activated subterm contextual rewriting (set flags `-RFRew=4 -RBRew=3 -RTaut=2`) yields an overall gain of 31 problems on the TPTP (actually 108 losses and 139 wins). The losses are partly due to the additional time needed for subterm contextual rewriting, partly due to a differently spanned search space. Eventually SPASS with subterm contextual rewriting solved 6 "open" TPTP problems (rating 1.0): SWC261+1, SWC308+1, SWC329+1, SWC335+1, SWC342+1, and SWC345+1. For this experiment we deactivated the advanced split backtracking introduced in the next section and used the split backtracking of SPASS version 3.0.

3 Improved Split Backtracking

The splitting rule implementation of SPASS until version 3.0 already contains a form of intelligent backtracking via branch condensing. For each clause we store in a bitfield the splits it depends on. If an empty clause is derived then all splits that did not contribute to the empty clause, easily checked from the empty clause's split bitfield, are removed by branch condensing.

We recently refined the splitting calculus [FW08] by storing with each split the bitfield of an empty clause after backtracking one of its branches. If now the second branch is also refuted then this information can be propagated upwards in the split tree by combining the bitfields of sibling branches or subtrees. Any splits that neither contributed to left nor to the right empty clause can be undone. It turns out that this refinement saves a remarkable number of splits on the TPTP problems.

On the average on all TPTP problems where SPASS performs splitting and which are solved both by version 3.0 and version 3.5, version 3.5 does 783 splits whereas version 3.0 performs 916 splits per problem. This means a saving of 14%. Furthermore, due to splitting SPASS 3.5 solves 28 more problems from the TPTP. Actually it loses 21 problems and wins 49 problems. For these experiments subterm contextual rewriting was not activated so that only the effect of the new split backtracking implementation shows up.

4 Improvements to FLOTTER

FLOTTER is the powerful CNF transformation procedure of SPASS. In particular, it contains sophisticated transformation rules such as optimized Skolemization [NW01] which actually require the computation of proofs during CNF transformation. For larger problems these techniques may actually consume so much time that FLOTTER does not terminate in an acceptable amount of time.

Therefore, we both improved the implementation of crucial FLOTTER parts with respect to large problems and added further flags that can be used to restrict sophisticated reductions during CNF transformation. The new flags `-CNFSub` and `-CNFCon` control the usage of subsumption and condensing during CNF transformation, respectively. The new flag `-CNFRedTimeLimit` can be used to set an overall time limit on all reductions performed in FLOTTER during CNF translation.

5 Further Enhancements

Faster Parsing: Until SPASS version 3.0 the overall input machinery was developed for "small" input files. Due to our own and our customers interest in "large" problems, e.g., expressing real-world finite domain theories, we have reimplemented the overall SPASS parsing technology. We now can parse a 60 MB file in less than 10 seconds and build the CNF for a 1 MB input file like SEU410+2 from TPTP version 3.5.0 with *full* FLOTTER CNF translation and reduction in about 30 seconds.

TPTP Input Syntax Support. Starting from SPASS version 3.5 we support TPTP input files via the new flag `-TPTP`. As TPTP input files may contain include commands, they are resolved by looking in the local directory or the value of the `TPTP` environment variable.

Include Commands. SPASS input files may now also contain include directives. They are resolved at parse time and include files are looked up in the local directory as well as in the directory bound to the `SPASSINPUTS` environment variable.

tptp2dfg. The new tool tptp2dfg translates input files from TPTP into SPASS syntax. Includes can either be expanded or translated into SPASS include directives controlled by the flag `-include`.

Sort Module. In SPASS sorts are used for soft typing and sort simplification reductions [Wei96, GMW97, Wei01]. For example, a clause $S(f(a)), \Gamma \rightarrow \Delta$ is reduced to $\Gamma \rightarrow \Delta$ in the presence of the clauses $\rightarrow S(a)$ and $S(x) \rightarrow S(f(x))$. We reimplemented the module such that it is now about 10-times faster and extended its scope. For example, the new module reduces a clause $S(x), T(x), \Gamma \rightarrow \Delta$ where x does not occur in Γ, Δ to $\Gamma \rightarrow \Delta$ in the presence of the three clauses $\rightarrow S(a)$, $S(x) \rightarrow S(f(x))$, and $\rightarrow T(f(a))$.

Symmetric Reduction: Until SPASS version 3.0 some of the more sophisticated rewrite reductions were only implemented in the forward direction [Wei01]. We now added also the backward direction for all reduction rules.

6 Future Work

There are five major directions for future improvements. Firstly, we will continue integrating results on the superposition theory into SPASS. We are currently working on integrating support for transitive relations and refinements thereof [BG94] and will shortly start implementing particular techniques for finite domain problems [HW07]. Secondly, we will continue our work on hierarchic combinations, in particular with the theory of linear arithmetic. Thirdly, we are planning a reimplementation of a reasonable portion of the SPASS basic modules in order to save memory consumption at run time, gaining further speed and simplifying structures that have been grown during the meanwhile 15 years of SPASS development. Fourthly, we are working on the theory and implementation of a specific SPASS version for reasoning in large ontologies. Fifthly, we will integrate support for model generation and minimal model reasoning into SPASS [HW09].

SPASS version 3.5 is available from its homepage

`http://www.spass-prover.org/`

Acknowledgments. We thank Geoff Sutcliffe for his persistent support for implementing parsing of the TPTP input syntax and our reviewers for their valuable comments.

References

[BG94] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. Journal of Logic and Computation 4(3), 217–247 (1994); Revised version of Max-Planck-Institut für Informatik technical report, MPI-I-91-208 (1991)

[FW08] Fietzke, A., Weidenbach, C.: Labelled splitting. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 459–474. Springer, Heidelberg (2008)

[GMW97] Ganzinger, H., Meyer, C., Weidenbach, C.: Soft typing for ordered resolution. In: McCune, W. (ed.) CADE 1997. LNCS (LNAI), vol. 1249, pp. 321–335. Springer, Heidelberg (1997)

[GN94] Ganzinger, H., Nieuwenhuis, R.: The saturate system 1994 (1994), `http://www.mpi-sb.mpg.de/SATURATE/Saturate.html`

[GS92] Ganzinger, H., Stuber, J.: Inductive theorem proving by consistency for first-order clauses. In: Rusinowitch, M., Remy, J.-L. (eds.) CTRS 1992. LNCS, vol. 656, pp. 226–241. Springer, Heidelberg (1992)

[HW07] Hillenbrand, T., Weidenbach, C.: Superposition for finite domains. Research Report MPI-I-2007-RG1-002, Max-Planck Institute for Informatics, Saarbruecken, Germany (April 2007)

[HW09] Horbach, M., Weidenbach, C.: Decidability results for saturation-based model building. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 404–420. Springer, Heidelberg (2009)

[NW01] Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 6, vol. 1, pp. 335–367. Elsevier, Amsterdam (2001)

[SS98] Sutcliffe, G., Suttner, C.B.: The TPTP problem library – cnf release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)

[Wei96] Weidenbach, C.: Unification in sort theories and its applications. Annals of Mathematics and Artificial Intelligence 18(2-4), 261–293 (1996)

[Wei01] Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 27, vol. 2, pp. 1965–2012. Elsevier, Amsterdam (2001)

[WSH+07] Weidenbach, C., Schmidt, R., Hillenbrand, T., Rusev, R., Topic, D.: Spass version 3.0. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 514–520. Springer, Heidelberg (2007)

[WW08] Weidenbach, C., Wischnewski, P.: Contextual rewriting in Spass. In: PAAR/ESHOL, Sydney, Australien. CEUR Workshop Proceedings, vol. 373, pp. 115–124 (2008)