SPASS Version 2.0

Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topić

Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany,
contact: spass@mpi-sb.mpg.de

Abstract. SPASS is an automated theorem prover for full first-order logic with equality. This system description provides an overview of recent developments in SPASS 2.0, including among others an implementation of contextual rewriting, refinements of the clause normal form transformation, and enhancements of the inference engine.

1 Introduction

SPASS is an automated theorem prover for full sorted first-order logic with equality that extends superposition by sorts and a splitting rule for explicit case analysis (Weidenbach, Afshordel, Brahm, Cohrs, Engel, Keen, Theobalt & Topic 1999). The prover runs on almost any recent computer platform (see Section 2.5).

SPASS is meant to be useful for two groups of users: people using it as a theorem-proving tool and people using it as a development platform for theorem-proving tools. Many of our recent efforts went into improving value and ease of use for both groups.

2 New Features

Compared to SPASS version(s) 1.0.x described earlier by Weidenbach et al. (1999), we added the functionality described in the following subsections. Most of the underlying theory is explained by Nonnengart & Weidenbach (2001) and Weidenbach (2001). The notation and notions of these handbook chapters are used throughout this article.

2.1 Reduction Rules

Although contextual rewriting is a rather old concept (Bachmair & Ganzinger 1994), refining and instantiating and actually implementing this reduction rule such that it performs well in relevant application areas is a difficult task.

The reduction rule contextual rewriting[1]

[1] This is a simplified version, hiding our integrated sort concept, and only the left variant of the rule.

$$\mathcal{R}\frac{\Gamma_1 \rightarrow \Delta_1, s \approx t \quad \Gamma_2, E[s']_p \rightarrow \Delta_2}{\begin{array}{c}\Gamma_1 \rightarrow \Delta_1, s \approx t \\ \Gamma_2, E[p/t\sigma] \rightarrow \Delta_2\end{array}}$$

allows a reduction of the term $s' = s\sigma$ provided among other restrictions that the literals in $\Gamma_1 \rightarrow \Delta_1$ can be shown to be logically implied by $\Gamma_2 \rightarrow \Delta_2$ and all other smaller clauses: $N_C \models \Gamma_2 \rightarrow E$ for all atoms $E \in \Gamma_1\sigma$ ($N_C \models E \rightarrow \Delta_2$ for all atoms $E \in \Delta_1\sigma$) where N_C denotes all available clauses smaller than $C = \Gamma_1 \rightarrow \Delta_1, s \approx t$.

This semantic test could easily be approximated recursively, e.g. by considering only syntactically smaller clauses with respect to variable renaming and by replacing the models relation $\models$ by recursive application of contextual rewriting and congruence closure. But this approach, although effective, is not without problems. In relevant cases too much time is spent to test the applicability of the rule or the prover even gets stuck in searching for contextual rewriting applications. Therefore, we decided to start with stronger restrictions that still give many benefits in practice, but can be much more efficiently computed. Our restrictions to the rule are: (i) $s\sigma = s'$, (ii) $s \succ t$, (iii) for any term t' in $\Gamma_1 \rightarrow \Delta_1, s \approx t$ we require $s \succ t'$, (iv) $\models \mathit{fred}(\Gamma_2 \rightarrow E)$ for all atoms $E \in \Gamma_1\sigma$, (v) $\models \mathit{fred}(E \rightarrow \Delta_2)$ for all atoms $E \in \Delta_1\sigma$.

The conditions (ii) and (iii) guarantee the necessary ordering restrictions, while the conditions (iv) and (v) force the context of $s \approx t$ to be valid. The function *fred*, for *forward reduction*, (recursively) applies the reduction rules of SPASS to the constructed subclauses before they are checked as tautologies. This is a weaker but much more efficient variant of the above-mentioned syntactic test. The tautology check itself is highly efficiently integrated into SPASS 2.0 by the implementation of a variant of the well-known congruence closure algorithm (Downey, Sethi & Tarjan 1980, Brinker 2000) on top of the SPASS datastructures. The performance of *fred*, i.e., which reduction rules are actually selected by the user for a SPASS run, determines the strength as well as the cost for testing and applying contextual rewriting.

One nice result of this enhancement is that SPASS is now able to finitely saturate Balbiani's (Balbiani 1995) formulation of a certain fragment of the Euclidean geometry (Brinker 2000). Currently we are investigating examples from formal system analysis, e.g. using SPASS as the reasoning backend in static analysis of list-processing C programs (Topić 2002), to get an appropriate mix of reduction rules for the *fred* function in this application domain.

As an additional reduction mechanism we integrated the so-called *terminator* suggested by Antoniou & Ohlbach (1983) into SPASS 2.0. The idea is to exhaustively search for a unit refutation with a bounded use of non-unit clauses.

2.2 Clause Normal Form Translation

In addition to the theory now described in full detail by Nonnengart & Weidenbach (2001) and presented in our previous system description (Weidenbach et al. 1999), we improved the renaming selection. Recall that we rename a formula if the introduction of a new atom for the formula plus the definition equivalence reduces the number of eventually generated clauses. Once a formula ϕ is detected as such a renaming candidate, we now first search for a more general formula ψ that occurs in the overall formula. We generalize terms in ϕ by free variables that have to be properly universally bound in ψ. If we find such a formula, we use it instead of ϕ as the renaming formula and rename ϕ and potential further hits as instances of ψ.

We revisited our notion of an atom definition extended by a guard as in

$$\forall x_1, \ldots, x_m \, (\phi \supset (P(t_1, \ldots, t_n) \equiv \psi)).$$

The theory is now described in full detail in a paper by Afshordel, Hillenbrand & Weidenbach (2001) and implemented exactly that way in SPASS 2.0. For the proof of the guard formulae we use SPASS as a proper sub proof procedure, now possible in SPASS 2.0, as indicated in Section 2.4.

2.3 Inference Engine

Theorem provers tend to be sensitive to the addition of even single new clauses or just a reordering of the clauses in the input problem. We want the behavior of SPASS to be more robust against such changes. To this end we implemented an initial clause ordering that is not simply based on the weight of the clauses, but their syntactic characteristics. Part of it is an abstraction of a clause as nested multiset of its literal and term structure, where all symbols are eventually abstracted by their arity. These nested multisets are then used, in addition to the weight, to get the initial ordering of the clause set. The technique is a generalization of what is implemented in WALDMEISTER (Hillenbrand, Jaeger & Löchner 1999) for equations.

The automatic selection of the precedence for the term ordering is no longer simply based on function symbol arity in SPASS 2.0. The overall idea here is to orient as many equations (get unique maximal literals in clauses) as possible. To this end a graph is built based on the left- and right-hand side terms of positive equations. Whenever there is an equation $f(t_1, \ldots, t_n) \approx g(s_1, \ldots, s_m)$ a link between f and g is established annotated with the equation. Then it is tried to turn the resulting graph into a directed acyclic graph with an appropriate precedence, where a link can be oriented if all annotated equations can be appropriately oriented with respect to the selected precedence. The strategy for the precedence search is a hill-climbing approach.

SPASS performs explicit splitting of clauses whenever a clause can be separated into two variable disjoint parts that both contain at least one positive literal. The selection of such a clause in SPASS 1.0.x was determined by the selection of the next *given* clause for inference selection (Weidenbach 2001). The main motivation for splitting a clause is to use the separated parts for exhaustive reduction, e.g. subsumption. SPASS 2.0 now selects the clause for splitting by this criterion. The clause with the highest reduction potential for subsumption and unit reductions after splitting is selected for splitting.

With Spass 2.0 we do not only provide an inference/reduction procedure with complete interreduction, but also a variant where the set of *usable* clauses is not interreduced (Weidenbach 2001). Although we do not support memory compression techniques for this case, the time for interreduction of the *usable* clauses is saved in this mode.

2.4 Code Development

One of our long-term goals is to provide as much as possible of the functionality of Spass in form of a documented C library. Towards this end, we improved modularization, debugging support and code documentation. And as is always the case for such a complex piece of software, we fixed several bugs in the code, thanks to reports from our users.

In Spass 2.0 we now introduced a `PROOFSEARCH` datatype that holds all information about a particular proof attempt. This includes the input problem, options, ordering settings, clause data bases, etc. For example, any clause belongs to exactly one `PROOFSEARCH` object (via labelling). The search engine is parameterized by the `PROOFSEARCH` object. This was made possible by further modularizations of the underlying source code resulting in the elimination of almost all global variables. So we can now run several proof attempts completely independently at the same time within Spass. The test of the guard formula, see Section 2.2, or our soft typing approach (Ganzinger, Meyer & Weidenbach 1997) are examples of such a usage.

Spass 2.0 has a complete documentation of its search engine, inference rules and options with pointers into the source code.

2.5 Web Interface and Distribution

The Web interface available from the Spass homepage:

`http://spass.mpi-sb.mpg.de/`

is now also updated to version 2.0 and has links to the new documentation.

The Spass distribution contains binaries for SUN Sparc Solaris, PC Linux and PC Windows 95/Windows 98/Windows NT/Windows 2000 and Windows XP. For the Windows versions we moved our GUI to Qt Version 3.0[2] and this GUI is also available as a SUN Sparc Solaris binary. Under a Unix environment, the source code (excluding the GUI) should compile without any problems if recent versions of the standard GNU[3] tools `bison`, `flex`, `make` and `gcc` are available. The distribution is available from the above-mentioned Spass homepage where also links to the Web interface, documentation and problem libraries exist.

The distribution of Spass contains `texinfo`-based documentation in different formats: `man` pages, `info` pages and HTML pages. Furthermore, full documentation of the search engine, inference rules and underlying theory with pointers into the source code is available in PDF format.

[2] See `http://www.trolltech.com/`

[3] See `http://www.gnu.org/`

3 Future Directions

Further development of SPASS will concentrate on enhancements towards improved performance on problems from concrete application domains, like static program analysis.

References

Afshordel, B., Hillenbrand, T. & Weidenbach, C. (2001), First-order atom definitions extended, *in* R. Nieuwenhuis & A. Voronkov, eds, 'Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2001', Vol. 2250 of *LNAI*, Springer, pp. 309–319.

Antoniou, G. & Ohlbach, H. J. (1983), Terminator, *in* A. Bundy, ed., 'Proceedings of 8th International Joint Conference on Artificial Intelligence, IJCAI-83', pp. 916–919.

Bachmair, L. & Ganzinger, H. (1994), 'Rewrite-based equational theorem proving with selection and simplification', *Journal of Logic and Computation*.

Balbiani, P. (1995), Equation solving in geometrical theories, *in* N. Dershowitz & N. Lindenstrauss, eds, 'Proceedings of the 4th international workshop on conditional and typed rewriting systems', Vol. 968 of *LNCS*, Springer.

Brinker, C. (2000), Geometrisches Schliessen mit SPASS, Diplomarbeit, Universität des Saarlandes and Max-Planck-Institut für Informatik, Saarbrücken, Germany.

Downey, P. J., Sethi, R. & Tarjan, R. E. (1980), 'Variations on the common subexpression problem', *Journal of the ACM* **27**(4), 758–771.

Ganzinger, H., Meyer, C. & Weidenbach, C. (1997), Soft typing for ordered resolution, *in* 'Proceedings of the 14th International Conference on Automated Deduction, CADE-14', Vol. 1249 of *LNAI*, Springer, Townsville, Australia, pp. 321–335.

Hillenbrand, T., Jaeger, A. & Löchner, B. (1999), WALDMEISTER – improvements in performance and ease of use, *in* H. Ganzinger, ed., '16th International Conference on Automated Deduction, CADE-16', LNAI, Springer, pp. 232–236.

Nonnengart, A. & Weidenbach, C. (2001), Computing small clause normal forms, *in* A. Robinson & A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. 1, Elsevier, chapter 6, pp. 335–367.

Topić, D. (2002), Static analysis of list-processing C programs, Diplomarbeit, Universität des Saarlandes and Max-Planck-Institut für Informatik, Saarbrücken, Germany.

Weidenbach, C. (2001), Combining superposition, sorts and splitting, *in* A. Robinson & A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. 2, Elsevier, chapter 27, pp. 1965–2012.

Weidenbach, C., Afshordel, B., Brahm, U., Cohrs, C., Engel, T., Keen, E., Theobalt, C. & Topic, D. (1999), System description: SPASS version 1.0.0, *in* H. Ganzinger, ed., '16th International Conference on Automated Deduction, CADE-16', Vol. 1632 of *LNAI*, Springer, pp. 314–318.