System Description: SPASS Version 3.0

Christoph Weidenbach[1], Renate A. Schmidt[2], Thomas Hillenbrand[1], Rostislav Rusev[1], and Dalibor Topic[1]

[1] Max-Planck-Institut für Informatik, Germany, spass@mpi-sb.mpg.de
[2] University of Manchester, UK, spass@mpi-sb.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 3.0, including support for dynamic modal logics, relational logics and expressive description logics, additional renaming and selection strategies, and significant interface enhancements for human and machine users.

1 Introduction

New in SPASS 3.0 are facilities for supporting automated reasoning in a large class of related logics which we refer to as *EML* logics (extended modal logics). These include (traditional) propositional modal logics such as $K_{(m)}$, $KD_{(m)}$, $KT4_{(m)}$ etc., which are widely used for studying and formalizing e.g. multi-agent systems, but have many applications in other areas of computer science as well as mathematics, linguistics and philosophy. *EML* logics also include dynamic modal logics which are PDL-like modal logics in which the modal operators are parameterized by relational formulas [6]. These can be used to formalize dynamic notions such as actions or programs and are useful in linguistic and AI applications. Examples of dynamic modal logics are Boolean modal logic, tense logic, information logics, logics expressing inaccessibility and sufficiency as well as a large class of description logics. The *EML* class further includes relational logics, i.e. logical versions of Tarski's relation algebras. SPASS handles these logics by translation to first-order logic, see Sect. 2.

For most decidable *EML* logics, SPASS is actually a decision procedure on the first-order formulas resulting from the translation. For some logics, e.g., description logics including negation of roles, it is currently the only available decision procedure. SPASS is competitive even with special-purpose systems.

Further enhancements in SPASS 3.0 are additional renaming and selection strategies, see Sect. 3, and an improved user/machine interface including an extended formula-clause relationship handling, input and output of saturated clause sets and documentation, see Sect. 4 and 5.

2 Modal Logics, Relational Logics and Description Logics

The facilities in SPASS 3.0 for supporting automated reasoning in *EML* logics were first implemented from 1998 onwards in the MSPASS theorem prover [2,

Translation method	Options
relational translation	`-EMLTranslation=0` (default)
(monadic) functional translation	`-EMLTranslation=1`
polyadic functional translation	`-EMLTranslation=1 -EMLFuncNary=1`
(monadic) optimized functional translation	`-EMLTranslation=2`
polyadic optimized functional translation	`-EMLTranslation=2 -EMLFuncNary=1`
semi-functional translation	`-EMLTranslation=3`
relational-functional translation	`-EMLTranslation=0 -EMLFuncNary=1`
relational-relational translation	`-EML2Rel=1 [-EMLTranslation=0]`

Table 1. Available translation methods

4] as an extension of SPASS 1.0. This code has been upgraded and integrated into SPASS 3.0 (and FLOTTER 3.0) so that support for modal, relational and description logic reasoning is now immediately available to SPASS users and the latest SPASS technology is immediately available to MSPASS users.

The `dfg` input language of SPASS was extended to support the input of *EML* problems without changing the syntax for formulas in first-order logic or clause form. There are three types of *EML* formulas which can be used simultaneously in one file: first-order formulas, Boolean type formulas and relational type formulas. Boolean and relational type formulas can be constructed using common modal, relational and description logic operators. The pre-defined logical operators are:

- the standard Boolean operators (for all three types of formulas): `true`, `false`, `not`, `and`, `or`, `implies` (subsumed by), `implied` (subsumes), `equiv`,
- multi-modal operators with atomic or complex relational arguments: `dia` and `box` (synonyms are `some` and `all`), as well as `domain` and `range`,
- additional relational operators: `comp` (composition), `sum` (relative sum), `conv` (converse), `id` (the identity relation), `div` (the diversity relation), and
- `test` (test), `domrestr` (domain restriction) and `ranrestr` (range restriction).

We give three examples of *EML* formulas, two Boolean type formulas and one relational type formula.

```
prop_formula( implies(box(bel1,p), box(know1,box(bel1,p))) ).   (1)
concept_formula( implies(expert_AR,                             (2)
                 not(some(not(has_studied),proof_methods))) ).
rel_formula( implies(comp(r,r), r) ).                           (3)
```

(1) is an example from modal logic and says that if agent 1 believes `p` then it knows that it believes `p`, i.e. it is aware that it believes `p`. The example (2) is a description logic example; it says that an expert in automated reasoning is someone who has studied every proof method. This kind of example cannot be handled by current tableau-based description logic provers because it requires negation of roles. (3) expresses transitivity of a relation in relational logic (or in description logics).

Table 1 summarizes the implemented translation methods. The different translation methods are based on first-order encodings of the different ways

of defining the semantics of the logics. The basis for the *relational translation* method, or *standard translation* method, is the standard set-theoretic semantics of *EML* logics. It is implemented for all Boolean and relational *EML* formulas.

The basis for the different *functional translations* is the functional semantics of traditional modal logics. The *optimized functional translations* are obtained from the functional translations by a non-standard quantifier exchange operation, which is implemented by replacing non-constant Skolem terms by Skolem constants. The *polyadic functional translation* methods are variations of functional translation methods and differ in the way they encode world paths (transition sequences). The polyadic translations avoid the use of an extra function symbol by using n-ary predicates of different arities. The *semi-functional translation* approach is a mixture of the relational and functional translation approaches. It translates box modalities in the standard relational way, while diamond modalities are translated functionally. The (monadic) functional translations and the semi-functional translation are implemented for the basic multi-modal logic $K_{(m)}$ possibly with serial (total) modalities, plus frames or models, and non-logical axioms. The corresponding description logics are $\mathcal{ALC}$ with concept ABox and TBox statements, possibly with total roles. The polyadic functional translations are implemented for the basic multi-modal logic $K_{(m)}$ possibly with D (serial) modalities. The corresponding description logic is $\mathcal{ALC}$ possibly with total roles.

The *relational-functional translation* method, or *tree-layered relational translation*, is a variation of the relational translation specialized for the basic modal logic $K_{(m)}$. The *relational-relational translation* converts Boolean *EML* formulas into the relational formulas via a cylindrification operation. This translation is implemented for $K_{(m)}$.

All translation methods are sound and complete for the logics they are implemented for and have linear time complexity.

There are various additional *EML* options. For example, the `-EMLTheory` option can be used to add standard relational properties to the background theory. There are also options for varying the translation methods and the preprocessing done on *EML* formulas. The functional translations can be varied slightly with the two options `-EMLFuncNdeQ` and `-EMLFFSorts`. With the option `-EMLElimComp=1` top-level occurrences of relational composition in modal parameters can be eliminated as part of preprocessing. When enabled, the option `-QuantExch=1` causes non-constant Skolem terms in the clausal form to be replaced by constants. The option is automatically set for the optimised functional translation methods. The option can also be used for classical formulas and clauses, but it is not sound in general and therefore switched off by default.

With the new *EML* facilities SPASS supports reasoning for *EML* logics with the following additional additional features.

1. For dynamic modal logics and relational logics: non-logical axioms, modal operators characterized by any first-order frame correspondence properties and accessibility relations satisfying any first-order properties, specification of concrete worlds as constants, (first-order) relationships between concrete worlds, specifications of frames and models.

2. For description logics: the corresponding features, including in particular terminological axioms, TBox and RBox statements, and ABox statements for concept and role expressions.

Because SPASS is a first-order superposition based prover its capabilities as a modal, relational or description logic prover are very different and more varied than those of other provers for these logics. It is possible to use SPASS as a decision procedure for a large class of *EML* logics. For instance, it decides extensions of Boolean modal logic with converse, domain/range restriction, and positive occurrences of composition, and the corresponding description logics, i.e. extensions of $\mathcal{ALB}$ with positive occurrences of composition. No other (special-purpose) prover currently decides these logics. SPASS can be used as a decision procedure for many solvable first-order fragments including the guarded fragment, Maslov's class K, first-order logic in two variables, the clausal class DL*, and many decidable quantifier prefix classes. Using the new features it is possible to approximate the behaviour of modal and description logic tableau provers with SPASS. Additionally, it can be used as a model finder.

For definitions of the different translation methods and further details, the various applications and references to original work, the reader is invited to consult the survey paper [6]. The paper [3] surveys decidable first-order fragments relevant to description logics. References to resolution decision procedures of *EML* logics and first-order fragments can be found in both [6] and [3].

3 Renaming and Selection Enhancements

Renaming transformations, or structural transformations, are not standard in current first-order theorem provers but have various advantages. They are essential for linear conversion of first-order formulas into clausal form. They are useful to control the way the search is performed in order to enhance the performance of a prover or to define decision procedures. They preserve the structure of the input formulas and make it easier to read resolution derivations and translate them back into first-order logic or the original *EML* logic. Renaming transformations introduce new predicate symbols for subformulas in the input problem. The renaming strategy available in SPASS 2.2 [7] is aimed at minimizing the number of eventually generated clauses [5]. A subformula is renamed if the replacement of the subformula by an atom headed with a new Skolem predicate plus the definition of the Skolem atom eventually results in fewer clauses. We added two more renaming strategies to SPASS 3.0: *complex formula renaming* and *quantified formula renaming*. Complex formula renaming introduces a new Skolem predicate for any subformula that is not an atom and does not start with a negation symbol. Quantified formula renaming introduces a new Skolem predicate for any subformula that starts with an existential or universal quantifier. The definition formulas for the Skolem predicates are generated in a polarity dependent way for all three versions. The renaming strategy is controlled via the `-CNFRenaming=`n flag where $n = 1$ selects minimizing renaming, $n = 2$ selects complex renaming, and $n = 3$ selects quantified renaming.

For the finite saturation of many first-order theories it is indispensable to select certain literals in order to protect variables in different literals of the same clause. An example is the (simplified) formalization of LAN router functionality that contains clauses of the form below. The clause states that if a packet is to be routed to the destination `xdst` and there is a route entry saying that all destination addresses "anded" with the mask `xmsk` produce the network `xnet` can be forwarded to `xhop`, then the packet is actually forwarded to `xhop`.

$$[\texttt{RouteIP}(\texttt{packet}(\texttt{xsrc}, \texttt{xdst}, \texttt{xpld})), \texttt{RouteEntry}(\texttt{xmsk}, \texttt{xnet}, \texttt{xhop}),\\ \texttt{ipand}(\texttt{xdst}, \texttt{xmsk}) \approx \texttt{xnet}] \rightarrow \texttt{Send}(\texttt{xhop}, \texttt{packet}(\texttt{xsrc}, \texttt{xdst}, \texttt{xpld}))$$

As all symbols starting with '`x`' are variables, superposition left inferences can produce many clauses with the clause above and the theory for logical "and" on bit vectors (the function `ipand`). If the literals `RouteIP` and `RouteEntry` are selected then these inferences can be avoided.

Such situations are supported by enhanced selection mechanisms in SPASS 3.0. First, via the command `set_selection` a list of predicates can be defined in the input file to be candidates for selection. Second, this list can be combined with the following selection strategies that are set via the `-Select=`n option. For $n = 1$ in any clauses with more than one maximal literal one negative literal is selected. Either a negative literal with a predicate from the selection list is chosen or if no such negative literal is available, a negative literal with maximal weight is chosen. For $n = 2$ in any clause containing at least one negative literal, a negative literal is selected. Again, either a negative literal with a predicate from the selection list is chosen or if no such negative literal is available, a negative literal with maximal weight is chosen. For $n = 3$ in any clause containing negative literals with predicates specified by the selection list, one negative literal out of these is selected.

In SPASS 3.0 we changed the heuristic for selecting splitting clauses. SPASS now selects the clause with the highest unit reduction potential after splitting.

4 Interface Enhancements

Starting with SPASS 3.0, FLOTTER writes the formula-clause relation into the setting part of the clause normal form output file. When processing such a `.cnf` file SPASS is now also able to tell which input formulas were used in an eventually found proof.

When SPASS finitely saturates a set of clauses, the result can be output to a file via the `-FPModel` option. So far the generated file did not contain any information about the selection of literals. Hence, it could happen that running SPASS again on such a file would produce further clauses by inferences. With SPASS 3.0 we have defined an additional clause input format that is similar to the clause output given by SPASS at run time and includes the possibility to mark negative literals in clauses to be selected. In SPASS 3.0 this format is used when `-FPModel` is set and inferences on saturated sets produce no additional clauses when such sets are resubmitted to SPASS.

At run time, SPASS now selects literals before it prints the `Given clause`. This improves manual inspection of the SPASS output.

5 Conclusion and Future Work

Finally, we want to point the reader to the new handbook [8] distributed with SPASS. It contains detailed descriptions of the most important features and facilities implemented in the prover, covering the sophisticated reasoning technology, the superposition calculus implemented in SPASS, the theory and implementation details for the translation methods for *EML* logics, and the theory behind the clause set transformations of `dfg2dfg`. A detailed, formal specification of the extended input language can be found in the SPASS documentation [9]. It also includes examples of input files for the different *EML* logics supported by SPASS.

A start has been made at implementing the techniques introduced and studied in [1] for the bottom-up model generation paradigm in SPASS. Moreover, we are developing efficient superposition based reasoning techniques for finite domains, further improving the performance of the prover for several *EML* logics.

SPASS 3.0 is available from `http://spass.mpi-sb.mpg.de`.

Acknowledgements. We thank the SPASS user community for delivering enhancement requests as well as bug reports, and the reviewers for their comments.

References

1. P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In *Automated Reasoning: IJCAR 2006*, vol. 4130 of *LNAI*, pp. 125–139. Springer, 2006.
2. U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In *Proc. TABLEAUX 2000*, vol. 1847 of *LNAI*, pp. 67–71. Springer, 2000.
3. U. Hustadt, R. A. Schmidt, and L. Georgieva. A survey of decidable first-order fragments and description logics. *J. Relational Meth. in Computer Sci.*, 1:251–276, 2004.
4. U. Hustadt, R. A. Schmidt, and C. Weidenbach. MSPASS: Subsumption testing with SPASS. In *Proc. DL'99*, pp. 136–137. Linköping University, 1999.
5. A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In *Handbook of Automated Reasoning*, pp. 335–367. Elsevier, 2001.
6. R. A. Schmidt and U. Hustadt. First-order resolution methods for modal logics. In *Volume in memoriam of Harald Ganzinger*, LNCS. Springer, 2006. To appear, `http://www.cs.man.ac.uk/~schmidt/publications/SchmidtHustadt06a.html`.
7. C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS version 2.0. In *Automated Deduction: CADE-18*, vol. 2392 of *LNAI*, pp. 275–279. Springer, 2002.
8. C. Weidenbach, R. A. Schmidt, and E. Keen. SPASS handbook version 3.0. Contained in the distribution of SPASS Version 3.0, 2007.
9. C. Weidenbach, R. A. Schmidt, and D. Topic. SPASS input syntax version 3.0. Contained in the distribution of SPASS Version 3.0, 2007.