Appears in T. Eiter et al. (Eds.): LPNMR 2001, LNAI 2173

The Smodels System

Tommi Syrjänen* and Ilkka Niemelä*

Helsinki University of Technology, Dept. of Computer Science and Eng.,
Laboratory for Theoretical Computer Science,
P.O.Box 5400, FIN-02015 HUT, Finland
{Tommi.Syrjanen,Ilkka.Niemela}.@hut.fi

1 Introduction

The `Smodels` system is an Answer Set Programming (ASP) implementation based on the stable model semantics of normal logic programs. The basic idea of ASP is to encode the constraints of a problem as a logic program such that the answer sets (stable models) of the program correspond to the solutions of the problem. Then we can solve the problem by letting a logic program engine to find the answer sets of the program.

The `Smodels` system provides such an engine for computing answer sets. It extends the class of normal logic programs with cardinality and weight constraints as well as arithmetic built-in functions. Additionally, function symbols are also supported. However, the input programs have to be *domain-restricted* in a sense that will be explained in Section 2. The `Smodels` system is available for download at http://www.tcs.hut.fi/Software/smodels.

For a practical `Smodels` example, consider the following logical puzzle: *On the Island of Knights and Knaves there are two types of persons: knights, who always tell the truth, and knaves, who always lie. A visiting logician met two natives, A and B. When asked about their types, A answered: "We are both knaves." What are their types?* We can solve the puzzle with the following `Smodels` program:

```
person(a; b). type(knight; knave).
1 { is_type(P, T) : type(T) } 1 :- person(P).
true_statement :- is_type(a, knave), is_type(b, knave).
   :- is_type(a, knight), not true_statement.
   :- is_type(a, knave), true_statement.
```

Here the first line defines facts for the persons and types, and the second line assigns a unique type for all persons. The third line tells that A's answer is true if both of them are knaves and the last two rules enforce the condition that knights never lie and knaves never tell the truth. The constructs of the form $L\ \{\ l_1, l_2, \ldots, l_n\}\ U$ are *cardinality constraints* that are satisfied whenever the number of satisfied literals l_i is between the integral bounds L and U, inclusive. A cardinality constraint in a rule head imposes a non-deterministic *choice* over the

* This work has been funded by Academy of Finland (project no. 43963). The work of the first author has been supported by HeCSE and Tekniikan edistämissäätiö.

literals in it when the rule body is satisfied. The construct $a(X) : b(X)$ denotes the set of atoms $a(X)$ for which $b(X)$ also holds. For example, $is\_type(P, T) : type(T)$ denotes the set $\{is\_type(P, knight), is\_type(P, knave)\}$.

The `Smodels` system itself consists of two parts, `smodels`, which is the actual inference engine, and `lparse`, a front-end that instantiates and simplifies user programs. If we have stored the above program in a file `puzzle.lp`, we can find its answer sets by invocating `Smodels` as follows:

```
% lparse -d none puzzle.lp | smodels 0
smodels version 2.26. Reading...done
Answer: 1
Stable Model: is_type(a,knave) is_type(b,knight)
False
```

We see that in the unique solution A is a knave and B is a knight. The `lparse` argument '`-d none`' discards the domain predicates *person* and *type* from the output, and the argument '`0`' asks `smodels` to compute all stable models.

2 Theoretical Background

The `Smodels` system has a declarative formal semantics that extends the stable model semantics of normal logic programs with cardinality and weight constraints. Intuitively, a set of atoms is a stable model of if it satisfies all rules of the program and each atom in the model occurs as a head of a rule with a satisfied body. A program may have none, one, or many stable models. The details of the semantics are explained in Niemelä and Simons [8].

The predicate symbols of a `Smodels` program P are automatically divided into two classes, *domain* and *non-domain predicates*. The domain predicates of P are the predicates that are not defined in terms of negative recursion or using choice rules, and they form a stratified hierarchy where complex domains are defined in terms of simple ones. The intuition of domain predicates is that they are used to define the set of terms over which the variables range in each rule of the program. All rules in a program have to be *domain-restricted* in the sense that each variable in a rule has to occur also in a positive domain predicate in the rule body. In a rule defining a domain predicate this *domain literal* has to belong to a strictly lower stratum than the head of the rule. This syntactic restriction is strong enough to guarantee that the problem of finding an answer set stays decidable even when function symbols are allowed since it ensures that only a finite number of ground atoms can be derived using a single rule [11]. Consider the following program:

```
number(0..n). even(0).
even(X+1) :- number(X), odd(X).
odd(X+1) :- number(X), even(X).
interesting(X) :- odd(X), not dull(X).
dull(X) :- odd(X), not interesting(X).
```

The predicate *number* is trivially a domain predicate since it does not depend on any other predicates. Here n is a numeric constant that can be defined from the command line. Predicates *even* and *odd* are also domain predicates since they depend only on each other and on *number* that gives the domain for X in the two recursive rules. If we left the atom $number(X)$ out of the rule bodies, the stable models of the program would be infinite. The predicates *interesting* and *dull* are not domain predicates since they depend negatively on each other.

The computational complexity of determining whether there exists an answer set of a propositional `Smodels` program is NP-complete. For function-free programs with variables, it is EXP-complete and when function symbols are used, it is 2-EXP-complete [11]. Thus, the high expressive power has a significant computational cost.

3 Implementation

The `Smodels` system is composed of two independent components, `smodels` and `lparse`, that are both implemented in C++. The `smodels` finds answer sets of variable-free primitive logic programs using a Davis-Putnam like backtracking search procedure [9]. It also uses inherent properties of the stable model semantics to infer and propagate truth values to prune the search space. The `lparse` is a front-end that translates user programs into the `smodels` internal format by instantiating non-ground rules and simplifying complex constructs.The `lparse` first creates the dependency graph of the program and identifies the domain predicates by computing strongly connected components of the graph. Next, the extensions of the domain predicates are computed and they are used to instantiate the rest of the rules.

4 Applications

The `Smodels` system is actively used in dozens of research groups all over the world. Next we briefly review some interesting application areas already studied.

Planning is a potential application area for ASP and `Smodels` has been used in a number of approaches, see e.g. [4, 10]. An interesting example of an advanced project is research at Texas Tech University on developing a decision support system for the flight controllers of space shuttles using the ASP approach [3].

General methodology for product configuration using ASP techniques has been developed and interesting application projects have been started, see e.g. the WeCoTin project (http://www.soberit.hut.fi/WeCoTin/). Research on software configuration using ASP has lead to a prototype configurator for the Debian Linux system distribution (http://www.tcs.hut.fi/~tssyrjan/configuration/).

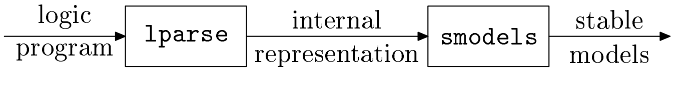

Fig. 1. System Architecture

`Smodels` has been used for a variety of key inference tasks in computer aided verification. An analysis method for Petri nets based on finite complete prefixes has been built on top of `Smodels`, see the `mcsmodels` and `unfsmodels` tools (http://www.tcs.hut.fi/~kepa/tools/). Similarly, it has been used in a stubborn set reduction method for Petri net reachability analysis, see the `prod` tool (http://www.tcs.hut.fi/Software/prod/). Recently, a symbolic model checking method for asynchronous systems based on bounded model checking techniques and ASP has been developed [5].

There are a number other interesting areas where `Smodels` has been employed including logical cryptanalysis [6], security protocol analysis [1], network inhibition analysis [2], and computation of (total and partial) stable models for disjunctive programs [7].

References

1. L.C. Aiello and F. Massacci. An executable specification language for planning attacks to security protocols. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 88–102, Cambridge, UK, 2000. IEEE Computer Society Press.
2. T. Aura, M. Bishop, and D. Sniegowski. Analyzing single-server network inhibition. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 108–117, Cambridge, UK, July 2000. IEEE Computer Society Press.
3. M. Balduccini, M. Barry, M. Gelfond, M. Nogueira, and R. Watson. An A-Prolog decision support system for the space shuttle. In *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages*. Springer-Verlag, 2001.
4. Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in non-monotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pages 169–181, Toulouse, France, September 1997. Springer-Verlag.
5. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. In *Proceedings of the 6th International Conference on Logic Programming and Non-monotonic Reasoning*, Vienna, Austria, September 2001. Springer-Verlag.
6. M. Hietalahti, F. Massacci, and I. Niemelä. DES: a challenge problem for non-monotonic reasoning systems. In *Proc. of the 8th International Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA, April 2000. cs.AI/0003039.
7. T. Janhunen, I. Niemelä, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 411–419, Breckenridge, Colorado, USA, April 2000. Morgan Kaufmann Publishers.
8. I. Niemelä and P. Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, chapter 21, pages 491–521. Kluwer Academic Publishers, 2000.
9. P. Simons. Extending and implementing the stable model semantics. Research Report 58, Helsinki University of Technology, Helsinki, Finland, 2000.
10. T.C. Son, C. Baral, and S. McIlraith. Extending answer set planning with sequence, conditional, loop, non-deterministic choice, and procedure constructs. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming*, pages 202–209, Stanford, USA, March 2001. AAAI Press.
11. T. Syrjänen. Omega-restricted logic programs. In *Proc. of the 6th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*. Springer-Verlag, 2001.