Computing Answer Sets of a Logic Program via-enumeration of SAT certificates

Yuliya Lierler[1] and Marco Maratea[2]

[1]Department of Computer Sciences, University of Texas, Austin, USA
[2]Dipartimento di Informatica, Sistemistica e Telematica, Università di Genova, Genova, Italy
[1]yuliya@cs.utexas.edu, [2]marco@mrg.dist.unige.it

Abstract

Answer set programming is a new programming paradigm proposed in [1] and [2], and based on the answer set semantics of Prolog [3]. It is well known that an answer set for a logic program is also a model of the program's completion [4]. The converse is true when the logic program is "tight" [6, 5]. Lin and Zhao [7] showed that for non-tight programs the models of completion which do not correspond to answer sets can be eliminated by adding to the completion what they called "loop formulas". Nevertheless, their solver ASSAT[1] has some disadvantages: it can work only with basic rules, and it can compute only one answer set. Answer set solver CMODELS-1[2] [12] is a system that computes answer sets for logic programs that are tight or can be transformed into tight programs, and does not suffer from these limitations.

We are going to present a new system CMODELS-2[1], that is able to fix ASSAT's disadvantages. Another attractive feature of the new system is that it organizes the search process more efficiently then ASSAT, because it does not explore the same part of the search tree more than once. In the rest of the paper we will omit number 2 in the name of the system.

1 General Information

The input language of CMODELS can be generated by the preprocessor LPARSE. The input may contain negation as failure, cardinality expressions ("constraint literals" in the terminology of [9, Section 5.3]) and choice rules [9, Section 5.4]—two constructs widely used in answer set programming.[3] These constructs are eliminated in favor of nested expressions in the sense of [10] using the method described in [11]. CMODELS is an answer set solver, that uses SAT solvers as a search engines. It may invoke different SAT solvers for finding solutions, namely SIMO [4], MCHAFF and ZCHAFF [5], and RELSAT [6]. The system can be easily extended to new SAT solvers. This allows us to take advantage of the rapid progress in the area of satisfiability solvers.

[1] http://assat.cs.ust.hk/
[2] http://www.cs.utexas.edu/users/tag/cmodels/
[3] The input can also contain general weight expressions ("weight literals"). However, optimize statements [9, Section 5.6] are not allowed.
[4] http://www.mrg.dist.unige.it/~sim/simo/
[5] http://www.ee.princeton.edu/~chaff/
[6] http://www.satlib.org/Solvers/SAT/REL_SAT.2/

The algorithm of CMODELS for tight programs is identical to the one in CMODELS-1 and is described in details in [12]. The difference appears only when the program is non-tight. In such case CMODELS needs to verify that each model of completion is indeed the answer set. Whenever the model of completion does not correspond to any answer set the computation of loop formulas is performed. We will talk about the mechanism of dealing with non-tight programs in section 3.

2 Options of the solver

CMODELS's command line is:

`cmodels number [-mc] [-zc] [-si] [-sia] [-rs] [rs1] [-t] [-nt] [-s] [-le] [-bj]`

where `number` specifies the number of answer sets to be found. 0 stands for "compute all answer sets"; 1 is the default.

CMODELS has the possibility to invoke different SAT solvers as back-end solver:
`-mc` MCHAFF is used for finding answer sets (default).
`-zc` ZCHAFF is used for finding answer sets.
`-si` SIMO is used for finding answer sets.
`-sia` SIMO is used for finding answer sets with an ASSAT-like algorithm.
`-rs` RELSAT version 2 is used for finding answer sets.
`-rs1` RELSAT version 1.1.2 is used for finding answer sets.

Moreover, there are some switches related to checks and simplifications:
`-t` Stands for omitting the tightness check [12]; CMODELS expects input program to be tight. By default, the tightness check is performed.
`-nt` Stands for omitting the tightness check [12]; CMODELS expects input program to be non-tight and invokes the procedures related to non-tight programs.
`-s` Stands for omitting the simplification step [12]. By default the simplification is performed.

Finally, a couples of switches related to the communication with the SAT solver, when the SAT solver used is SIMO:
`-le` and `-bj` are relevant to the invocation of SIMO. They refer to "learning" and "back-jumping" using SAT terminology; by default [-le] is performed. For more details see [13].

3 CMODELS on Non-tight Programs

In case of non-tight programs the algorithm of CMODELS is similar to the one of system ASSAT described in [7], but there are several differences in the algorithms.

First, CMODELS works with more general programs, and uses extended definition of a loop formula [8]. The technique for computing loop formulas is nevertheless very similar to the one described in [7]. Second, CMODELS introduces a "generate and test" approach in accordance to the SAT-based methodology [13] with such SAT solvers as SIMO and ZCHAFF. Basically, according to the SAT-based methodology, we first:

- *generate* a total propositional model satisfying the propositional clauses, and then
- *test* if the generated valuation is an answer set.

Also ASSAT uses a "generate and test" approach, but using CMODELS:

- the SAT solver never explores the same part of the search tree (this is only partially true in case of ZCHAFF communication [7]).
- it is possible to compute more than one answer set.

The main difference between the two algorithms is: when the propositional model is not an answer set, instead of clausifying the loop formulas, giving all clauses to the SAT solver and restarting the search from scratch, CMODELS finds only one clause ("reason" using SAT terminology) unsatisfied by the current propositional model and satisfied by one of the loop formulas, without restarting the search from scratch. During this computation, the SAT solver is "frozen" in its state. It uses the reason to restart the search from the previous state.

In this way, CMODELS avoids the potential exponential blow up in the number of propositional clauses.

Further details will be presented in a future paper.

4 Experimental analysis with CMODELS

In this section we report some experimental data. All experiments were run on two identical Pentium IV 1.8GHz processors with 512MB of RAM, DDR 266MHz, running Linux.

In Figures 1, 2 and 3, CPU time is reported in seconds as the sum of *user* and *system* time using *time* UNIX command. *timeout* means that the process was stopped after one hour, *mem* means that the process reached the memory limit of the machine. The number in parentheses is the number of times CMODELS checked that a propositional model does not correspond to an answer set. In the tables, CMODELS uses SIMO with its default configuration or together with an implementation of an ASSAT-like algorithm in CMODELS and uses MCHAFF with the ASSAT-like algorithm. We compare CMODELS only with SMODELS: this is due to the fact that DLV is optimized for disjunctive logic programs and ASSAT works with programs of more limited syntax. Nevertheless, we have implemented an ASSAT-like algorithm in our framework. We will focus on finding one answer set.

First we evaluate the performances of CMODELS on Hamiltonian circuits (HC) benchmarks, in particular on publicly available complete graphs and on some hand-coded graphs from. [8] We also built some bigger complete graphs. Complete graphs are in particular interesting because, as observed in [7], they have exponential number of loops. In Figures 1 and 2 we present the comparison of CMODELS using SIMO with "learning", SMODELS and CMODELS using SIMO employing the algorithm of ASSAT.

For complete graphs, we can see a clear edge between CMODELS running SIMO with "learning" and as black-box communication. This seem to point out the usefulness of a communication with a SAT solver not treated as black-box. The advantage in comparison with SMODELS is smaller: around a factor of two.

For hand-coded graphs, first some informations about the instances. nxpm.i means n complete graphs with m vertexes connected by i arcs. If ".i" is not present, this means that the m complete graphs are not connected. The results on this domain are quite negative for CMODELS running SIMO with "learning", in particular respect to the black-box

[7] CMODELS uses an incremental learning option of ZCHAFF.

[8] http://assat.cs.ust.hk/assat-1.0.html

Instance name	CMODELS SIMO (-le)	SMODELS	CMODELS SIMO assat alg.
np10c	(4)0.03	0.01	(6)0.09
np20c	(9)0.13	0.07	(16)1.46
np30c	(14)0.45	0.45	(20) 5.21
np40c	(42) 1.56	2.49	(27) 16.52
np50c	(108) 5.47	8.66	(9) 10.17
np60c	(106) 8.80	21.45	(35) 76.12
np70c	(217) 26.46	42.86	(41) 139.50
np80c	(223) 37.87	79.78	(44) 241.55
np90c	(356) 77.15	131.26	mem
np100c	(286) 78.93	200.43	(51) 561.94
np120c	(698) 314.93	430.98	mem
np150c	(1074) 841.91	1171.38	mem

Fig. 1. Complete graphs CMODELS employing learning vs. SMODELS vs. CMODELS employing assat algorithm.

Instance name	CMODELS SIMO (-le)	SMODELS	CMODELS SIMO assat alg.
2xp30	(0)0.01	0.01	(0)0.01
2xp30.1	timeout	0.12	(90)57.58
2xp30.2	(155)3092.13	timeout	(152)24.12
2xp30.4	timeout	timeout	timeout
4xp20	(0)0.01	0.01	(0)0.01
4xp20.1	(2) 73.26	timeout	(1) 2.03
4xp20.3	(13) 82.90	0.01	(5) 1.56

Fig. 2. Hand-coded graphs CMODELS employing learning vs. SMODELS vs. CMODELS employing assat algorithm.

communication. In these problems seem that is very useful to add all the loop formulas when the propositional model in not an answer set, instead of adding only the reason.

The second domain we demonstrate in Fig. 3 is bounded LTL model checking problem[9]. The comparison is between CMODELS using MCHAFF employing the algorithm of ASSAT, CMODELS using SIMO with "learning" and SMODELS. In this domain, we can see a clear edge between CMODELS using SIMO both versus CMODELS using MCHAFF as a black-box and SMODELS.

Finally, we want to point out that: First, in general the number of propositional models checked is less when we use an ASSAT-like algorithm, but this does not pay off in terms of cpu time; second, a more "strict" communication with a SAT solver seems to improve performances in particular on instances arising from real world applications.

5 Conclusions and future works

In this paper we have presented CMODELS-2, a SAT-based answer set solver that is competitive with other state-of-the-art answer set solver, and it can outperform other solvers

[9] `http://www.tcs.hut.fi/~kepa/experiments/boundsmodels/`

Instance Name	CMODELS MCHAFF	CMODELS SIMO (-le)	SMODELS
dp_6.formula1-s-O2-b7	(9) 0.89	(11) 0.30	0.19
dp_6.formula1-i-O2-b8	(28) 6.17	(6) 0.43	0.46
dp_8.formula1-s-O2-b8	(15) 5.14	(14) 0.94	1.83
dp_8.formula1-i-O2-b10	(24) 28.72	(41) 6.61	5.08
dp_10.formula1-s-O2-b9	(24) 19.47	(36) 3.51	29.05
dp_10.formula1-i-O2-b12	(21) 51.36	(162) 14.34	428.85
dp_12.formula1-s-O2-b10	(69) 96.95	(93) 7.56	949.95
dp_12.formula1-i-O2-b14	(29) 469.83	(14) 81.80	timeout

Fig. 3. Non-tight Bounded Model Checking CMODELS using MCHAFF employing ASSAT-like algorithm and SIMO employing learning vs. SMODELS

on particular domains. Although the interesting results we have presented, we are planning to run our system on other domains and to evaluate other systems. Moreover, we would like to evaluate new techniques in the system, in particular the possibility of using some heuristic guided from the theory built on top of the SAT solver. Finally, it would be also interesting to understand if it is possible to extend our approach to disjunctive logic programs, using the results presented in [8].

Acknowledgments
We are grateful to Enrico Giunchiglia and Vladimir Lifschitz for their comments and suggestions related to the subject of this paper. We are also grateful to Armando Tacchella for his help related to the integration of the systems. This work is partially supported by MIUR and ASI.

References

1. Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 1999.
2. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.
3. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.
4. Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
5. Yuliya Babovich, Esra Erdem, and Vladimir Lifschitz. Fages' theorem and answer set programming.[10] In *Proc. Eighth Int'l Workshop on Non-Monotonic Reasoning*, 2000.
6. François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
7. Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proc. AAAI-02*, 2002.

[10] `http://arxiv.org/abs/cs.ai/0003042.`

8. Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs.[11] In *Proc. ICLP-03*, To appear.
9. Tomi Syrjanen. Lparse manual.[12] 2003.
10. Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.
11. Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. Theory and Practice of Logic Programming, to appear.
12. Yuliya Babovich and Vladimir Lifschitz. Computing answer sets using program completion[13] Submitted to LPNMR-7.
13. Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, Fausto Giunchiglia and Armando Tacchella. SAT-Based Decision Procedures for Automated Reasoning: a Unifying Perspective. In Festschrift in Honor of Jörg H. Siekmann, to appear in LNAI, Springer-Verlag 2002.

[11] `http://www.cs.utexas.edu/users/appsmurf/papers/disjunctive.ps.`
[12] `http://www.tcs.hut.fi/software/smodels/lparse.ps.gz.`
[13] `http://www.cs.utexas.edu/users/yuliya/cmodels.ps.`