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

Yuliya Lierler[1] **and** **Marco Maratea**[2]

yuliya@cs.utexas.edu

marco@dist.unige.it

[1]University of Texas at Austin

[2]MRG-LAB DIST, Università di Genova

CMODELS - ASP tool

CMODELS is an answer set programming system, which computes answer sets of a logic program using SAT solver.

CMODELS first finds the completion of a program and then applies SAT solver for finding the models of completion.

Depending on tightness property of a program the completion may be extended and SAT solver may be invoked several times into the computation.

Earlier work:

- SMODELS
- LPARSE

Accepted syntax

CMODELS accepts programs that include the following rules:

Basic rules $a \leftarrow a_1, \ldots, a_n,$ **not** $a_{n+1}, \ldots,$ **not** a_m

Choice rules $\{a, \ldots, a_k\} \leftarrow a_{k+1}, \ldots, a_m,$ **not** $a_{m+1}, \ldots,$ **not** a_n

Weight constraint rules $a \leftarrow L\{a_1 = w_1, \ldots, a_m = w_m,$
not $a_{m+1} = w_{m+1}, \ldots,$ **not** $a_n = w_n\}U$

Basic nested rules

A basic nested rule is an expression of the form:

$$a \leftarrow a_1, \ldots, a_n, \textbf{not } a_{n+1}, \ldots, \textbf{not } a_m, \textbf{not not } a_{m+1}, \ldots, \textbf{not not } a_n.$$

A *basic nested program* is a finite set of basic nested rules.

Translating an input program into a basic nested program

Choice rules are translated into basic nested rules.

Weight constraint rules are eliminated by introducing auxiliary atoms, Ferraris and Lifschitz (2003).

The concept of completion for nested programs is defined by Lloyd and Topor (1984).

Translating a logic program: Examples (1)

The logic program:

$$\{p, q\}$$
$$r \leftarrow p$$
$$r \leftarrow q$$

is translated into the following program with nested expressions:

$$p \leftarrow \textbf{not not } q$$
$$q \leftarrow \textbf{not not } p$$
$$r \leftarrow p$$
$$r \leftarrow q$$

Translating a logic program: Examples (2)

The rule:

$$p \leftarrow 3\{q = 3, r = 2, s = 2\}$$

is first replaced by four rules:

$$p \leftarrow aux_1$$
$$aux_1 \leftarrow 3\{r = 2, s = 2\}$$
$$p \leftarrow q, aux_2$$
$$aux_2 \leftarrow 0\{r = 2, s = 2\}$$

(note that the last expression is identically true). And recursively till:

$$p \leftarrow aux_1$$
$$aux_1 \leftarrow r, aux_4$$
$$aux_4 \leftarrow s$$
$$p \leftarrow q$$

Agenda

- Completion
- Tight and nontight logic programs
- Loop formulas
- ASSAT
- CMODELS-2
- Conclusions and future work

Completion

Given a basic nested program Π , for every atom a make the list of all rules in Π with the head a:

$$a \leftarrow Body_i$$

and form the equivalence:

$$a \equiv \bigvee_i Body_i \qquad (1)$$

and, for every constraint $\bot \leftarrow Body$ in Π form the negation of its body:

$$\neg Body \qquad (2)$$

The *completion* *Comp*(Π) of Π consists of all formulas (1) and (2).

Completion: Examples

The completion of the program:

$$p \leftarrow \textbf{not } q$$
$$q \leftarrow \textbf{not } p$$
$$r \leftarrow p$$
$$r \leftarrow q$$

is the set of propositional formulas: $\{p \equiv \neg\, q, q \equiv \neg\, p, r \equiv p \vee q\}$,

and the completion of:

$$p \leftarrow q, r$$
$$q \leftarrow \textbf{not } r$$

is: $\{p \equiv q \wedge r, q \equiv \neg\, r, r \equiv \bot\}$.

Tight logic programs

Tightness is a syntactic condition on a logic program.

To verify tightness of a program Π we build the *positive dependency graph* G which corresponds to Π. Each atom of Π is a vertex of G and for each rule $a \leftarrow \ \ldots, b, \ldots$ in Π there is an edge between a and b in G.

The program Π is *absolutely tight* iff there are no cycles in the graph of Π.

Before checking the tightness, CMODELS applies two transformations to the program that can modify the property:

- drop the rule $a \leftarrow \ldots, a, \ldots$
- replace all occurrences of a with **not not** a in the positive parts of the bodies of all rules if the program contain $a \leftarrow$ **not not** a

Nontigth logic programs

Nontigth logic programs are programs in which there are *cycles* in the correspondent graphs.

Cycles (or loops) cause the difference between answer sets of a program and models of program's completion.

Example:
$p \leftarrow p$ is not tight; it has two model of completion $\emptyset$ and $\{\mathrm{p}\}$ and only one answer set $\emptyset$.

Lin and Zhao proposed in 2002, with their solver ASSAT, a method for using SAT solvers to find answer sets of nontight logic programs.

The main idea is: adding "loop formulas" to the completion eliminates "bad" models of completion.

Loop formulas

If L is a loop in a program then the loop formula is build from the subset $R^-(L)$ of program's rules define by Lin and Zhao (2002) as:

$$R^-(L) = \{p \leftarrow G \mid p \in L, \neg(\exists\, q).q \in G \wedge q \in L\}$$

Let $p \leftarrow G_i$ belong to $R^-(L)$, than the loop formula associated with L has the form:

$$\neg(\bigvee_i G_i) \supset \bigwedge_{p \in L} \neg p$$

Extended definition of loop formulas

So far, loop formulas have been defined for programs with rules of the form:

$$a \leftarrow a_1, \ldots, a_n, \textbf{not } a_{n+1}, \ldots, \textbf{not } a_m.$$

Extended loop formulas (due to Lee and Lifschitz 2003) are defined for programs with nested expressions, in particular to programs with rules of the form:

$$a \leftarrow a_1, \ldots, a_n, \textbf{not } a_{n+1}, \ldots, \textbf{not } a_m, \textbf{not not } a_{m+1}, \ldots, \textbf{not not } a_n.$$

ASSAT's algorithm

Given a logic program Π:

1. Let T be the Comp(Π)
2. Find a model M of T. If there is no such model, then terminate with failure
3. If M is an answer set, then exit with it
4. If M is not an answer set, then find a loop L such that its loop formula ϕ_L is not satisfied by M
5. Let T be T U $\{\phi_L\}$ and go back to step 2

ASSAT's disadvantages

The system ASSAT has some disadvantages:

- covers only basic rules
- finds only one answer set
- may explore the same parts of the search tree already explored
- may blow up in the number of propositional clauses

CMODELS-2

CMODELS-1 (Babovich and Lifschitz 2003), is a system that does not suffer of the ASSAT's first and second disadvantages, but is limited to tight programs.

CMODELS-2 (Babovich and Maratea 2003), combines the attractive features of ASSAT and CMODELS-1.

CMODELS-2's algorithm is:

1. Let T be Comp(Π)
2. Find a model M of T. If there is not such model, then terminate with failure
3. If M is an answer set, add to T the negation of the propositional model and go back to step 2
4. If M is not an answer set, consider M as failure, then find a loop L such that its loop formula ϕ_L is not satisfied by M , find one reason from ϕ_L, backjump in the search tree with it, and go back to step 2

Implementing the communication with the SAT solver

function CMODELS-2.Solve(Π)

  f = Comp(Π)

  while ((assignment = SATSOLVER.Solve(f)) != NULL)

    if ((reason = CMODELS-2.hasCycles(assignment)) == NULL)

      return SAT

    SATSOLVER.BacktrackWithReason(reason)

  return UNSAT

How can reason be computed from loop formula?

The reason must be only one clause (in CNF), unsatisfied by the current propositional model. Assume the loop formula (LF) is not satisfied by the current model of completion. Let LF be:

$$\neg(\bigvee_i G_i) \supset \bigwedge_{P \in L} \neg p$$

LF implies the formula:

$$\bigvee_i G_i \vee \neg p$$

where $\bigvee_i G_i \equiv G_1 \vee \ldots \vee G_k, p \in L$. Let $g_1, \ldots, g_k$ be one of the literals in $G_1, \ldots, G_k$ respectively, such that none of these literals is satisfied by the current propositional model. The reason we pick in the clause:

$$g_1 \vee \ldots \vee g_k \vee \neg p$$

Experiments with CMODELS-2 (1)

Instance name	CMODELS SIMO (-bj)	CMODELS SIMO (-le)	SMODELS	CMODELS SIMO assat alg.
np40c	(42) 1.59	(42) 1.56	2.49	(27) 16.52
np60c	(115) 9.35	(106) 8.80	21.45	(35) 76.12
np70c	(172) 20.75	(217) 26.46	42.86	(41) 139.50
np80c	(278) 43.92	(223) 37.87	79.78	(44) 241.55
np100c	(406) 103.22	(286) 78.93	200.43	(51) 561.94
np120c	—	(698) 314.93	430.98	mem
np150c	—	(1074) 841.91	1171.38	mem

Figura 1: Complete graphs CMODELS employing backjumping (and learning) vs. SMODELS vs. CMODELS employing assat algorithm.

Experiments with CMODELS-2 **(2)**

Instance name	CMODELS SIMO (-bj)	CMODELS SIMO (-le)	SMODELS	CMODELS SIMO assat alg.
2xp30	(0) 0.01	(0) 0.01	0.01	(0) 0.01
2xp30.1	timeout	timeout	0.12	(90) 57.58
2xp30.2	timeout	(155) 3092.13	timeout	(152) 24.12
2xp30.4	timeout	timeout	timeout	timeout
4xp20	(0) 0.01	(0) 0.01	0.01	(0) 0.01
4xp20.1	(23) 61.88	(2) 73.26	timeout	(1) 2.03
4xp20.3	(43) 89.37	(13) 82.90	0.01	(5) 1.56

Figura 2: Hand-coded graphs CMODELS employing backjumping (and learning) vs. SMODELS vs. CMODELS employing assat algorithm.

Experiments with CMODELS-2 (3)

Instance Name	CMODELS MCHAFF	CMODELS SIMO (-bj)	CMODELS SIMO (-le)	SMODELS
dp_6.formula1-i-O2-b8	(28) 6.17	(6) 0.45	(6) 0.43	0.46
dp_8.formula1-s-O2-b8	(15) 5.14	(29) 1.02	(14) 0.94	1.83
dp_8.formula1-i-O2-b10	(24) 28.72	(55) 7.22	(41) 6.61	5.08
dp_10.formula1-s-O2-b9	(24) 19.47	(89) 4.17	(36) 3.51	29.05
dp_10.formula1-i-O2-b12	(21) 51.36	(719) 73.09	(162) 14.34	428.85
dp_12.formula1-s-O2-b10	(69) 96.95	(100) 8.49	(93) 7.56	949.95
dp_12.formula1-i-O2-b14	(29) 469.83	(24) 242.56	(14) 81.80	timeout

Figura 3: Nontight Bounded Model Checking CMODELS using MCHAFF employing ASSAT-like algorithm and SIMO employing backjumping (and learning) vs. SMODELS

Conclusions

- a SAT-based approach seems to be a very competitive approach to find answer sets of logic programs
- CMODELS competes with specialized answer set solvers as SMODELS, and outperforms SMODELS on some domains
- backward SAT techniques seems to help in particular on problems arising from real world applications

Future work

- broad experimental analysis, in order to understand if the approach can be competitive on more problem domains and in finding more than one answer set
- evaluate other techniques (ex: find a “better” reason, customize SAT heuristic)
- evaluate if the approach can be used also to solve disjunctive logic programs

References (1)

- P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. Theory and Practice of Logic Programming, to appear.
- J. Lloyd and R. Topor. Making Prolog more expressive. Journal of Logic Programming, 1984, 3:225-240.
- ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proc. AAAI-02*, 2002.

References (2)

- J. Lee and V. Lifschitz. Loop formulas for disjunctive logic programs. In *Proc. ICLP-03*, To appear.
- Yu. Babovich and V. Lifschitz. Computing answer sets using program completion. Unpublished draft, 2003.
- Yu. Babovich and M. Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. Submitted to LPNMR-7.