SAT-Baseb Answer Set Programming

Enrico Giunchiglia DIST, Univ. Genova

Yuliya Lierler Univ. Erlangen-Nürnberg

Marco Maratea DIST, Univ. Genova

Motivation

1. Propositional satisfiability (SAT) is one of the most studied fields in AI and CS
2. Very efficient and specialized SAT procedures exist

$\Rightarrow$ use SAT solvers for computing answer sets (AS)

$\Rightarrow$ ASSAT, Lin and Zhao [02], the first SAT-Based solver ...

$\Rightarrow$ ...not a new idea, but a new approach.

Compilation methods: tight programs

1. Marek and Subrahmanian [89] showed that the answer sets of a program Π correspond to a subset of the models of the Clark's completion of Π
2. Fages [94] proved that if a program Π is "tight" then the answer sets correspond to the models of the completion of Π.
3. This result was extended in various ways by Lifschitz [96]; Erdem, Lierler and Lifschitz [00]; Erdem and Lifschitz[03]

$\surd$ *Comp*(Π) does not introduce any new variables, and its size is at most double wrt the size of Π

$\surd$ Answer Set Programming (ASP) by computing models of *Comp*(Π) is viable and effective if the program is tight, ...

$\times$ ...but not in general

Compilation methods: Introducing variables

1. Starting from Ben-Eliyahu and Dechter [96] various translations from ASP to SAT were provided, each introducing new variables and clauses.
   Their encoding may need $O(|Atoms|^2)$ new variables and $O(|Atoms|^3)$ new clauses
2. Janhunen [03] introduced an optimized subquadratic encoding
3. Lin and Zhao [03] provided a translation requiring $O(|Atoms|^2 + |Rules|)$ new variables and $O(|Atoms| \times |Rules|)$ new clauses

$\surd$ Polynomial encodings

$\times$ still the number of variables or the size of the resulting SAT formula may become impractical

$\times$ a recent result by Lifschitz and Razborov [04] shows that introducing new variables is (likely to be) necessary to get a polynomial encoding

Compilation methods: incremental methods

1. Marek and Subrahmanian [89] showed that the answer sets of a program Π correspond to a subset of the models of the Clarke's completion of Π

2. Lin and Zhao [02] proposed the extension to program's completion such that models of completion satisfying the extension are in one-to-one correspondence with programs answer sets. This extension consists of loop formulas.

$\surd$ if the program is tight – no formulas are added

$\surd$ the loop formulas can be added incrementally on demand

$\times$ unfortunately, exponentially many loop formulas may be needed

$\times$ does not seem viable for computing all answer sets (in general)

$\times$ their solver, ASSAT, does not allow optimzed statements as ...

Question:

Is it possible to build an efficient *SAT-Based* answer set generator that

1. deals with any (non disjunctive) logic program,
2. works on a SAT formula without additional variables, and
3. is guaranteed to work in polynomial space?

Outline of the talk

1. Basic preliminaries
2. From SAT solvers to AS solvers (I)
3. From SAT solvers to AS solvers (II)
4. Implementation and experimental analysis
5. Conclusions

Basic preliminaries

A *(logic) program* Π is a finite set of rules of the form:

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n \quad (1)$$

where A_i is either $\bot$ (*False*) or an atom in P. A_0 is the *head*.

Comp(Π) consists of formulas of the type

$$A_0 \equiv \bigvee (A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n)$$

for each symbol in $P \cup \{\bot\}$. In the equation, the disjunction extends over all rules (1) in Π with head A_0.

From SAT to AS solvers (I): DPLL

SAT(φ) **return** DPLL(CNF(φ), $\emptyset$);

DPLL(Γ, S)

if $\Gamma = \emptyset$ **then return** *True*;

if $\emptyset \in \Gamma$ **then return** *False*;

if $\{l\} \in \Gamma$ **then return** DPLL($assign(l, \Gamma), S \cup \{l\}$);

$A :=$ an atom occurring in Γ;

return DPLL($assign(A, \Gamma), S \cup \{A\}$) **or**

DPLL($assign(\neg A, \Gamma), S \cup \{\neg A\}$).

From SAT to AS solvers (I): DPLL-based AS solvers

ASP-SAT(Π) **return** DPLL$($CNF$(\underline{\textit{Comp}(\Pi)}), \emptyset)$;

DPLL(Γ, S)

if $\Gamma = \emptyset$ **then return** $\underline{\textit{test}(S, \Pi)}$;

if $\emptyset \in \Gamma$ **then return** *False*;

if $\{l\} \in \Gamma$ **then return** DPLL$(assign(l, \Gamma), S \cup \{l\})$;

$A :=$ an atom occurring in Γ;

return DPLL$(assign(A, \Gamma), S \cup \{A\})$ **or**

DPLL$(assign(\neg A, \Gamma), S \cup \{\neg A\})$.

test(S, Π) returns *True* if $S \cap P$ is an answer set of Π, and *False*, otherwise.

From SAT to AS solvers (I): Discussion

1. ASP-SAT(Π) returns *True* iff Π has an answer set
2. ASP-SAT(Π) can be easily modified in order to compute all answer sets of a program Π
3. Most SOTAC SAT solvers are a (non-recursive) implementation of DLL
4. Most SOTAC SAT solvers are based on "learning" in order to backjump irrelevant nodes while backtracking and avoid the exploration of useless parts of the search tree

From SAT to AS solvers (II): Computing reasons

1. Learning procedures require *test*(S, Π) to return a $S' \subseteq S$ such that for no S'' entailing *Comp*(Π) and with $S' \subseteq S''$, $S'' \cap P$ is ensured not to be an AS of Π

2. One such set is S, but it is important that S be as small as possible:

$\Rightarrow$ one possibility it to return $S \cap P$, or (better)

$\Rightarrow$ we can compute the subset of S which falsifies one of the loop-formulas in Π

From SAT to AS solvers (II): Example

Assume Π is

$$A_i \leftarrow A_{i+1} \qquad A_{i+1} \leftarrow A_i \qquad i \in \{0, 2, \ldots, 2k\}$$

Then $Comp(\Pi)$ includes

$$A_i \equiv A_{i+1} \qquad (i \in \{0, 2, \ldots, 2k\})$$

- ASP-SAT without learning or with learning in which *test*(S, Π) computes $S \cap P$ as reason, may generate 2^k assignments entailing $Comp(\Pi)$.
- ASP-SAT with learning in which *test*(S, Π) computes as reason the subset of S falsifying one of the loop formulas, may generate at most k assignments entailing $Comp(\Pi)$.
- ASP-SAT modified in order to assign the atoms in P to *False* while branching, generates the only answer set of Π.

Implementation

1. ASP-SAT has been implemented on top of the learning SAT solver SIMO and integrated in CMODELS.
2. The implementation and integration posed some non trivial problems described in details in the paper.
3. The resulting system is called CMODELS2.

Experimental results: Blocks world

		Standard programs			Extended programs	
#b	#s	SMODELS	ASSAT	CMODELS2	SMODELS	CMODELS2
8	i-1	12.32	0.80	1.19	0.81	0.47
11	i-1	71.78	2.97	4.19	2.97	1.01
8	i	40.87	0.89	2.18	1.56	1.40
11	i	71.42	3.17	4.52	3.41	1.16
8	i+1	23.35	0.96	0.97	4.99	0.31
11	i+1	107.48	3.54	3.33	5.21	0.75

Table 1: Blocks world: “#b” is the number of blocks.

Experimental results: H.C. complete graphs

	Standard programs				Extended programs	
	SMODELS	ASSAT	DLV	CMODELS2	SMODELS	CMODELS2
np30c	11.70	1.14	22.08	0.69	0.36	0.36
np40c	62.89	41.81	97.96	1.63	2.48	0.87
np50c	219.56	14.51	314.46	3.37	8.39	1.79
np60c	594.46	48.80	770.07	5.81	20.47	3.41
np70c	1323.61	291.60	1679.12	8.22	39.41	5.87
np80c	2354.28	32.51	3407.35	14.20	75.36	9.18
np90c	TIME	779.06	TIME	22.23	122.53	14.19
np100c	TIME	—	TIME	28.63	185.52	20.76
np120c	TIME	—	TIME	53.33	418.15	41.84

Table 2: Complete graphs. npXc corresponds to a graph with "X" nodes.

Experimental results: FV problems

	SMODELS	ASSAT	DLV	CMODELS2
mutex4	33.92	(0)0.62	840.60	(0)0.68
phi4	0.24	(168)2.98	1.44	TIME
mutex2	0.09	(88)1.78		(0)0.12
mutex3	229.57	MEM		(0)24.16
phi3	2.87	(704)236.91		(57)3.91

Table 3: Checking requirements in a deterministic automaton.

Experimental results: BMC problems

BMC	SMODELS	CMODELS2	CMODELS2'
dp-10.i-02-b11	382.72	1476.72	442.14
dp-10.s-02-b8	15.24	8.20	14.22
dp-12.s-O2-b9	336.03	65.41	137.34
dp-8.i-O2-b9	8.08	12.62	10.69
dp-8.s-O2-b7	1.19	1.02	2.28
dp-10.i-O2-b12	445.47	3295.72	163.29
dp-10.s-O2-b9	28.87	16.07	15.03
dp-12.s-O2-b10	971.50	209.29	48.73
dp-8.i-O2-b10	5.05	40.01	6.44
dp-8.s-O2-b8	1.76	1.99	2.03

Table 4: Bounded Model Checking Problems.

Computing all solutions: Summing up

- Overall SMODELS and DLV perform better than CMODELS2 when all solutions are computed.
- CMODELS2 is competitive (and even better) whenever number of loops in the program is small.
- More work is needed, in particular on the design of new SAT heuristics.

Conclusions

We have presented a SAT-Based polynomial space algorithm for Answer Set Programming that

1. it is relatively easy to implement of SAT solvers with/without learning,

2. it is easy to extend to compute all the AS,

3. shows good results when computing one AS

Ongoing work on Cmodels

1. Extend the SAT-based approach to disjuntive logic programs (viable approach?)
2. Working on logic programs structure to enhance SAT search (one of the Mirek's proposed challenge at NMR'04)