On the Relation Between ASP and SAT procedures

Enrico Giunchiglia DIST, Univ. Genova

Marco Maratea DIST, Univ. Genova

Motivation

- The relation between Answer Set Programming (ASP) and propositional satisfiability (SAT) has been at the center of several papers, expecially in the last years.
- Despite state-of-the art ASP solvers are apparently quite different,
- the main search procedures used by ASP solvers (i.e., "native" and SAT-based) have been advocated "similar" in many works. But this has never been formally stated before.

Goal

In this work we study the computational properties of ASP systems, in order to formally characterize under which condictions different systems have same behaviour.

We begin our study with SMODELS and CMODELS, and then we see how the results extend to other systems like DLV, SMODELS-CC and ASSAT.

The main focus of this work is on *tight* programs (Fages 1994; Babovich, Erdem and Lifschitz 2000; Erdem and Lifschitz 2003), where we will establish a strong relation between SMODELS and CMODELS procedures.

We will use the result both on the theoretical side (in order to show new complexity results for SMODELS) and on the experimental side (for evaluating efficient strategies and heuristics coming from SAT, in ASP systems).

Basic preliminaries

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

$$p_0 \leftarrow p_1, \ldots, p_m, \overline{p}_{m+1}, \ldots, \overline{p}_n \tag{1}$$

where $p_0 \in P \cup \{\bot\}$, and $\{p_1, \ldots p_n\} \subseteq P$.

head$(r) = p_0$, *body*$(r) = \{p_1, \ldots, p_m, \overline{p}_{m+1}, \ldots, \overline{p}_n\}$.

If p is a an atom, $\overline{p}$ is the *negation* of p.

Atoms, their negations, and the symbols $\bot$, $\top$ form the set of *literals*.

A *clause* is a finite set of literals different from $\bot, \top$.

A *propositional formula* is a finite set of clauses.

We say that a program Π is *tight* if there exists a function λ from atoms to ordinals such that, for every rule (1) in Π whose head is not $\bot$, $\lambda(p_0) > \lambda(p_i)$ for each i = 1 . . . m.

What is an answer set?

Consider a program Π, and let X be a set of atoms. In order to give the definition of an answer set we consider first the special case in which the body of each rule in Π contains only atoms. Under these assumptions, we say that

- X is *closed* under Π if for every rule (1) in Π, $p_0 \in X$ whenever $\{p_1, \ldots, p_m\} \subseteq X$, and that
- X is an *answer set* for Π if X is the smallest set closed under Π.

Now we consider the case in which Π is an arbitrary program. The *reduct* Π^X of Π relative to X is the set of rules

$$p_0 \leftarrow p_1, \ldots, p_m$$

for all rules (1) in Π such that $X \cap \{p_{m+1}, \ldots, p_n\} = \emptyset$. X is an *answer set* for Π if X is an answer set for Π^X.

Agenda

- SMODELS and CMODELS algorithms
- Relation between SMODELS and CMODELS
- New complexity results for SMODELS
- Experimental analysis on CMODELS
- Conclusions

SMODELS **procedure (I)**

function SMODELS(Π) **return** SMODELS-REC(Π, $\{\top\}$);
function SMODELS-REC(Π,S)
  $\langle \Pi, S \rangle$:= *expand*(Π, S);
  if ($\{l, \overline{l}\} \subseteq S$) **return** FALSE;
  if ($\{p : p \in P, \{p, \overline{p}\} \cap S \neq \emptyset\} = P$) **return** TRUE;
  p := *ChooseAtom*(S);
  return SMODELS-REC(*p-assign*(p, Π)), $S \cup \{p\}$) **or**
    SMODELS-REC(*p-assign*$(\overline{p}, \Pi)$), $S \cup \{\overline{p}\}$);
function *expand*(Π,S)
  S' := S;
  S := *AtLeast*(Π, S);
  Π := *p-assign*(S, Π);
  S := $S \cup \{\overline{p} : p \in P, p \notin$ *AtMost*$(\Pi^{\emptyset}, S)\}$;
  Π := *p-assign*(S, Π);
  if ($S \neq S'$) **return** *expand*(Π,S);
  return $\langle \Pi, S \rangle$;

SMODELS **procedure (II)**

function *AtLeast*(Π,S)

if ($r \in \Pi$ **and** *body*$(r) = \emptyset$ **and** *head*$(r) \notin S$)

return *AtLeast*(*p-assign*(*head*$(r), \Pi), S \cup \{$*head*$(r)\})$;

if ($\{p, \overline{p}\} \cap S = \emptyset$ **and** $\nexists r \in \Pi :$ *head*$(r) = p$)

return *AtLeast*(*p-assign*$(\overline{p}, \Pi), S \cup \{\overline{p}\})$;

if ($r \in \Pi$ **and** *head*$(r) \in S$ **and** *body*$(r) \neq \emptyset$ **and**

$\nexists r' \in \Pi, r' \neq r :$ *head*$(r') =$ *head*(r))

return *AtLeast*(*p-assign*(*body*$(r), \Pi), S \cup$ *body*$(r))$;

if ($r \in \Pi$ **and** $\overline{head(r)} \in S$ **and** *body*$(r) = \{l\}$)

return *AtLeast*(*p-assign*$(\bar{l}, \Pi)), S \cup \{\bar{l}\})$;

return S;

function *AtMost*(Π,S)

if ($r \in \Pi$ **and** *body*$(r) = \emptyset$ **and** *head*$(r) \notin S$)

return *AtMost*(*p-assign*(*head*$(r), \Pi), S \cup \{$*head*$(r)\})$;

return S;

CMODELS **procedure**

function CMODELS(Π) **return** DLL-REC(*lp2sat*(Π),$\emptyset$,Π);

function DLL-REC(Γ,S,Π)

$\langle\Gamma, S\rangle$:= *unit-propagate*(Γ, S);

if ($\emptyset \in \Gamma$) **return** FALSE;

if ($\Gamma = \emptyset$) **return** *test*(S,Π);

p := *ChooseAtom*(S);

return DLL-REC(*s-assign*$(p, \Gamma)), S \cup \{p\}, \Pi)$ **or**

DLL-REC(*s-assign*$(\overline{p}, \Gamma)), S \cup \{\overline{p}\}, \Pi)$;

function *unit-propagate*(Γ,S)

if ($\{l\} \in \Gamma$) **return** *unit-propagate*(*s-assign*$(l, \Gamma), S \cup \{l\})$;

return $\langle\Gamma, S\rangle$;

From a logic program to a set of clauses

We have defined *lp2sat*(Π) to be the set of clauses corresponding to the Clark's completion (Clark 1978) of Π. More precisely, if p_0 is an atom, the *translation of* Π *relative to* p_0, denoted with *lp2sat*(Π, p_0), consists of

1. for each rule $r \in \Pi$ of the form (1) and whose head is p_0, the clauses:

$$\{p_0, \overline{n}_r\}, \{n_r, \overline{p}_1, \ldots, \overline{p}_m, p_{m+1}, \ldots, p_n\},$$
$$\{\overline{n}_r, p_1\}, \ldots, \{\overline{n}_r, p_m\}, \{\overline{n}_r, \overline{p}_{m+1}\}, \ldots, \{\overline{n}_r, \overline{p}_n\},$$

   where n_r is a newly introduced atom, and

2. the clause $\{\overline{p}_0, n_{r_1}, \ldots, n_{r_q}\}$ where $n_{r_1}, \ldots, n_{r_q}$ $(q \geq 0)$ are the new symbols introduced in the previous step.

The *translation of* Π *relative to* $\perp$, denoted with *lp2sat*$(\Pi, \perp)$, consists of a clause $\{\overline{p}_1, \ldots, \overline{p}_m, p_{m+1}, \ldots, p_n\}$, one for each rule in Π of the form (1) with head $\perp$. Finally, the *translation of* Π, denoted with *lp2sat*(Π), is $\cup_{p \in P \cup \{\perp\}}$ *lp2sat*(Π, p).

From a set of clauses to a logic program

If C is a clause $\{l_1, \ldots, l_l\}$ $(l \geq 0)$ we define *sat2tlp*(C) to be the rule

$$\bot \leftarrow \bar{l}_1, \ldots, \bar{l}_l.$$

Then, if Γ is a formula, the *translation of* Γ, denoted with *sat2tlp*(Γ), is

$$\cup_{C \in \Gamma} \textit{sat2tlp}(C) \cup \cup_{p \in P} \{p \leftarrow \overline{p'}, p' \leftarrow \overline{p}\}$$

where, for each atom $p \in P$, p' is a new atom associated to p.

Relating SMODELS and CMODELS

Our goal is to prove that the computations of SMODELS and CMODELS are highly related if Π is tight. We establish this comparing the search trees of SMODELS-REC$(\Pi, \{\top\})$ and DLL-REC(*lp2sat*$(\Pi), \emptyset, \Pi)$.

We say that a set of literals S is a *branching node of* SMODELS(Π) if there is a call to SMODELS-REC(Π', S), following the invocation of SMODELS(Π). Similar considerations are made for CMODELS.

If $proc$ is SMODELS(Π) or CMODELS(Π), we define

$$\mathit{Br}(proc) = \{S \cap (P \cup \overline{P}) : S \text{ is a branching node of } proc\}.$$

We say that SMODELS(Π) and CMODELS(Π) are *equivalent* if *Br*(SMODELS(Π)) $=$ *Br*(CMODELS(Π)).

Theorem 1 *For each tight program* Π, SMODELS(Π) *and* CMODELS(Π) *are equivalent.*

New results for SMODELS: Pigeonhole principle

The *complexity of a procedure* $proc$ *on a program* Π is the smallest N such that $|\mathit{Br}(proc)| = N$.

Consider the formula PHP_n^m where n, m are two natural numbers, and consisting of the clauses

$$\{p_{i,1}, p_{i,2}, \ldots, p_{i,n}\} \qquad (i \leq m),$$

$$\{\overline{p}_{i,k}, \overline{p}_{j,k}\} \qquad (i, j \leq m, k \leq n, i \neq j).$$

The formulas PHP_n^m are from (Haken 1985) and encode the pigeonhole principle. If $n < m$, PHP_n^m are unsatisfiable and it is well known that any procedure based on resolution (like DLL) has an exponential behavior on these formulas.

Corollary 1 *The complexity of* SMODELS *and* CMODELS *on sat2tlp*(PHP_{n-1}^n) *is exponential in* n.

The result extends to CMODELS because it is based on DLL. For SMODELS, it relies on the fact that *sat2tlp*(PHP_{n-1}^n) is tight, and thus SMODELS and CMODELS are equivalent.

New results for SMODELS: Randomly generated k-cnf formulas

A formula Γ is a k-*cnf* if each clause in Γ consists of k literals.

The *random family of k-cnf formulas* is a k-cnf whose clauses have been randomly selected with uniform distribution among all the clauses C of k literals and such that, for each two distinct literals l and l' in C, $\bar{l} \neq l'$.

Corollary 2 *Consider a random k-cnf formula Γ with n atoms and m clauses. With probability tending to one as n tends to infinity, the complexity of* SMODELS *and* CMODELS *on sat2tlp(Γ) is exponential in n if the density $d = m/n \geq 0.7 \times 2^k$.*

This result follows from (Chvátal and Szemerédi 1988), and again from the fact that *sat2tlp*(Γ) is tight on the random family, from the fact that CMODELS is based on DLL and our equivalence result on tight programs.

New results for SMODELS: Deciding the best literal

We define a literal l to be *optimal for a program* Π if there exists a minimal search tree of SMODELS(Π) whose root is labeled with l. The following result echoes the one by (Liberatore 2000) for DLL.

Corollary 3 *In* SMODELS*, deciding the optimal literal to branch on is both NP-hard and co-NP hard, and in PSPACE for tight programs.*

There are many other results holding for DLL that can be lifted to SMODELS, including (Monasson 2004) and (Achlioptas et al. 2001) for average complexity of coloring randomly generated graphs and for exponential lower bounds on random 3-CNF formulas also below the satisfiability threshold.

SMODELS **and** CMODELS **are not equivalent on non-tight programs**

Consider again the pigeonhole formulas. They give us the opportunity to define a class of formulas that are exponentially hard for CMODELS but easy for SMODELS.

For each formula Γ, defines *sat2nlp*(Γ) to be the program $\cup_{C\in\Gamma}$*sat2tlp*$(C) \cup \cup_{p\in P}\{p \leftarrow p\}$.

Corollary 4 *The complexity of* SMODELS *and* CMODELS *on sat2nlp*(PHP^n_{n-1}) *is* 0 *and exponential in* n *respectively.*

In this case, *sat2nlp*(PHP^n_{n-1}) is non-tight, and SMODELS can determine the non existence of answer sets without branching mainly thanks to the procedure *AtMost*.

The above results can be easily generalized to any formula Γ which is known to be exponentially hard for DLL.

Extending the results to other systems

ASSAT is different from CMODELS only on non-tight programs, assuming that Γ is computed as *lp2sat*(Π).

SMODELS-CC is SMODELS enhanced with "clause-learning" look-back strategies.

Results in (Haken 1985) and (Chvátal and Szemerédi 1988) hold for any proof systems based on resolution. Enhancing SMODELS and CMODELS with "learning" look-back strategies does not lower the exponential complexity.

Thus, the related corollaries hold also for SMODELS-CC and ASSAT.

DLV core algorithm is similar to the one of SMODELS. In particular, the rules used by *AtLeast* to extend the assignment S are very similar to those used by the DLV procedure $DetCons$, see (Faber 2002), pagg. 41-44.

Experimental analysis: Assessment (I)

Given the above results, one expects that the combinations of reasoning strategies that currently dominate in SAT, are also bound to dominate in ASP, at least on tight logic programs.

We show experimentally, on a wide set of currently challenges benchmarks, that this is the case, and results extend (on the experimental side) to non-tight programs.

We have used our solver, CMODELS ver. 2, because it is SAT-based and thus strengths the relation between SAT and ASP, and also

- its front-end is LPARSE (Simons 2000), a widely used grounder for logic programs;
- its back-end solver already incorporates (lazy) data structures for fast unit propagation as well as some state-of-the-art strategies and heuristics evaluated in this work; and
- can be also run on non-tight programs.

Experimental analysis: Assessment (II)

We have further extended CMODELS with a variety a look-ahead, look-back strategies and heuristics coming from the SAT community. We have considered

- Look-ahead: basic unit-propagation (u), unit-propagation+failed-literal (f) (Freeman 1995)
- Look-back: basic backtracking (b), backtracking+backjumping+learning (l) (Sakallah and Silva 1996; Bayardo and Schrag 1997; Zhang et. al 2001)
- Heuristic: VSIDS (v) (Moskewicz et al. 2001), Unit-based (u and p) (Li and Anbulagan 1997)

We considered 5 combination of strategies built out of them: ulv, flv, flu, fbu and ulp.

Performing the experiment on a unique platform is of fundamental importance, otherwise results can be biased by implementation issues.

Given the established "equivalence", results would extend to SMODELS (and to the other systems, according to the considerations made) if enhanced with corresponding techniques (at least on tight programs).

Experimental analysis: Tight logic programs

PB	#VAR	ulv	flv	flu	fbu	ulp
4.5	300	TIME	TIME	81.92	**22.53**	TIME
5	300	448.21	485.36	<u>8.27</u>	**4.72**	452.75
5.5	300	73	38.61	<u>2.26</u>	**1.7**	38.48
dp-12.fsa-i-b9	1186	**223.93**	<u>383.66</u>	<u>353.53</u>	TIME	2910.96
key-2-i-b29	3199	415.54	204.87	**44.14**	589.45	1329.53
mmgt-3.fsa-i-b10	1933	16.23	32.23	26.71	16.55	**6.19**
mmgt-4.fsa-s-b8	1586	<u>17.02</u>	27.59	421.30	327.55	**13.79**
q-1.fsa-i-b17	2201	1539.96	<u>505.15</u>	**259.05**	816.26	TIME
queens21	925	786.14	1864.49	384.87	47.33	**0.24**
queens24	1201	TIME	TIME	TIME	368.76	**0.28**
queens50	5101	TIME	TIME	TIME	TIME	**347.98**
bw-large.d9	9956	<u>1.02</u>	5.84	2.69	2.75	**1.01**
bw-large.e9	12260	**0.98**	<u>1.91</u>	<u>1.92</u>	<u>1.93</u>	<u>1.03</u>
bw-large.e10	13482	**1.29**	7.51	5.03	4.95	<u>1.55</u>
p1000	14955	**0.48**	37.86	15.41	15.23	3.69
p3000	44961	**8.86**	369.27	144.12	142.83	223.62
p6000	89951	**99.50**	TIME	583.55	578.98	2549.50

Experimental analysis: Non-tight logic programs

PB	#VAR	ulv	flv	flu	fbu	ulp
4	300	265.43	218.48	41.97	**31.05**	77.41
5	300	TIME	TIME	136.67	**99.75**	439.71
6	300	TIME	TIME	107.34	**65.83**	591.3
bw-basic-P4-i	5301	**2.16**	15.54	6.07	5.79	2.54
bw-basic-P4-i-1	4760	**1.64**	4.92	2.47	2.44	1.86
bw-basic-P4-i+1	5842	2.49	24.27	22.01	19.71	**2.41**
np60c	10742	**2.83**	1611.32	44.12	44.12	4.77
np70c	14632	**4.69**	TIME	97.44	97.89	5.91
np80c	19122	**6.91**	TIME	192.29	196.32	12.88

Conclusions (I)

- We have shown that, on the wide class of tight logic programs, SMODELS and CMODELS are "equivalent", i.e., they explore search trees with the same branching nodes
- Thanks to this "equivalence" results, we have outlined new complexity results on SMODELS procedure
- We have then extended the results to other systems
- Motivated by the "equivalence" result, we have extended CMODELS with a variety of strategies and heuristics and run an extensive experimental analysis
- The experimental results suggest directions for future development of efficient ASP solvers.

Conclusions (II)

- Our work should be particularly important for AS researchers how are interested in
  - formally establishing the computational behavior of systems
  - develop efficient AS solvers, and thus focusing either on random or real-world problems
  - benchmarking systems (the ASPARAGUS initiative)
- we believe that this work is a major step in the direction of closing the gap between SAT and ASP, as advocated by Mirek Truszczyński in his invited talk at the last NMR workshop in Whistler, Canada