Propositional satisfiability (SAT), SAT-based ASP and relation between ASP and SAT procedures

Marco Maratea

j.w.w. Enrico Giunchiglia

Department of Communication, Computer and System Sciences
DIST, University of Genova, STAR-Lab

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 deciding more expressive logics and formalisms . . .

$\Rightarrow$ . . . reusing most of the work and knowledge available in SAT

SAT: The problem

A *literal* l is a proposition p or its negation $\neg p$.

Given the literals $l_1, \ldots, l_k$, a *clause* is $l_1 \vee \cdots \vee l_k$.

Given the clauses $c_1, \ldots, c_m$, a Conjunctive Normal Form (CNF) formula is $c_1 \wedge \cdots \wedge c_m$.

An *assignment*, or *valuation* v, is a partial function from the propositions to $\{\text{TRUE,FALSE}\}$.

We can extend the definition of v in the natural way to assign truth values to literals, clauses and formulas.

Given a CNF formula Γ, we define the *propositional satisfiability problem (SAT)*:

Does there exist an assignment v to the propositions in Γ such that Γ is true?

SAT: Examples

1. $\varphi := \{p, p \vee \neg q, \neg r\}$ has the satisfying assignments
   - $\{p :=$ TRUE, $q :=$ TRUE, $r :=$ FALSE$\}$
   - $\{p :=$ TRUE, $q :=$ FALSE, $r :=$ FALSE$\}$
2. $\varphi := \{\neg p, p \vee \neg q, r \vee \neg p, q\}$ has no satisfying assignments because the clause $\{p \vee \neg q\}$ can not be satisfied.

SAT: Solving methods

- Resolution algorithm
- Local search algorithms
- (Ordered) Binary Decision Diagrams (OBDDs) (Bryant 1992)
- Stalmark's method (1989)
- Davis-Logemann-Loveland (DLL) algorithm

Agenda

- DLL algorithm
- CMODELS2: DLL-based (SAT-based) decision procedure for ASP
- Experiments with CMODELS2
- Relation between ASP and SAT procedures
- Conclusions

DLL algorithm

function DLL-REC(Γ,S)

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

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

if ($\Gamma = \emptyset$) **return** TRUE;

A := *ChooseAtom*(S);

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

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

function *unit-propagate*(Γ,S)

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

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

Introduction to ASP

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

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

Let P be the set of atoms in Π, $A_0 \in P \cup \{\bot\}$, $\{A_1, \ldots, A_n\} \subseteq P$. A_0 is the *head*.

Comp(Π) (Clark 1978) 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.

CMODELS2: DLL-based decision procedure for ASP

function CMODELS2(Π) **return** DLL-REC(*lp2sat*(Π),$\emptyset$);

function DLL-REC(Γ,S)

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

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

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

A := *ChooseAtom*(S);

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

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

function *unit-propagate*(Γ,S)

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

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

CMODELS2 employs the SAT solvers SIMO, a ZCHAFF-like solver. *test*(S, Π) returns TRUE if $S \cap P$ is an answer set of Π, and FALSE, otherwise.

Extension to non-basic rules

CMODELS2 can work with other types of rules other than the basic ones showed before, namely:

- choice rules, $\{A_0, \ldots, A_k\} \leftarrow A_{k+1}, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$
- cardinality and weight constraint rules

  $A_0 \leftarrow L\{A_1 = w_1, \ldots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n\}U$

All these rules, together with the basic, can be translated into *basic nested* rules

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_k, not\ not\ A_{k+1}, \ldots, not\ not\ A_n.$$

A choice rule $\{A\} \leftarrow$. is translated in $A \leftarrow not\ not\ A$, while weight constraint are translated using the method presented in (Ferraris and Lifschitz, TPLP 2005).

For a basic nested program Π, *Comp*(Π) is defined as well.

CMODELS2: **Discussion**

1. CMODELS2(Π) returns TRUE iff Π has an answer set

2. CMODELS2 works in polynomial space

3. CMODELS2(Π) can be modified in order to compute all the answer sets of a program Π

4. *test*(S, Π) can fail because of "loops" in Π

5. Most state-of-the-art SAT solvers are a (non-recursive) implementation of DLL

6. Most state-of-the-art 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

CMODELS2: **Computing reasons**

If SAT solvers are based on learning

1. Learning procedures require *test*(S, Π) to return a $S' \subseteq S$ such that for each 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

CMODELS2: **Advantages**

With respect to ASSAT, CMODELS2 has a number of advantages, other than points 2. and 3. in the discussion slide

- it works with basic and non-basic rules
- no computation is ever repeated
- it does not introduce extra variables (except the ones needed by the clause form transformation)

With respect to SMODELS and DLV, CMODELS2 has the advantage of being SAT-based, and thus it can leverage on the great amount of work done in SAT

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: Formal Verification problems

	SMODELS	ASSAT	DLV	CMODELS2
mutex4	33.92	0.62	840.60	0.68
phi4	0.24	2.98	1.44	TIME
mutex2	0.09	1.78		0.12
mutex3	229.57	MEM		24.16
phi3	2.87	236.91		3.91

Table 3: Checking requirements in a deterministic automaton. (Heljanko and Stefanescu 2003)

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. (Heljanko and Niemela 2003)

On the relation between AS and SAT procedures: Motivation

- The relation between Answer Set Programming (ASP) and propositional satisfiability (SAT) has been at the center of several papers, especially 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.

On the relation between AS and SAT procedures: Goal

We study the computational properties of ASP systems, in order to formally characterize under which conditions different systems have same behavior.

We begin our study with SMODELS and CMODELS2, 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) using basic rules, where we will establish a strong relation between SMODELS and CMODELS2 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).

SMODELS **procedure (I)**

function SMODELS(Π) **return** SMODELS-REC(Π, $\{\top\}$);
function SMODELS-REC(Π,S)
    $\langle \Pi, S \rangle$:= *expand*(Π, S);
    if ($\{l, not\ l\} \subseteq S$) **return** FALSE;
    if ($\{A : A \in P, \{A, not\ A\} \cap S \neq \emptyset\} = P$) **return** TRUE;
    A := *ChooseAtom*(S);
    return SMODELS-REC$($*p-assign*$(A, \Pi)), S \cup \{A\})$ **or**
            SMODELS-REC$($*p-assign*$(not\ A, \Pi)), S \cup \{not\ A\})$;
function *expand*(Π,S)
    S' := S;
    S := *AtLeast*(Π, S);
    Π := *p-assign*(S, Π);
    $S := S \cup \{not\ A : A \in P, A \notin$ *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 ($\{A, not\ A\} \cap S = \emptyset$ **and** $\nexists r \in \Pi$: *head*$(r) = A$)
    return *AtLeast*(*p-assign*$(not\ A, \Pi), S \cup \{not\ A\}$);
  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** not *head*$(r) \in S$ **and** *body*$(r) = \{l\}$)
    return *AtLeast*(*p-assign*$(not\, l, \Pi)), S \cup \{not\, l\}$);
  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;

From a logic program to a set of clauses

We have defined *lp2sat*(Π) to be the set of clauses corresponding to *Comp*(Π). More precisely, if A_0 is an atom, the *translation of* Π *relative to* A_0, denoted with *lp2sat*(Π, A_0), consists of

1. for each rule $r \in \Pi$ of the form (1) and whose head is A_0, the clauses:

$$\{A_0, \overline{n}_r\}, \{n_r, \overline{A}_1, \ldots, \overline{A}_m, A_{m+1}, \ldots, A_n\},$$
$$\{\overline{n}_r, A_1\}, \ldots, \{\overline{n}_r, A_m\}, \{\overline{n}_r, \overline{A}_{m+1}\}, \ldots, \{\overline{n}_r, \overline{A}_n\},$$

   where n_r is a newly introduced atom, and

2. the clause $\{\overline{A}_0, n_{r_1}, \ldots, n_{r_q}\}$ 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* $\bot$, denoted with *lp2sat*$(\Pi, \bot)$, consists of a clause $\{\overline{A}_1, \ldots, \overline{A}_m, A_{m+1}, \ldots, A_n\}$, one for each rule in Π of the form (1) with head $\bot$. Finally, the *translation of* Π, denoted with *lp2sat*(Π), is $\cup_{p \in P \cup \{\bot\}}$*lp2sat*$(\Pi, p)$.

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

$$\bot \leftarrow not\ l_1, \ldots, not\ l_l.$$

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

$$\cup_{C \in \Gamma} \textit{sat2tlp}(C) \cup \cup_{p \in P} \{p \leftarrow not\ p', p' \leftarrow not\ p\}$$

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

Relating SMODELS and CMODELS2

Our goal is to prove that the computations of SMODELS and CMODELS2 are highly related if Π is tight. We establish this comparing the search trees of SMODELS-REC$(\Pi, \{\top\})$ and DLL-REC(*lp2sat*$(\Pi), \emptyset)$.

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 CMODELS2.

If $proc$ is SMODELS(Π) or CMODELS2(Π), we define

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

We say that SMODELS(Π) and CMODELS2(Π) are *equivalent* if $Br(\text{SMODELS}(\Pi)) = Br(\text{CMODELS2}(\Pi))$.

Theorem 1 *For each tight program* Π*,* SMODELS(Π) *and* CMODELS2(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 $|Br(proc)| = N$.

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

$$\{A_{i,1}, A_{i,2}, \ldots, A_{i,n}\} \qquad (i \leq m),$$
$$\{\overline{A}_{i,k}, \overline{A}_{j,k}\} \qquad (i, j \leq m, k \leq n, i \neq j).$$

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* CMODELS2 *on sat2tlp*(PHP_{n-1}^n) *is exponential in* n*.*

The result extends to CMODELS2 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 CMODELS2 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* CMODELS2 *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 CMODELS2 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** CMODELS2 **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 CMODELS2 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* CMODELS2 *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 CMODELS2 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 CMODELS2 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. We are working on the comparison between DLV algorithm and DLL-REC.

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 to certain degrees, and results extend (on the experimental side) to non-tight programs.

We have used our solver, CMODELS2, 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 CMODELS2 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 focus on 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	8.27	**4.72**	452.75
5.5	300	73	38.61	2.26	**1.7**	38.48
dp-12.fsa-i-b9	1186	**223.93**	383.66	353.53	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	17.02	27.59	421.30	327.55	**13.79**
q-1.fsa-i-b17	2201	1539.96	505.15	**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	1.02	5.84	2.69	2.75	**1.01**
bw-large.e9	12260	**0.98**	1.91	1.92	1.93	1.03
bw-large.e10	13482	**1.29**	7.51	5.03	4.95	1.55
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

Main results

- the SAT-based approach used by CMODELS2 is competitive w.r.t. rival systems, at least on non-disjunctive case and when looking for one answer set
- ASP and SAT procedures have been demonstrated to be “equivalent” on tight programs; this lead to establish new, previously unknown results for SMODELS that can be extended to ASSAT and SMODELS-CC with the extents we have seen. Extending the results to DLV is work in progress
- a deep experimental investigation, motivated by the previous theoretical result, has shown how SAT techniques can be beneficial for ASP solvers, and has shed light on future directions for develop ASP systems

SAT-based: Other applications

The SAT-based approach has been used (in our group) to develop decision procedure for

- Separation Logic, a decidable quantifier-free fragment of the first order logic involving propositional logic and linear arithmetic, with applications in FV and scheduling (TSAT++)
- optimization problem related to SAT (namely Max-SAT, Min-ONE) with application in model checking and planning (OPTSAT)

and

- QSAT, or QBF, (QuBE++)
- disjunctive logic programming (Cmodels3)
- conformant planning (CPlan)

References (I): ASP

E. Giunchiglia and M. Maratea - Evaluating Search Stategies and Heuristics for Efficient Answer Set Programming. Accepted to 9th Congress of the Italian Association for Artificial Intelligence (AI*IA 2005). To appear LNAI.

E. Giunchiglia and M. Maratea - An Experimenal Study on Search Stategies and Heuristics in Answer Set Programming. Accepted to ASP05.

E. Giunchiglia, Yu. Lierler and M. Maratea - A SAT-based Polynomial Space Algorithm for Answer Set Programming. In Proc. 10th International Workshop on Non Monotonic Reasoning (NMR 2004).

E. Giunchiglia, Yu. Lierler and M. Maratea - SAT-based Answer Set Programming. In Proc. 19th American Association for Artificial Intelligence (AAAI 2004).

Yu. Lierler and M. Maratea - Cmodels2: SAT-based Answer Set Solvers Extended to Non-tight Programs. In Proc. 7th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2004).

References (II): Others

A. Armando, C. Castellini, E. Giunchiglia and M. Maratea - The SAT-based Approach to Separation Logic Accepted to the Journal of Automated Reasoning (JAR). 2005.

A. Armando, C. Castellini, E. Giunchiglia and M. Maratea - A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. Accepted to 7th International Conference on Theory and Practice of Satisfiability Testing (SAT 2004). To appear LNCS.

A. Armando, C. Castellini, E. Giunchiglia, M. Idini and M. Maratea - TSAT++: An Open Reasoning Platform for Satisfiability Modulo Theory. Accepted to 2th Workshop on Pragmatic of Decision Procedures in Automated Reasoning (PDPAR 2004). To appear ENTCS.

E. Giunchiglia, M. Maratea and A. Tacchella - (In)Effectiveness of Look-ahead Techniques in a Modern SAT solver. In Proc. 9th Int. Conference on Principle and Practice of Constraint Programming (CP 2003)

E. Giunchiglia, M. Maratea and A. Tacchella - Dependent and Independent Variables in Propositional Satisfiability. In Proc. 8th European Conference on Logics in Artificial Intelligence (JELIA 2002)

E. Giunchiglia, M. Maratea, A. Tacchella and D.Zambonin - Evaluating Search Heuristics and Optimization Techniques in Propositional Satisfiability. In Proc. 1st International Joint Conference on Automated Reasoning (IJCAR 2001)