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

Marco Maratea
(j.w.w. Enrico Giunchiglia)

Dipartimento di Matematica
Università della Calabria

DEIS, UNICAL, 14/06/2006
Motivation

1. Propositional satisfiability (SAT) is one of the most studied fields in AI and CS

2. Very efficient and specialized SAT procedures exist
   ⇒ use SAT solvers for deciding more expressive logics and formalisms . . .
   ⇒ . . . reusing most of the work and knowledge available in SAT
A literal $l$ is a proposition (variable/atom) $p$ or its negation $\neg p$. Given the literals $l_1, \ldots, l_k$, a clause is $l_1 \lor \cdots \lor l_k$. Given the clauses $c_1, \ldots, c_m$, a Conjunctive Normal Form (CNF) formula is $c_1 \land \cdots \land c_m$.

An assignment, or valuation $\nu$, is a partial function from the propositions to \{TRUE,FALSE\}. We can extend the definition of $\nu$ in the natural way to assign truth values to literals, clauses and formulas.

Given a CNF formula $\Gamma$, we define the propositional satisfiability problem (SAT):

Does there exist an assignment $\nu$ to the propositions in $\Gamma$ such that $\Gamma$ is true?
1. \( \varphi := \{p, p \lor \neg q, \neg r\} \) has the satisfying assignments
   - \( \{p := \text{TRUE}, \ q := \text{TRUE}, \ r := \text{FALSE}\} \)
   - \( \{p := \text{TRUE}, \ q := \text{FALSE}, \ r := \text{FALSE}\} \)

2. \( \varphi := \{\neg p, p \lor \neg q, r \lor \neg p, q\} \) has no satisfying assignments because the clause \( \{p \lor \neg q\} \) can not be satisfied.
SAT: Solving methods

- Resolution algorithm
- Local search algorithms
- (Ordered) Binary Decision Diagrams (OBDDs) (Bryant 1992)
- Davis-Logemann-Loveland (DLL) algorithm
Agenda

- DLL algorithm
- SAT/DLL-based approaches to ASP
- Experiments with ASP solvers
- Relation between ASP and SAT procedures
- Further experiments

Marco Maratea

SAT, SAT-based ASP and their relation.
function DLL-REC(Γ,S)
    ⟨Γ, S⟩ := unit-propagate(Γ, S);
    if (∅ ∈ Γ) return FALSE;
    if (Γ = ∅) return TRUE;
    A := ChooseAtom(S);
    return DLL-REC(s-assign(A, Γ)), S ∪ {A}) or
          DLL-REC(s-assign(¬A, Γ)), S ∪ {¬A});

function unit-propagate(Γ,S)
    if ({l} ∈ Γ) return unit-propagate(s-assign(l, Γ), S ∪ {l});
    return ⟨Γ, S⟩;

s-assign(l, Γ) deletes all the clauses containing l, and all the occurrences of ¬l.
A (logic) program $\Pi$ is a finite set of rules of the form

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

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

$\text{Comp}(\Pi)$ (Clark 1978) consists of formulas of the type

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

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

For the class of tight logic programs (Fages 1994; Erdem and Lifschitz 2003), there is 1-1 correspondence between the ASP solutions and the models of $\text{Comp}(\Pi)$. 
CMODELS algorithm (focus on finding one answer set)

1. Computes $\Gamma = \text{Comp}(\Pi)$. (and converts it into a set of clauses)
2. Checks if $\Gamma$ is tight.
3. If it is, finds a model $X$ of $\Gamma$ by a SAT solver. If such a model exists returns TRUE, otherwise FALSE.
ASSAT algorithm (Lin and Zhao 2002, 2004)

1. Computes $\Gamma = \text{Comp}(\Pi)$. (and converts it into a set of clauses)

2. Finds a model $X$ of $\Gamma$ by a SAT solver. If no such a model exists, return false.

3. Checks if $X$ is an answer set: If $X$ is an answer set, then returns true. Otherwise
   a) finds (at least) one “loop formula” which is not satisfied by $X$, and adds it to $\Gamma$; and
   b) goes back to step 2.

The foundation of the algorithm is in the following theorem:

**Theorem**

Let $\text{LF}(\Pi)$ be the set of loop formulas associated with the loops of $\Pi$. $X$ is an answer set iff is a model of $\text{Comp}(\Pi) \cup \text{LF}(\Pi)$. 
ASSAT disadvantages

- It is not guaranteed to work in polynomial space.
- Some computation can be repeated. (several times)
- It introduces new variables, other than the ones needed by the clause-form transformation

Besides these weaknesses, **ASSAT** showed to be very competitive wrt state-of-the-art systems like **SMODELS** and **DLV**.

But (much) better could be done not considering the SAT solver as a “black-box”.

Marco Maratea SAT, SAT-based ASP and their relation.
\textbf{Cmodels2:} DLL-based decision procedure for ASP

\begin{verbatim}
function \texttt{Cmodels2}(\Pi) return \texttt{dll-rec}(lp2sat(\Pi),\emptyset,\Pi);

function \texttt{dll-rec}(\Gamma,S,\Pi)
\langle \Gamma, S \rangle := \texttt{unit-propagate}(\Gamma, S);
if (\emptyset \in \Gamma) return \texttt{FALSE};
if (\Gamma = \emptyset) return \texttt{test}(S,\Pi);
A := \texttt{ChooseAtom}(S);
return \texttt{dll-rec}(s-assign(A, \Gamma)), S \cup \{A\} or
\texttt{dll-rec}(s-assign(\overline{A}, \Gamma)), S \cup \{\overline{A}\});

function \texttt{unit-propagate}(\Gamma,S)
\texttt{if} (\{l\} \in \Gamma) return \texttt{unit-propagate}(s-assign(l, \Gamma), S \cup \{l\});
return \langle \Gamma, S \rangle;
\end{verbatim}

\texttt{Cmodels2} employs the SAT solvers \texttt{simo}, a \texttt{zchaff}-like solver. \texttt{test}(S,\Pi) returns \texttt{TRUE} if $S \cap P$ is an answer set of $\Pi$, and \texttt{FALSE}, otherwise.
Cmodels2 can work with other types of rules other than the basic ones showed before, namely:

- choice rules,
  \[ \{A_0, \ldots, A_k\} \leftarrow A_{k+1}, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n \]

- cardinality and weight constraint rules
  \[ A_0 \leftarrow L\{A_1 = w_1, \ldots, A_m = w_m, \text{not } A_{m+1} = w_{m+1}, \ldots, \text{not } A_n = w_n\} U \]

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

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

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

For a basic nested program \( \Pi \), \( \text{Comp}(\Pi) \) 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
If SAT solvers are based on learning

1. Learning procedures require \( \text{test}(S, \Pi) \) to return a \( S' \subseteq S \) such that for each \( S'' \) entailing \( \text{Comp}(\Pi) \) and with \( S' \subseteq S'' \), \( S'' \cap P \) is ensured not to be an AS of \( \Pi \)

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 a subset of \( S \) which falsifies one of the loop formulas in \( \Pi \)
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: Tight programs

<table>
<thead>
<tr>
<th>Program</th>
<th>SMODELS</th>
<th>SMODELS-CC</th>
<th>ASSAT</th>
<th>DLV</th>
<th>Cmodels2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4c1000</td>
<td>22.28</td>
<td>4.95</td>
<td>0.6</td>
<td></td>
<td>0.48</td>
</tr>
<tr>
<td>4c3000</td>
<td>202.84</td>
<td>1143.13</td>
<td>2.19</td>
<td>8.86</td>
<td></td>
</tr>
<tr>
<td>4c6000</td>
<td>856.13</td>
<td>TIME</td>
<td>14.85</td>
<td>99.50</td>
<td></td>
</tr>
<tr>
<td>schur.4-43</td>
<td>0.43</td>
<td>0.95</td>
<td>0.67</td>
<td>590.57</td>
<td>1.4</td>
</tr>
<tr>
<td>schur.4-44</td>
<td>0.44</td>
<td>91.25</td>
<td>1.07</td>
<td>TIME</td>
<td>5.97</td>
</tr>
<tr>
<td>schur.4-45</td>
<td>571.17</td>
<td>1110.68</td>
<td>434.93</td>
<td>TIME</td>
<td>229.04</td>
</tr>
<tr>
<td>15puz.18</td>
<td>17.55</td>
<td>6.94</td>
<td>1.06</td>
<td>141.68</td>
<td>0.98</td>
</tr>
<tr>
<td>15puz.19</td>
<td>20.94</td>
<td>7.14</td>
<td>3.61</td>
<td>208.41</td>
<td>1.35</td>
</tr>
<tr>
<td>15puz.20</td>
<td>70.27</td>
<td>8.22</td>
<td>4.59</td>
<td>TIME</td>
<td>1.28</td>
</tr>
<tr>
<td>pigeon.9.10</td>
<td>44.77</td>
<td>65.91</td>
<td>1.1</td>
<td></td>
<td>1.26</td>
</tr>
<tr>
<td>pigeon.10.11</td>
<td>484.63</td>
<td>1029.38</td>
<td>23.83</td>
<td>12.41</td>
<td></td>
</tr>
<tr>
<td>pigeon.51.50</td>
<td>106.79</td>
<td>24.29</td>
<td>2.49</td>
<td></td>
<td>1.63</td>
</tr>
</tbody>
</table>

Table: 4c* = 4 coloring; schur* = schur numbers; 15puz* = puzzle; and pigeon* = pigeons programs.
### Experimental results: Blocks world

<table>
<thead>
<tr>
<th>#b</th>
<th>#s</th>
<th>Standard programs</th>
<th></th>
<th>Extended programs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SM</td>
<td>SMCC</td>
<td>ASSAT</td>
<td>Cm2</td>
</tr>
<tr>
<td>8</td>
<td>i-1</td>
<td>7.48</td>
<td>7.17</td>
<td>0.86</td>
<td><strong>0.49</strong></td>
</tr>
<tr>
<td>11</td>
<td>i-1</td>
<td>36.18</td>
<td>35.53</td>
<td><strong>3.15</strong></td>
<td><strong>1.64</strong></td>
</tr>
<tr>
<td>8</td>
<td>i</td>
<td>17.35</td>
<td>9.3</td>
<td>0.98</td>
<td><strong>0.63</strong></td>
</tr>
<tr>
<td>11</td>
<td>i</td>
<td>37.71</td>
<td>43.9</td>
<td><strong>3.59</strong></td>
<td><strong>2.16</strong></td>
</tr>
<tr>
<td>8</td>
<td>i+1</td>
<td>12.08</td>
<td>15.17</td>
<td><strong>1.09</strong></td>
<td><strong>1.34</strong></td>
</tr>
<tr>
<td>11</td>
<td>i+1</td>
<td>54.3</td>
<td>62.39</td>
<td><strong>3.9</strong></td>
<td><strong>2.49</strong></td>
</tr>
</tbody>
</table>

**Table:** Blocks world: “#b” is the number of blocks. “#s” is the number of steps.
# Experimental results: H.C. complete graphs

<table>
<thead>
<tr>
<th></th>
<th>Standard programs</th>
<th></th>
<th></th>
<th></th>
<th>Extended programs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SM</td>
<td>SMCC</td>
<td>ASSAT</td>
<td>DLV</td>
<td>Cm2</td>
<td>SM</td>
<td>SMCC</td>
</tr>
<tr>
<td>np30c</td>
<td>4.08</td>
<td>2.18</td>
<td>1.61</td>
<td>9.55</td>
<td>0.35</td>
<td>0.26</td>
<td>0.68</td>
</tr>
<tr>
<td>np40c</td>
<td>22.05</td>
<td>6.57</td>
<td>70.97</td>
<td>44.7</td>
<td>0.85</td>
<td>0.75</td>
<td>1.16</td>
</tr>
<tr>
<td>np50c</td>
<td>84.49</td>
<td>15.3</td>
<td>24.17</td>
<td>142.55</td>
<td>1.66</td>
<td>2.3</td>
<td>1.95</td>
</tr>
<tr>
<td>np60c</td>
<td>242.61</td>
<td>30.81</td>
<td>84.87</td>
<td>361.8</td>
<td>2.83</td>
<td>7.05</td>
<td>3.82</td>
</tr>
<tr>
<td>np70c</td>
<td>557.08</td>
<td>55.31</td>
<td>520.8</td>
<td>798.96</td>
<td>4.69</td>
<td>15.67</td>
<td>5.92</td>
</tr>
<tr>
<td>np80c</td>
<td>1001.88</td>
<td>90.59</td>
<td>53.25</td>
<td>1587.6</td>
<td>7.2</td>
<td>32.29</td>
<td>9.01</td>
</tr>
<tr>
<td>np90c</td>
<td>2064.61</td>
<td>144.72</td>
<td>1416.24</td>
<td>2807.84</td>
<td>10.42</td>
<td>53.21</td>
<td>14.13</td>
</tr>
<tr>
<td>np100c</td>
<td>3573.19</td>
<td>215.37</td>
<td>TIME</td>
<td>TIME</td>
<td>14.23</td>
<td>83.11</td>
<td>14.95</td>
</tr>
</tbody>
</table>

**Table:** Complete graphs. npXc corresponds to a graph with “X” nodes.
### Experimental results: Formal Verification problems

<table>
<thead>
<tr>
<th></th>
<th>SMODELS</th>
<th>SMODELS-CC</th>
<th>ASSAT</th>
<th>DLV</th>
<th>CMODELS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex4</td>
<td>14.14</td>
<td>5.35</td>
<td>0.54</td>
<td>367.89</td>
<td>0.46</td>
</tr>
<tr>
<td>phi4</td>
<td>0.18</td>
<td>0.55</td>
<td>1.96</td>
<td>0.83</td>
<td>179.71</td>
</tr>
<tr>
<td>mutex2</td>
<td>0.28</td>
<td>0.3</td>
<td>2.6</td>
<td></td>
<td>0.15</td>
</tr>
<tr>
<td>mutex3</td>
<td>163.94</td>
<td>110.27</td>
<td>MEM</td>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td>phi3</td>
<td>3.23</td>
<td>3.04</td>
<td>53.28</td>
<td></td>
<td>1.43</td>
</tr>
</tbody>
</table>

**Table:** Checking requirements in a deterministic automaton. (Heljanko and Stefanescu 2003)
### Experimental results: BMC problems

<table>
<thead>
<tr>
<th>BMC</th>
<th>SMODELS</th>
<th>SMODELS-CC</th>
<th>CMODELS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp-10.i-O2-b12</td>
<td>132.72</td>
<td>2.25</td>
<td>488.76</td>
</tr>
<tr>
<td>dp-10.s-O2-b9</td>
<td>9.75</td>
<td>3.11</td>
<td>6.38</td>
</tr>
<tr>
<td>dp-12.s-O2-b10</td>
<td>296.45</td>
<td>1.1</td>
<td>53.2</td>
</tr>
<tr>
<td>dp-8.i-O2-b10</td>
<td>1.76</td>
<td>2.42</td>
<td>12.28</td>
</tr>
<tr>
<td>dp-8.s-O2-b8</td>
<td>0.73</td>
<td>0.14</td>
<td>0.47</td>
</tr>
</tbody>
</table>

**Table:** Bounded Model Checking Problems. (Heljanko and Niemela 2003)
Enhancements to **Cmodels**

From the point of view of search

1. Introduce new SAT techniques (see next part of the talk!)
2. Design specialized heuristic (see next slide!)
3. Integrate a new SAT solver
4. Loop (formulas) are “too generous” (elementary loops/sets)

From the point of view of expressivity, **Cmodels3**

1. Extension of **Cmodels2** that allows for (non-nested) disjunctive rules, choice and weight constraints rules
2. *test()* is a co-NP check: It uses (another) SAT solver for it
3. Interesting preliminary results, more has to be done
### Extended programs

<table>
<thead>
<tr>
<th></th>
<th>SMODELS</th>
<th>SMODELS-CC</th>
<th>CMODELS2</th>
<th>CMODELS2 hf</th>
</tr>
</thead>
<tbody>
<tr>
<td>np30c</td>
<td>0.26</td>
<td>0.68</td>
<td>0.17</td>
<td>0.16</td>
</tr>
<tr>
<td>np40c</td>
<td>0.75</td>
<td>1.16</td>
<td>0.66</td>
<td>0.45</td>
</tr>
<tr>
<td>np50c</td>
<td>2.3</td>
<td>1.95</td>
<td>2.21</td>
<td>0.91</td>
</tr>
<tr>
<td>np60c</td>
<td>7.05</td>
<td>3.82</td>
<td>3.55</td>
<td>1.74</td>
</tr>
<tr>
<td>np70c</td>
<td>15.67</td>
<td>5.92</td>
<td>10.54</td>
<td>2.96</td>
</tr>
<tr>
<td>np80c</td>
<td>32.29</td>
<td>9.01</td>
<td>15.05</td>
<td>4.73</td>
</tr>
<tr>
<td>np90c</td>
<td>53.21</td>
<td>14.13</td>
<td>32.19</td>
<td>7.61</td>
</tr>
<tr>
<td>np100c</td>
<td>83.11</td>
<td>14.95</td>
<td>34.18</td>
<td>10.79</td>
</tr>
</tbody>
</table>

### Extended programs

<table>
<thead>
<tr>
<th></th>
<th>SMODELS</th>
<th>SMODELS-CC</th>
<th>ASSAT</th>
<th>DLV</th>
<th>CMODELS2</th>
<th>Cm2 hf</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex4</td>
<td>14.14</td>
<td>5.35</td>
<td>0.54</td>
<td>367.89</td>
<td>0.46</td>
<td>0.34</td>
</tr>
<tr>
<td>phi4</td>
<td>0.18</td>
<td>0.55</td>
<td>1.96</td>
<td>0.83</td>
<td>179.71</td>
<td>TIME</td>
</tr>
<tr>
<td>mutex2</td>
<td>0.28</td>
<td>0.3</td>
<td>2.6</td>
<td>0.15</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>mutex3</td>
<td>163.94</td>
<td>110.27</td>
<td>MEM</td>
<td>TIME</td>
<td>11.82</td>
<td></td>
</tr>
<tr>
<td>phi3</td>
<td>3.23</td>
<td>3.04</td>
<td>53.28</td>
<td>1.43</td>
<td>1.94</td>
<td></td>
</tr>
</tbody>
</table>
The SAT/DLL-based approach has been used (in our group) to develop decision procedures 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++)

**Example**

\[ \text{START} \land ((e_i - s_i \leq 10) \lor (s_j - e_i \leq 0)) \]

- optimization problem related to SAT (namely Max-SAT, Min-ONE) with main application in planning (OPTSAT)

and

- QSAT, or QBF, (QuBE++)
- conformant planning (CPlan)
On the relation between ASP and SAT procedures: Motivation

The relation between ASP and SAT has been at the center of several papers, especially in the last years.

This is confirmed by the upcoming new (ICLP-)workshop Lash06: Search and Logic: Answer Set Programming and SAT.

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.
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 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).
function SMODELS($\Pi$) return SMODELS-REC($\Pi$, \{\top\});

function SMODELS-REC($\Pi$, $S$)
    $\langle \Pi, S \rangle := $ expand($\Pi$, $S$);
    if (\{l, not l\} \subseteq S) return FALSE;
    if (\{A : A \in P, \{A, not A\} \cap S \neq \emptyset\} = P) return TRUE;
    $A := \text{ChooseAtom}(S)$;
    return SMODELS-REC($p$-elim($A$, $\Pi$)), $S \cup \{A\}$ or
    SMODELS-REC($p$-elim(not $A$, $\Pi$)), $S \cup \{not A\}$;

function expand($\Pi$, $S$)
    $S' := S$;
    $S := \text{AtLeast}(\Pi, S)$;
    $\Pi := p$-elim($S$, $\Pi$);
    $S := S \cup \{not A : A \in P, A \notin \text{AtMost}(\Pi^0, S)\}$;
    $\Pi := p$-elim($S$, $\Pi$);
    if ($S \neq S'$) return expand($\Pi$, $S$);
    return $\langle \Pi, S \rangle$;
function AtLeast(\Pi, S) 
    if (r \in \Pi \text{ and body}(r) = \emptyset \text{ and head}(r) \not\in S)
        return AtLeast(p-elim(head(r), \Pi), S \cup \{head(r)\});
    if (\{A, not A\} \cap S = \emptyset \text{ and } \not\exists r \in \Pi : head(r) = A)
        return AtLeast(p-elim(not A, \Pi), S \cup \{not A\});
    if (r \in \Pi \text{ and head}(r) \in S \text{ and body}(r) \neq \emptyset \text{ and }
        \not\exists r' \in \Pi, r' \neq r : head(r') = head(r))
        return AtLeast(p-elim(body(r), \Pi), S \cup body(r));
    if (r \in \Pi \text{ and not head}(r) \in S \text{ and body}(r) = \{l\})
        return AtLeast(p-elim(not l, \Pi)), S \cup \{not l\});
    return S;

function AtMost(\Pi, S) 
    if (r \in \Pi \text{ and body}(r) = \emptyset \text{ and head}(r) \not\in S)
        return AtMost(p-elim(head(r), \Pi), S \cup \{head(r)\});
    return S;
We have defined $lp2sat(\Pi)$ to be the set of clauses corresponding to $Comp(\Pi)$. More precisely, if $A_0$ is an atom, the *translation of $\Pi$ relative to $A_0$*, denoted with $lp2sat(\Pi, A_0)$, consists of

1. for each rule $r \in \Pi$ of the form (1) and whose head is $A_0$, the clauses:
   
   $\{A_0, \bar{n}_r\}, \{n_r, \bar{A}_1, \ldots, \bar{A}_m, A_{m+1}, \ldots, A_n\}$,
   $\{\bar{n}_r, A_1\}, \ldots, \{\bar{n}_r, A_m\}, \{\bar{n}_r, \bar{A}_{m+1}\}, \ldots, \{\bar{n}_r, \bar{A}_n\}$,

   where $n_r$ is a newly introduced atom, and

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

The *translation of $\Pi$ relative to $\bot$*, denoted with $lp2sat(\Pi, \bot)$, consists of a clause $\{\bar{A}_1, \ldots, \bar{A}_m, A_{m+1}, \ldots, A_n\}$, one for each rule in $\Pi$ of the form (1) with head $\bot$. Finally, the *translation of $\Pi$*, denoted with $lp2sat(\Pi)$, is $\bigcup_{p \in P \cup \{\bot\}} lp2sat(\Pi, p)$. 
If $C$ is a clause $\{l_1, \ldots, l_l\}$ ($l \geq 0$) we define $\text{sat2tlp}(C)$ to be the rule

$$\bot \leftarrow \text{not } l_1, \ldots, \text{not } l_l.$$ 

Then, if $\Gamma$ is a formula, the \textit{translation of} $\Gamma$, denoted with $\text{sat2tlp}(\Gamma)$, is

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

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

Marco Maratea

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

We say that a set of literals \( S \) is a branching node of SMODELS(\( \Pi \)) if there is a call to SMODELS-REC(\( \Pi' \), \( S \)), following the invocation of SMODELS(\( \Pi \)). Similar considerations are made for CMODELS2. If proc is SMODELS(\( \Pi \)) or CMODELS2(\( \Pi \)), we define

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

We say that SMODELS(\( \Pi \)) and CMODELS2(\( \Pi \)) are equivalent if 

\[
Br(SMODELS(\Pi)) = Br(CMODELS2(\Pi)).
\]

**Theorem**

*For each tight program \( \Pi \), SMODELS(\( \Pi \)) and CMODELS2(\( \Pi \)) are equivalent.*
New results for **SMODELS**: Pigeonhole principle

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

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

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

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

**Corollary**

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-REC. 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 \textsc{smodels}: Randomly generated $k$-CNF formulas

A formula $\Gamma$ is a $k$-CNF if each clause in $\Gamma$ consists of $k$ literals. The \textit{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 $C$, $\overline{l} \neq l'$.

\textbf{Corollary}

Consider a random $k$-CNF formula $\Gamma$ with $n$ atoms and $m$ clauses. With probability tending to one as $n$ tends to infinity, the complexity of \textsc{smodels} and \textsc{Cmodels} on $\text{sat2tlp}(\Gamma)$ 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 $\text{sat2tlp}(\Gamma)$ is tight on the random family, from the fact that $\textsc{Cmodels}$ is based on \textsc{dll-rec} 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* $\Pi$ if there exists a minimal search tree of $\text{SMODELS}(\Pi)$ whose root is labeled with $l$. The following result echoes the one by (Liberatore 2000) for \text{DLL-REC}.

**Corollary**

In \text{SMODELS}, deciding the optimal literal to branch on is both \text{NP-hard} and \text{co-NP hard}, and in \text{PSPACE} for tight programs.

There are many other results holding for \text{DLL-REC} that can be lifted to \text{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 $\Gamma$, defines $\text{sat2nlp}(\Gamma)$ to be the program $\bigcup_{C \in \Gamma} \text{sat2tlp}(C) \cup \bigcup_{p \in P} \{p \leftarrow p\}$.

**Corollary**

*The complexity of **SMODELS** and **CMODELS2** on $\text{sat2nlp}(\text{PHP}_{n-1}^n)$ is 0 and exponential in $n$ respectively.*

In this case, $\text{sat2nlp}(\text{PHP}_{n-1}^n)$ 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 $\Gamma$ which is known to be exponentially hard for **DLL-REC**.
ASSAT is different from CMODELS2 only on non-tight programs, assuming that $\Gamma$ is computed as $lp2sat(\Pi)$.

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 (Enrico, Nicola and I) are working on the comparison between DLV algorithm, dll-rec (thus CMODELS) and SMODELS.
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, \textit{Cmodels2}, because it is SAT-based and thus strengthens the relation between SAT and ASP, and also

\begin{itemize}
  \item its front-end is \texttt{lparsE} (Simons 2000), a widely used grounder for logic programs;
  \item 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
  \item can be also run on non-tight programs.
\end{itemize}
Experimental analysis: Assessment (II)

We have further extended CMODELS2 with a variety of look-ahead, look-back strategies and heuristics coming from the SAT community.

- **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) (Li and Anbulagan 1997)

We focus on 4 combinations of strategies built out of them: ulv, flv, flu and fbu.

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: Small, random programs**

<table>
<thead>
<tr>
<th>PB</th>
<th># VAR</th>
<th>ulv</th>
<th>flv</th>
<th>flu</th>
<th>fbu</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>300</td>
<td>0.59</td>
<td>0.8</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>4.5</td>
<td>300</td>
<td>TIME</td>
<td>TIME</td>
<td>115.29</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>300</td>
<td>456.22</td>
<td>538.89</td>
<td>17.64</td>
</tr>
<tr>
<td>4</td>
<td>5.5</td>
<td>300</td>
<td>72.83</td>
<td>53.26</td>
<td>4.42</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>300</td>
<td>24.73</td>
<td>21.89</td>
<td>1.83</td>
</tr>
</tbody>
</table>

| 6  | 4     | 300 | 265.43 | 218.48 | 41.97 | 31.05|
| 7  | 4.5   | 300 | TIME | TIME | 190.73 | 135.11|
| 8  | 5     | 300 | TIME | TIME | 136.67 | 99.75|
| 9  | 5.5   | 300 | TIME | TIME | 129.29 | 78.63|
| 10 | 6     | 300 | TIME | TIME | 107.34 | 65.83|

**Table:** Performances on randomly generated logic programs. Problems (1)-(5) are tight programs being the translation of 3-SAT benchmarks. Problems (6)-(10) are randomly generated logic programs using Lin and Zhao’s methodology.
## Experimental analysis: Large programs

<table>
<thead>
<tr>
<th>PB</th>
<th># VAR</th>
<th>ulv</th>
<th>flv</th>
<th>flu</th>
<th>fbu</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>bw*d9</td>
<td>9956+</td>
<td>1.02</td>
<td>5.84</td>
<td>2.69</td>
</tr>
<tr>
<td>12</td>
<td>bw*e9</td>
<td>12260</td>
<td>0.98</td>
<td>1.91</td>
<td>1.92</td>
</tr>
<tr>
<td>13</td>
<td>bw*e10</td>
<td>13482+</td>
<td>1.29</td>
<td>7.51</td>
<td>5.03</td>
</tr>
<tr>
<td>14</td>
<td>4c1000</td>
<td>14955+</td>
<td>0.48</td>
<td>37.86</td>
<td>15.41</td>
</tr>
<tr>
<td>15</td>
<td>4c3000</td>
<td>44961+</td>
<td>8.86</td>
<td>369.27</td>
<td>144.12</td>
</tr>
<tr>
<td>16</td>
<td>4c6000</td>
<td>89951+</td>
<td>99.50</td>
<td>TIME</td>
<td>583.55</td>
</tr>
<tr>
<td>17</td>
<td>np80c</td>
<td>19122+</td>
<td>7.2</td>
<td>TIME</td>
<td>195.08</td>
</tr>
<tr>
<td>18</td>
<td>np90c</td>
<td>24212+</td>
<td>10.42</td>
<td>TIME</td>
<td>364.54</td>
</tr>
<tr>
<td>19</td>
<td>np100c</td>
<td>29902+</td>
<td>14.23</td>
<td>TIME</td>
<td>610.2</td>
</tr>
<tr>
<td>20</td>
<td>np80c</td>
<td>19043+</td>
<td>15.05</td>
<td>1538.86</td>
<td>23.63</td>
</tr>
<tr>
<td>21</td>
<td>np90c</td>
<td>24123+</td>
<td>32.19</td>
<td>2918.82</td>
<td>38.75</td>
</tr>
<tr>
<td>22</td>
<td>np100c</td>
<td>29803+</td>
<td>34.18</td>
<td>TIME</td>
<td>59.15</td>
</tr>
<tr>
<td>23</td>
<td>mutex4</td>
<td>14698+</td>
<td>0.46</td>
<td>28.29</td>
<td>28.3</td>
</tr>
<tr>
<td>24</td>
<td>phi3</td>
<td>16930+</td>
<td>1.43</td>
<td>55.62</td>
<td>12.15</td>
</tr>
</tbody>
</table>

**Table:** (11)-(13) are blocks world; (14)-(16) are 4 coloring; (17)-(22) are HC on complete graphs; (23)-(24) are “verification” problems.

Marco Maratea | SAT, SAT-based ASP and their relation.
### Table: (25)-(34) are BMC; (35)-(37) are schur numbers; (38)-(40) are 15 puzzle; (41)-(43) are pigeons problems.

<table>
<thead>
<tr>
<th>PB</th>
<th># VAR</th>
<th>ulv</th>
<th>flv</th>
<th>flu</th>
<th>fbu</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>k<em>i</em>29</td>
<td>3199</td>
<td>415.54</td>
<td>204.87</td>
<td>44.14</td>
</tr>
<tr>
<td>26</td>
<td>k<em>s</em>29</td>
<td>3169</td>
<td>353.69</td>
<td>1028.77</td>
<td>59.99</td>
</tr>
<tr>
<td>27</td>
<td>q<em>j</em>17</td>
<td>2201</td>
<td>1539.96</td>
<td>505.15</td>
<td>259.05</td>
</tr>
<tr>
<td>28</td>
<td>e<em>3</em>i*15</td>
<td>7832+</td>
<td>479.28</td>
<td>TIME</td>
<td>7.15</td>
</tr>
<tr>
<td>29</td>
<td>e<em>4</em>i*13</td>
<td>6447</td>
<td>87.63</td>
<td>567.27</td>
<td>20.02</td>
</tr>
<tr>
<td>30</td>
<td>d<em>10</em>i*12</td>
<td>1488+</td>
<td>488.76</td>
<td>1212.89</td>
<td>152.8</td>
</tr>
<tr>
<td>31</td>
<td>d<em>10</em>s*9</td>
<td>1140+</td>
<td>6.38</td>
<td>19.31</td>
<td>87.64</td>
</tr>
<tr>
<td>32</td>
<td>d<em>12</em>s*10</td>
<td>1511+</td>
<td>53.2</td>
<td>165.9</td>
<td>733.9</td>
</tr>
<tr>
<td>33</td>
<td>d<em>8</em>i*10</td>
<td>1003+</td>
<td>12.28</td>
<td>25.03</td>
<td>1.21</td>
</tr>
<tr>
<td>34</td>
<td>d<em>8</em>s*8</td>
<td>819+</td>
<td>0.47</td>
<td>3.73</td>
<td>2.38</td>
</tr>
<tr>
<td>35</td>
<td>schur.4-43</td>
<td>736+</td>
<td>1.4</td>
<td>2.07</td>
<td>0.82</td>
</tr>
<tr>
<td>36</td>
<td>schur.4-44</td>
<td>753+</td>
<td>5.97</td>
<td>5.62</td>
<td>92.63</td>
</tr>
<tr>
<td>37</td>
<td>schur.4-45</td>
<td>770</td>
<td>229.04</td>
<td>417.34</td>
<td>244.35</td>
</tr>
<tr>
<td>38</td>
<td>15puz.18</td>
<td>5945+</td>
<td>0.98</td>
<td>2.9</td>
<td>9.85</td>
</tr>
<tr>
<td>39</td>
<td>15puz.19</td>
<td>6258+</td>
<td>1.35</td>
<td>2.93</td>
<td>11.65</td>
</tr>
<tr>
<td>40</td>
<td>15puz.20</td>
<td>6571</td>
<td>1.28</td>
<td>10.22</td>
<td>64.54</td>
</tr>
<tr>
<td>41</td>
<td>pige.9.10</td>
<td>210</td>
<td>1.26</td>
<td>4.33</td>
<td>1259.84</td>
</tr>
<tr>
<td>42</td>
<td>pige.10.11</td>
<td>253</td>
<td>12.41</td>
<td>55.46</td>
<td>TIME</td>
</tr>
<tr>
<td>43</td>
<td>pige.51.50</td>
<td>5252+</td>
<td>1.63</td>
<td>221.33</td>
<td>6.85</td>
</tr>
</tbody>
</table>
Possible enhancements

1. Design/Implement “better” look-ahead techniques (like DLV),
   ▶ to “bound” the number of failed-literals

2. Design/Implement heuristic suited for random benchmarks
   ▶ based on “backbones”, but this point “calls” . . .
   ▶ . . . for the research issue of generating random logic programs

3. Refine VSIDS heuristic
   ▶ initialization and variables scoring
Main results

- the SAT-based approach used by *Cmodels*₂ 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.
1. Look-back (VSIDS-like) heuristic for DLV. Preliminary results

2. Extending the relation between smodels/Cmodels to dlv.

(Expected) Results (to be proved . . .)

- dlv, smodels and Cmodels are equivalent on tight programs
- dlv and smodels are equivalent on non-tight programs
- dlv and smodels are not equivalent on non-tight programs if deprived of well-founded/atmost procedures.
Main references (I): ASP

- E. Giunchiglia, M. Maratea - On the relation between Answer Set and SAT procedures (or, between smodels and cmodels). In Proc. 21th International Conference on Logic Programming (ICLP 2005).
Main references (II): Others

- E. Giunchiglia, M. Maratea - Solving Optimization Problems with DLL. Accepted to the 17th European Conference on Artificial Intelligence (ECAI) 2006.
Tight on the completion model

There exist a function $\lambda$ from atoms to nonnegative integers such that, for each rule (1) in $\Pi$

$$\lambda(A_1), \ldots, \lambda(A_m) < \lambda(A_0)$$

Example

$p \leftarrow \text{not } q$.
$q \leftarrow \text{not } p$.
$p \leftarrow p, r$.

The idea is to make function $\lambda$ partial: Instead of tight programs consider programs “tight on a set of literals”.

Marco Maratea

SAT, SAT-based ASP and their relation.
Given a loop $L$, we define $R(L)$ to be the set of formulas

$$A_1 \land \cdots \land A_m \land \overline{A}_{m+1} \land \cdots \land \overline{A}_n$$

for all rules (1) in $\Pi$ with $A_0 \in L$ and $\{A_1, \ldots, A_m\} \cap L = \emptyset$.

The loop formula associated with $L$ is

$$\bigvee_{A \in L} A \supset \bigvee R(L)$$