A Fast SAT-based Answer Set Solver

Zhijun Lin, Yuanlin Zhang and Héctor J. Hernández
Knowledge Representation Lab
Computer Science Department
Texas Tech University

\{lin, yzhang, hector\}@cs.ttu.edu

\(^a\) This is part of Mr. Z. Lin’s dissertation
Outline

- Logic Programs
- Answer Set Semantics
- Clark’s Completion
- Algorithm
- Findings and Experimental Results
- Future Work
A program $P$

reachable($X, Y$) ← edge($X, Y$).
reachable($X, Y$) ← edge($X, U$), reachable($U, Y$).

- **INPUT:** \{ edge($a, b$) \mid (a, b) is an edge \}
- **E.g.:** $D = \{ \text{edge}(1, 2), \text{edge}(2, 3), \text{edge}(3, 4), \text{edge}(4, 1) \}$.
- **What’s the ouput $P(D)$?**
**Semantics: $P(D)$**

reachable($X, Y$) ← edge($X, Y$).
reachable($X, Y$) ← edge($X, U$), reachable($U, Y$).

- All edges $(a, b)$ st $b$ is reachable from $a + D$.

\[
P(D) = \{ \text{reachable}(X, Y) \mid X = 1, 2, 3, 4 ; Y = 1, 2, 3, 4 \} \cup D
\]
Interpretations and Models

- An **interpretation** of a set of predicates assigns truth or falsehood to every possible instance (grounding) of those predicates.
- An **interpretation** can be specified as the set of ground atoms \( TRUE \) in it.
- A **model** of a program is an interpretation that makes the rules true for any assignment of values from the domain.
- \( P(D) \) is the minimal model of \( P \) consistent with \( D \).
Logic Program

• A set of fully grounded rules of the form
  \[ Q \leftarrow P_1, \ldots, P_n, \text{not } R_1, \ldots, \text{not } R_m. \quad (1) \]
  \[ \leftarrow P_1, \ldots, P_n, \text{not } R_1, \ldots, \text{not } R_m. \quad (IC) \]
  where are atoms

• A set of atoms \( M \) is an answer set for \( P \) if
  • it is a minimal model for
    \[ P^M = \{ Q \leftarrow P_1, \ldots, P_n | Q \leftarrow P_1, \ldots, P_n, \text{not } R_1, \ldots, \text{not } R_m \in P, \]
    \[ M \cap \{ R_1, \ldots, R_m \} = \emptyset \} \], and
  • \( M \) satisfies all the ICs in \( P \)
Models and Answer Sets

- As in propositional logic an *interpretation* can be specified as the set of atoms *TRUE* in it.
- Under this: a set of atoms $M$ can denote both an answer set and an interpretation.
Example:

• Consider the following program $P$:

\[ a \leftarrow b, c. \quad b \leftarrow a. \]
\[ a \leftarrow \text{not } c. \quad c \leftarrow d, \text{not } e. \]
\[ d \leftarrow b, c. \quad c \leftarrow \text{not } a. \]
Example:

- Consider the following program $P$:

  $a \leftarrow b, c.$  
  $b \leftarrow a.$  
  $a \leftarrow \text{not } c.$  
  $c \leftarrow d, \text{not } e.$  
  $d \leftarrow b, c.$  
  $c \leftarrow \text{not } a.$

- $P$ has 2 answer sets: $\{a, b\}$, $\{c\}$. 
Clark’s Completion

- For program $P$:

  $\begin{align*}
  a & \leftarrow b, c. \\
  b & \leftarrow a. \\
  a & \leftarrow \text{not } c. \\
  c & \leftarrow d, \text{not } e. \\
  d & \leftarrow b, c. \\
  c & \leftarrow \text{not } a.
  \end{align*}$
Clark’s Completion

- For program $P$:

  \[ a \leftarrow b, c. \quad b \leftarrow a. \]
  \[ a \leftarrow \text{not } c. \quad c \leftarrow d, \text{not } e. \]
  \[ d \leftarrow b, c. \quad c \leftarrow \text{not } a. \]

- Clark’s completion $Comp(P)$ is

  \[ a \equiv (b \land c) \lor (\neg c). \quad b \equiv a. \]
  \[ c \equiv (d \land \neg e) \lor (\neg a). \quad d \equiv (b \land c). \]
  \[ \neg e. \]
Completion and answer set

- $Comp(P)$ has 4 models: \{a, b\}, \{c\}, \{a, b, c, d\}.
- Remember $P$ has 2 answer sets: \{a, b\}, \{c\}.
Completion and answer set

- $Comp(P)$ has 4 models: \{a, b\}, \{c\}, \{a, b, c, d\}.
- Remember $P$ has 2 answer sets: \{a, b\}, \{c\}.
- If $M$ is an answer set of $P$ then $M$ is a model of $Comp(P)$. The other direction may not be true if "$P$ has loops."
- If "$P$ has loops," we can add extra clauses to make models = answer sets.
Loop

- Let $G_P = (V, E)$ where $V = \text{atoms}(P)$, and
  \[ E = \{(a, b) \mid (a \leftarrow b, G) \in P\} \]

- $L \subseteq \text{atoms}(P)$ is a loop of $P$ if there is a path between any two atoms in $L$ formed only with nodes in $L$.

- Loops in the previous program:
Loop

- Let $G_P = (V, E)$ where $V = \text{atoms}(P)$, and

\[ E = \{ (a, b) \mid (a \leftarrow b, G) \in P \} \]

- $L \subseteq \text{atoms}(P)$ is a loop of $P$ if there is a path between any two atoms in $L$ formed only with nodes in $L$

- Loops in the previous program:

\[
\begin{align*}
&\{a, b\} \\
&\{c, d\} \\
&\{a, b, c, d\}
\end{align*}
\]
Loop formula

For each loop, e.g., \( \{a, b\} \), add some formulas like:

\[ c \rightarrow (\neg a \land \neg b). \]
Loop formula

- For each loop, e.g., \{a, b\}, add some formulas like:

\[ c \rightarrow (\neg a \land \neg b). \]

- This is a loop formula.
Loop formula

- For each loop, e.g., \( \{a, b\} \), add some formulas like:

\[
c \rightarrow (\neg a \land \neg b).
\]

- This is a loop formula.

- From Lin’s and Zhao’s ASSAT paper:

\[
\text{model} (\text{Completion} + \text{loop formulas}) = \text{answer set}
\]
SAT-based Answer Set solver

- If a program has no loops, we say it is **tight**.
- For tight program we can use SAT solvers to find answer sets.
- When program is not tight, we need to add loop formulas.
- One way: first compute all loop formulas and add them to completion, then call a SAT solver.
- Problem: there may be an exponential number of loops.
ASSAT procedure

- The solution: generate and test.
  1. $DB = Comp(P)$.
  2. Invoke SAT solver to find a model $M$ of $DB$. If none, return $False$.
  3. Test if $M$ is an answer set.
  4. If yes, return $True$.
  5. If no, find some loops whose formulas $F$ are not satisfied by $M$, $DB = DB + F$, go to step (2).
// Input: DB = Comp(P) in CNF, S = ∅ is an assignment, ie a consistent set of literals
DLL(DB, S)

• if DB = ∅ return True
• if ∅ ∈ DB return False
• if \{l\} ∈ DB return DLL(assign(l, DB), S ∪ \{l\})
  A := an atom occurring in DB
  return DLL(assign(A, DB), S ∪ \{A\}) or
  DLL(assign(¬A, DB), S ∪ \{¬A\})
Definition of Assignment

- If $A$ is an atom in $DB$, $assign(A, DB)$ is the set of clauses obtained from $DB$ by removing the clauses to which $A$ belongs, and by removing $\neg A$ from the other clauses in $DB$.
- $assign(\neg A, DB)$ is defined similarly.
**SAT-based AS Generator**

\[
DLL(DB, S)
\]

- // modify first statement
- if \(DB = \emptyset\) return \(test(S, P)\)
- // where \(test(S, P)\):
  - returns \(True\) and \(S \cap \text{atoms}(P)\) if it is an answer set of \(P\)
  - returns \(False\) otherwise
- // rest is the same
**Lin’s Idea**

\[ DLL(DB, S) \]

- If \( DB = \emptyset \)
  - Returns \( True \) and \( S \cap \text{atoms}(P) \) if it is an answer set of \( P \)
  - Returns \( False \) otherwise

On \( False \): backtrack (like current state-of-the-art SAT solvers), but use answer set extensibility checking on a subset of \( S \) (ie on partial assignments) and find loop formulas active on current assignment (to guide search).
Findings

- Loops needed to guide search can be found in linear time.
- This enable us to take advantage of two important SAT techniques: backjumping and conflict learning.
Implementation

- On top of SAT solver *MChaff*, the default SAT solver used by *ASSAT* and *Cmodels*. 
Experimental Results

- We have tested some benchmarks including Hamiltonian circuit problems and Bounded Model Checking problems.
- The new solver outperforms other ASP solvers in many cases.
- In other cases, the solver is at least on the same level as the top performers.
Future work

- Develop a better scheme/heuristics on the timing of ASP propagation.
- Extend to support disjunctive programs.
- Implement support for weight expression with techniques from constraint programming.
Thanks and ... Questions?
Appendix - Atmost()

- Generalized-reduct: given a set of literals $B$ and rule $r : a_0 \leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m$, define
  
  $$r^{(B)} = \begin{cases} 
  \emptyset, & \text{if } \exists (i, j) (b_i \in B \text{ or } \neg a_j \in B) \\
  \{a_0 \leftarrow a_1, \ldots, a_n\}, & \text{otherwise.} 
  \end{cases}$$

- For program $P$, $\text{Atmost}(B)$ is the deductive closure of $P^{(B)}$, where $P^{(B)} = \{r^{(B)} | r \in P\}$. 

Appendix - Theorem

- Given program $P$, let $X$ be a partial assignment agrees with $\text{Comp}(P)$, i.e. for any clause $p \equiv Q$ in $\text{Comp}(P)$, $p$ is True whenever $Q$ is evaluated as True and $p$ is False whenever $Q$ is evaluated as False.

- Let $\text{cons}(P^{(X)})$ denote those atoms derivable from $P^{(X)}$, let $U = \text{atoms}(P) - X^-$, where $X^-$ is the set of atoms that are assigned False under $X$, and let $Y = U - \text{cons}(P^{(X)})$.

- (1) We claim if $Y \neq \emptyset$, then there must be a maximal loop under $Y$ such that for it’s loop formula $\text{LHS } \supset \text{RHS}$, $\text{LHS}$ is evaluated as False under $X$. 

A Fast SAT-based Answer Set Solver - p.25/37
Appendix - Theorem - contd.

(2) On the other hand, if there is a maximal loop $L$ under $U$ such that the LHS of $LF(L, P)$ if evaluated as $False$ according to the partial assignment $X$, then $Y \neq \emptyset$.

From (1) we know if there is no loop in the program, $Atmost()$ testing is useless. This is a source of inefficiency in Smodels, which uses $Atmost()$ extensively for all programs.

With the above theorem, when we do $Atmost()$ pruning we can use loop formulas in backjumping and conflict learning, which are two important ingredients of some successful SAT solvers.
Clark completion

- Given normal program $P$ with rules in form

$$Q \leftarrow P_1, \ldots, P_n, \text{not } R_1, \ldots, \text{not } R_m. \quad (1)$$

- Step 1: Replace each rule of the form (1) with the clause

$$Q \leftarrow P_1 \land \cdots \land P_n \land \neg R_1 \land \cdots \land \neg R_m. \quad (2)$$

- Step 2: Suppose

$$\{(Q \leftarrow G_1.), \ldots, (Q \leftarrow G_k.)\}$$

is the set of all clauses with $Q$ in the head, Replace it with the clause

$$Q \equiv G_1 \lor \cdots \lor G_k. \quad (3)$$
• If $Q$ appears only in the body of clauses, add

$\neg Q$.

• If there are constraint rules of form

$$\leftarrow P_1, \ldots, P_n, \text{not } R_1, \ldots, \text{not } R_m,$$

replace them with clauses of form

$$\neg (P_1 \land \cdots \land P_n \land \neg R_1 \land \cdots \land \neg R_m)$$
Observation

- Consider the following program:

\[
\begin{align*}
a & \leftarrow b. \quad b & \leftarrow a. \\
p_1 & \leftarrow \text{not } p_2. \quad p_2 & \leftarrow \text{not } p_1. \\
p_3 & \leftarrow \text{not } p_4. \quad p_4 & \leftarrow \text{not } p_3. \\
& \leftarrow p_1, p_3. \quad & \leftarrow p_2, p_3. \\
& \leftarrow p_1, p_4. 
\end{align*}
\]
Observation

- Consider the following program:

\[
\begin{align*}
    a &\leftarrow b. & b &\leftarrow a. \\
    p_1 &\leftarrow \text{not } p_2. & p_2 &\leftarrow \text{not } p_1. \\
    p_3 &\leftarrow \text{not } p_4. & p_4 &\leftarrow \text{not } p_3. \\
    &\leftarrow p_1, p_3. & &\leftarrow p_2, p_3. \\
    &\leftarrow p_1, p_4. & &
\end{align*}
\]

- Clearly \( a \) and \( b \) can not be in any answer set. So there is no need to find any model containing \( a \) and \( b \).
Observation

• Consider the following program:

\[
  a \leftarrow b. \quad b \leftarrow a.
\]
\[
  p_1 \leftarrow \text{not } p_2. \quad p_2 \leftarrow \text{not } p_1.
\]
\[
  p_3 \leftarrow \text{not } p_4. \quad p_4 \leftarrow \text{not } p_3.
\]
\[
  \leftarrow p_1, p_3. \quad \leftarrow p_2, p_3.
\]
\[
  \leftarrow p_1, p_4.
\]

• Clearly \( a \) and \( b \) can not be in any answer set. So there is no need to find any model containing \( a \) and \( b \).
Motivation

- In many cases we can tell a partial assignment can not be extended to answer set.
- This motivates us to develop a new solver that
  - uses SAT solver techniques to find model,
  - test answer set requirement before full assignment.
Answer set testing

- We use $Atmost()$ function from Smodels to do answer set testing.
- Given a set of literals $B$, $Atmost(B)$ returns a set of atoms, such that, if atom $p$ is not in $Atmost(B)$, $p$ cannot be in any answer set that agrees with $B$. 
## HC encoded as normal logic program

<table>
<thead>
<tr>
<th>Graph</th>
<th>Smodes</th>
<th>ASSAT</th>
<th>New(2)</th>
<th>New(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x30.1</td>
<td>0.19</td>
<td>21.29</td>
<td>0.29</td>
<td>0.61</td>
</tr>
<tr>
<td>2x30.2</td>
<td>—</td>
<td>22.53</td>
<td>0.16</td>
<td>3.89</td>
</tr>
<tr>
<td>2x30.3</td>
<td>—</td>
<td>16.02</td>
<td>0.30</td>
<td>0.23</td>
</tr>
<tr>
<td>2x30.4</td>
<td>—</td>
<td>466.51</td>
<td>68.78</td>
<td>70.63</td>
</tr>
<tr>
<td>3c20a</td>
<td>—</td>
<td>1049.54</td>
<td>6.10</td>
<td>0.93</td>
</tr>
<tr>
<td>r12.8</td>
<td>1.19</td>
<td>210.08</td>
<td>0.76</td>
<td>0.76</td>
</tr>
</tbody>
</table>
**HC encoded as Smo**

dels program

<table>
<thead>
<tr>
<th>Graph</th>
<th>Smodels</th>
<th>Cmodels</th>
<th>New(2)</th>
<th>New(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x30.1</td>
<td>0.06</td>
<td>35.92</td>
<td>0.06</td>
<td>0.16</td>
</tr>
<tr>
<td>2x30.2</td>
<td>—</td>
<td>10.73</td>
<td>0.36</td>
<td>0.67</td>
</tr>
<tr>
<td>2x30.3</td>
<td>—</td>
<td>10.75</td>
<td>0.36</td>
<td>0.65</td>
</tr>
<tr>
<td>2x30.4</td>
<td>—</td>
<td>1657.89</td>
<td>26.33</td>
<td>28.80</td>
</tr>
<tr>
<td>3c20a</td>
<td>—</td>
<td>18.52</td>
<td>0.48</td>
<td>0.32</td>
</tr>
<tr>
<td>cg200</td>
<td>—</td>
<td>—</td>
<td>4.93</td>
<td>3.80</td>
</tr>
<tr>
<td>cg300</td>
<td>—</td>
<td>—</td>
<td>15.66</td>
<td>10.07</td>
</tr>
</tbody>
</table>
### Bounded Model Checking problems

<table>
<thead>
<tr>
<th>Program</th>
<th>Smodels</th>
<th>Cmodels</th>
<th>New(2)</th>
<th>New(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp-10i-o2-b11</td>
<td>139.36</td>
<td>307.55</td>
<td>79.53</td>
<td>25.69</td>
</tr>
<tr>
<td>dp-10s-o2-b8</td>
<td>6.93</td>
<td>12.42</td>
<td>2.65</td>
<td>1.40</td>
</tr>
<tr>
<td>dp-12s-o2-b9</td>
<td>120.84</td>
<td>10.30</td>
<td>14.24</td>
<td>6.94</td>
</tr>
<tr>
<td>dp-10i-o2-b12</td>
<td>128.07</td>
<td>15.52</td>
<td>2.29</td>
<td>2.41</td>
</tr>
<tr>
<td>dp-10s-o2-b9</td>
<td>11.53</td>
<td>5.44</td>
<td>2.76</td>
<td>1.31</td>
</tr>
<tr>
<td>dp-12s-o2-b10</td>
<td>308.3</td>
<td>10.48</td>
<td>3.48</td>
<td>0.85</td>
</tr>
</tbody>
</table>
Cmodels vs. ASSAT

- Cmodels and ASSAT both use generate and test approach.
- Main difference: when Cmodels invokes a SAT solver, it does not use it as a "black box" as ASSAT does.
- Unlike ASSAT, Cmodels just backtracks and continues search.
ASP propagation

- **ASP propagation** is the following process:
  - (1) $B_0 = \text{partial assignment}$.
  - (2) Using $Atmost(B_0)$ from Smodels: compute $N$, a set of atoms that cannot be in any answer set that agrees with $B_0$.
  - (3) $B = B_0 \cup \text{not}(N)$.

- If there are some $p$ and $\neg p$ both in $B$, we say there are conflicts.

- Otherwise, if $B - B_0 \neq \emptyset$, we say there are implications.
New procedure frame work

- (1) DB = Comp(P).
- (2) Assign an unassigned atom. If none return True.
- (3) Do unit propagation on DB.
- (4) If there are conflicts, backtrack.
- (5) Do ASP propagation.
- (6) If there are conflicts, backtrack.
- (7) If there are implications, go to step (3).
- (8) Go to step (2).