Aalto University
School of Science

SAT and SMT for Answer Set Programming

Tomi Janhunen

Aalto University
School of Science
Department of Information and Computer Science
Tomi.Janhunen@aalto.fi

(Partly joint work with Ilkka Niemelä)

SAT/SMT School, Trento, June 15, 2012

Outline

1. ANSWER SET PROGRAMMING (ASP)

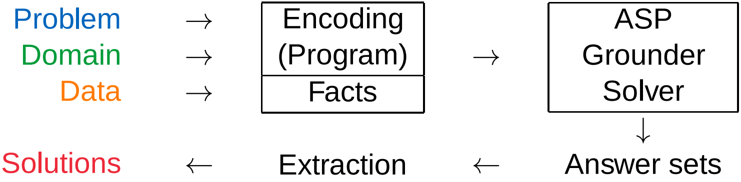
- A declarative programming paradigm from the late 90s: [Niemelä, 1999; Marek and Truszczyński, 1999; Gelfond and Leone, 2002; Baral, 2003; Brewka et al., 2011]
- The syntax is based on PROLOG-style rules.
- The semantics of a program is determined by its stable models [Gelfond and Lifschitz, 1988] a.k.a. answer sets.
- Answer sets are computed using answer set solvers:

SMODELS	`www.tcs.hut.fi/Software/smodels/`
DLV	`www.dlvsystem.com/index.php/DLV`
CMODELS	`www.cs.utexas.edu/~tag/cmodels/`
CLASP	`www.cs.uni-potsdam.de/clasp/`

- Applications of ASP: product configuration, combinatorial problems, planning, verification, information integration, ...

Modeling Principles for ASP

- Typical problem encodings aim at a very tight (1-to-1) correspondence between solutions and answer sets.
- A uniform encoding is independent of the input instance.
- Rules with variables are treated via Herbrand instantiation.



Rule-Based Syntax

Typical programs involve normal rules (1), constraints (2), or choice (3), cardinality (4), weight (6), or disjunctive (7) rules:

$$
\begin{array}{rcll}
a & \leftarrow & b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m. & (1) \\
 & \leftarrow & b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m. & (2) \\
\{a_1, \ldots, a_h\} & \leftarrow & b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m. & (3) \\
a & \leftarrow & l \leq \{b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m\}. & (4) \\
a & \leftarrow & w \leq [b_1 = w_{b_1}, \ldots, b_n = w_{b_n} & (5) \\
 & & \quad \mathbf{not}\, c_1 = w_{c_1}, \ldots, \mathbf{not}\, c_m = w_{c_m}]. & (6) \\
a_1 \mid \ldots \mid a_h & \leftarrow & b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m. & (7)
\end{array}
$$

ASP systems support further extensions and syntactic sugar!

Example: Some Dinner Rules

Main course:

$\{\text{dinner}\}.\quad \{\text{beef}, \text{pork}, \text{fish}\} \leftarrow \text{dinner}.$

$\leftarrow \text{dinner}, \textbf{not}\ \text{beef}, \textbf{not}\ \text{pork}, \textbf{not}\ \text{fish}.$

$\text{toomany} \leftarrow 2 \leq \{\text{beef}, \text{pork}, \text{fish}\}.$

$\leftarrow \text{toomany}.$

Drinks:

$\{\text{bycar}\} \leftarrow \text{dinner}.$

$\text{red} \leftarrow \text{beef}, \textbf{not}\ \text{bycar}.\quad \text{red} \leftarrow \text{pork}, \textbf{not}\ \text{bycar}.$

$\text{white} \leftarrow \text{fish}, \textbf{not}\ \text{bycar}.\quad \text{wine} \leftarrow \text{red}.\quad \text{wine} \leftarrow \text{white}.$

$\text{water} \leftarrow \text{dinner}, \textbf{not}\ \text{wine}.$

Budget:

$\text{bankrupt} \leftarrow 26 \leq$

$\quad [\text{beef} = 20, \text{pork} = 15, \text{fish} = 25, \text{red} = 7, \text{white} = 5].$

$\leftarrow \text{bankrupt}.$

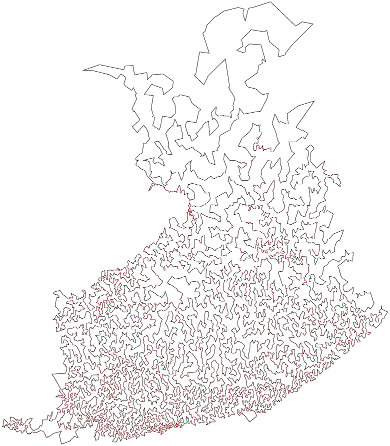
Simple Demo

```
$ gringo dinner.lp | clasp 0
clasp version 1.3.5
Reading from stdin
Solving...
Answer: 1

Answer: 2
dinner pork bycar water
Answer: 3
dinner beef bycar water
Answer: 4
dinner fish bycar water
Answer: 5
dinner pork red wine
SATISFIABLE

Models      : 5
Time        : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

Example: the Hamiltonian Cycle Problem (HCP)

Definition

Given an input graph $G = \langle N, E \rangle$ find a cycle which visits each node in N exactly once through the edges in $E \subseteq N^2$.



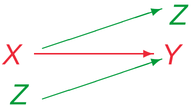
[www.tsp.gatech.edu]

Example: the Hamiltonian Cycle Problem (HCP)

- Suppose that the input graph G is given as a set of facts

$$\text{edge}(a,b),\ \ \text{edge}(b,c),\ \ \text{edge}(c,a),\ \ \ldots$$

- The following rules capture the Hamiltonian cycles of G:



$$\begin{array}{rcl}
\text{node}(X) & \leftarrow & \text{edge}(Y,X). \\
\text{node}(Y) & \leftarrow & \text{edge}(Y,X). \\
\text{in}(X,Y) & \leftarrow & \text{edge}(X,Y),\ \textbf{not}\ \text{out}(X,Y). \\
\text{out}(X,Y) & \leftarrow & \text{edge}(X,Y),\ \text{edge}(X,Z),\ \text{in}(X,Z),\ Y \neq Z. \\
\text{out}(X,Y) & \leftarrow & \text{edge}(X,Y),\ \text{edge}(Z,Y),\ \text{in}(Z,Y),\ X \neq Z. \\
\text{reach}(a). & & \\
\text{reach}(Y) & \leftarrow & \text{edge}(X,Y),\ \text{in}(X,Y),\ \text{reach}(X). \\
 & \leftarrow & \textbf{not}\ \text{reach}(X),\ \text{node}(X).
\end{array}$$

Answer-Set Semantics

- A normal program P is a finite set of normal rules.
- The Herbrand universe and the Herbrand base of P are denoted by $\mathrm{HU}(P)$ and $\mathrm{HB}(P)$, respectively.
- The formal semantics of a program P is determined by its answer sets $S \subseteq \mathrm{HB}(P)$ satisfying

$$S = \mathrm{cl}(\mathrm{Gnd}(P)^S)$$

where

1. the ground program $\mathrm{Gnd}(P)$ consists of all instances $r\sigma$ of rules $r \in P$ obtained by substitutions σ over $\mathrm{HU}(P)$;
2. the reduct $\mathrm{Gnd}(P)^S$ contains a positive rule $a \leftarrow b_1, \ldots, b_n$ for each $a \leftarrow b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m \in \mathrm{Gnd}(P)$ such that $c_1 \notin S, \ldots, c_m \notin S$; and
3. the closure $\mathrm{cl}(\mathrm{Gnd}(P)^S)$ is the least subset of $\mathrm{HB}(P)$ closed under the rules of $\mathrm{Gnd}(P)^S$.

Intelligent Grounding

- A rule with variables stands for its all ground instances.
- For the universe $\{a, b, c\}$, there are 9 instances of

$$\mathrm{in}(X, Y) \leftarrow \mathrm{edge}(X, Y), \mathbf{not}\, \mathrm{out}(X, Y).$$

- In the presence of $\mathrm{edge}(a, b)$, $\mathrm{edge}(b, c)$, and $\mathrm{edge}(c, a)$, i.e., facts describing the input graph, only 3 are needed:

$$\mathrm{in}(a, b) \leftarrow \mathrm{edge}(a, b), \mathbf{not}\, \mathrm{out}(a, b).$$
$$\mathrm{in}(b, c) \leftarrow \mathrm{edge}(b, c), \mathbf{not}\, \mathrm{out}(b, c).$$
$$\mathrm{in}(c, a) \leftarrow \mathrm{edge}(c, a), \mathbf{not}\, \mathrm{out}(c, a).$$

- In general, grounding can be a computationally hard task but a number of efficient implementations exist:
  - LPARSE [Syrjänen, 2001]
  - DLV [Perri et al., 2007]
  - GRINGO [Gebser et. al, 2007]
- Database techniques and minimal models are exploited.

Example: Complete Ground Program for a HCP

$$\begin{array}{l}
\mathrm{edge}(a,b).\ \ \mathrm{edge}(b,c).\ \ \mathrm{edge}(c,a).\ \ \mathrm{node}(a).\ \ \mathrm{node}(b).\\
\mathrm{edge}(b,a).\ \ \mathrm{edge}(c,b).\ \ \mathrm{edge}(a,c).\ \ \mathrm{node}(c).
\end{array}$$

$$\begin{array}{ll}
\mathrm{in}(a,b) \leftarrow \textbf{not}\, out(a,b). & \mathrm{in}(b,a) \leftarrow \textbf{not}\, out(b,a).\\
\mathrm{in}(b,c) \leftarrow \textbf{not}\, out(b,c). & \mathrm{in}(c,b) \leftarrow \textbf{not}\, out(c,b).\\
\mathrm{in}(c,a) \leftarrow \textbf{not}\, out(c,a). & \mathrm{in}(a,c) \leftarrow \textbf{not}\, out(a,c).
\end{array}$$

$$\begin{array}{ll}
\mathrm{out}(a,b) \leftarrow in(a,c). & \mathrm{out}(a,c) \leftarrow in(a,b).\\
\mathrm{out}(a,b) \leftarrow in(c,b). & \mathrm{out}(a,c) \leftarrow in(b,c).\\
\mathrm{out}(b,a) \leftarrow in(b,c). & \mathrm{out}(b,c) \leftarrow in(a,c).\\
\mathrm{out}(b,a) \leftarrow in(c,a). & \mathrm{out}(b,c) \leftarrow in(b,a).\\
\mathrm{out}(c,a) \leftarrow in(b,a). & \mathrm{out}(c,b) \leftarrow in(a,b).\\
\mathrm{out}(c,a) \leftarrow in(c,b). & \mathrm{out}(c,b) \leftarrow in(c,a).
\end{array}$$

$$\begin{array}{ll}
\mathrm{reach}(b) \leftarrow \mathrm{in}(a,b). & \mathrm{reach}(b) \leftarrow \mathrm{in}(c,b), \mathrm{reach}(c).\\
\mathrm{reach}(c) \leftarrow \mathrm{in}(a,c). & \mathrm{reach}(c) \leftarrow \mathrm{in}(b,c), \mathrm{reach}(b).
\end{array}$$

$$\mathrm{reach}(a).\ \ \leftarrow \textbf{not}\, \mathrm{reach}(b).\ \ \leftarrow \textbf{not}\, \mathrm{reach}(c).\ \ \leftarrow \textbf{not}\, \mathrm{reach}(d).$$

Example: Computing the Reduct

Consider $S = \{\text{edge}(a,b), \text{edge}(b,c), \text{edge}(c,a), \text{edge}(b,a),$
$\text{edge}(c,b), \text{edge}(a,c), \text{node}(a), \text{node}(b), \text{node}(c),$
$\text{in}(a,b), \text{in}(b,c), \text{in}(c,a),$
$\text{out}(b,a), \text{out}(c,b), \text{out}(a,c),$
$\text{reach}(a), \text{reach}(b), \text{reach}(c), \text{reach}(d)\}$

1. The rules involving **not**, i.e.,

$$\text{in}(a,b) \leftarrow \textbf{not}\, out(a,b). \quad \text{in}(b,a) \leftarrow \textbf{not}\, out(b,a).$$
$$\text{in}(b,c) \leftarrow \textbf{not}\, out(b,c). \quad \text{in}(c,b) \leftarrow \textbf{not}\, out(c,b).$$
$$\text{in}(c,a) \leftarrow \textbf{not}\, out(c,a). \quad \text{in}(a,c) \leftarrow \textbf{not}\, out(a,c).$$

reduce into facts: $\text{in}(a,b)$. $\text{in}(b,c)$. $\text{in}(c,a)$.

2. The set S satisfies the constraints:

$$\leftarrow \textbf{not}\, \text{reach}(b). \ \leftarrow \textbf{not}\, \text{reach}(c). \ \leftarrow \textbf{not}\, \text{reach}(d).$$

Example: Computing the Closure

The rules of the reduct $\text{Gnd}(P)^S$ are:

$\text{edge}(a, b)$. $\text{edge}(b, c)$. $\text{edge}(c, a)$. $\text{node}(a)$. $\text{node}(b)$.
$\text{edge}(b, a)$. $\text{edge}(c, b)$. $\text{edge}(a, c)$. $\text{node}(c)$.
$\text{in}(a, b)$. $\text{in}(b, c)$. $\text{in}(c, a)$.

$\text{out}(a, b) \leftarrow \text{in}(a, c)$.	$\text{out}(a, c) \leftarrow \text{in}(a, b)$.
$\text{out}(a, b) \leftarrow \text{in}(c, b)$.	$\text{out}(a, c) \leftarrow \text{in}(b, c)$.
$\text{out}(b, a) \leftarrow \text{in}(b, c)$.	$\text{out}(b, c) \leftarrow \text{in}(a, c)$.
$\text{out}(b, a) \leftarrow \text{in}(c, a)$.	$\text{out}(b, c) \leftarrow \text{in}(b, a)$.
$\text{out}(c, a) \leftarrow \text{in}(b, a)$.	$\text{out}(c, b) \leftarrow \text{in}(a, b)$.
$\text{out}(c, a) \leftarrow \text{in}(c, b)$.	$\text{out}(c, b) \leftarrow \text{in}(c, a)$.
$\text{reach}(b) \leftarrow \text{in}(a, b)$.	$\text{reach}(b) \leftarrow \text{in}(c, b), \text{reach}(c)$.
$\text{reach}(c) \leftarrow \text{in}(a, c)$.	$\text{reach}(c) \leftarrow \text{in}(b, c), \text{reach}(b)$.
$\text{reach}(a)$.	

$\implies S = \text{cl}(\text{Gnd}(P)^S)$ so that S is an answer set.

Key Features of ASP

- Typical ASP encodings follow a three-phase design:
  - Generate the solution candidates
  - Define the required concepts
  - Test if a candidate satisfies its criteria
- Default negation favors concise encodings.
- Basic database operations are definable in terms of rules:
  - Projection: $\mathrm{node}(X) \leftarrow \mathrm{edge}(Y, X).$
  - Union: $\mathrm{node}(X) \leftarrow \mathrm{edge}(Y, X).\quad \mathrm{node}(Y) \leftarrow \mathrm{edge}(Y, X).$
  - Intersection: $\mathrm{symm}(X, Y) \leftarrow \mathrm{edge}(X, Y), \mathrm{edge}(Y, X).$
  - Complement: $\mathrm{unidir}(X, Y) \leftarrow \mathrm{edge}(X, Y), \mathbf{not}\, \mathrm{edge}(Y, X).$
- Moreover, recursive definitions can be written, e.g., to capture various kinds of closures of relations:

$$\mathrm{path}(X, Y) \leftarrow \mathrm{path}(X, Z), \mathrm{path}(Z, Y).$$

$\Longrightarrow$ ASP = KR + DDB + Search

Solver Technology Behind the CLASP System

- Conflict analysis via the FirstUIP scheme
- Nogood recording and deletion
- Backjumping
- Restarts
- Conflict-driven decision heuristics
- Progress saving
- Unit propagation via watched literals
- Dedicated propagation of binary and ternary nogoods
- Dedicated propagation of cardinality/weight rules
- Equivalence reasoning
- Resolution-based preprocessing

[Gebser et al., 2007] [http://www.cs.uni-potsdam.de/clasp/]

Translation-Based Approach

- Counts on translations from ASP to other formalisms like
  - Propositional satisfiability (SAT)
  - Satisfiability modulo theories (SMT)
  - Linear programming (LP)
  - Mixed integer programming (MIP)
- The idea is to combine the expressiveness of rules with the existing powerful solver technology for SAT, SMT, . . .
- Further language extensions can be implemented by
  - — devising suitable translations for the extensions and
  - — using solvers as black boxes for computations.
- Solver technology is constantly improving and we expect to gain from this development work using translations.

2. TRANSLATING ASP INTO SAT

- SAT solvers provide a promising computational platform to implement the rule-based reasoning required in ASP.
- A number of ASP systems exploiting SAT solvers exist:
  - ASSAT [Lin and Zhao, 2004]
  - CMODELS [Giunchiglia et al., 2006]
  - LP2SAT [T.J., 2004]
  - LP2SAT2 [T.J. and Niemelä, 2011]
- However, due to the global nature of answer sets, devising a translation from ASP to SAT is nontrivial.

Example

$$\{a \leftarrow \mathbf{not}\, b.\quad b \leftarrow \mathbf{not}\, a.\ \} \longmapsto \{a \lor b, \neg a \lor \neg b\}.$$

$$\{a \leftarrow b.\quad b \leftarrow a.\ \} \longmapsto \{a \lor \neg b, \neg a \lor b\} \cup \{\neg a \lor \neg b\}\ !?$$

Fundamental Properties: PFM Translations

A translation function Tr is PFM iff it is

polynomial, i.e., for some polynomial f, the translation $\mathrm{Tr}(P)$ can be computed in at most $f(\|P\|)$ steps,

faithful, i.e., for all programs P

$$P \equiv_{\mathrm{v}} \mathrm{Tr}(P),$$

and modular, i.e., for all programs P and Q,

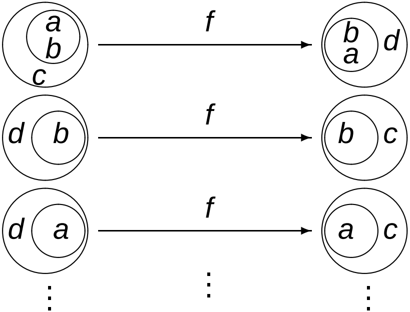
$$\mathrm{Tr}(P \cup Q) \equiv_{\mathrm{v}} \mathrm{Tr}(P) \cup \mathrm{Tr}(Q).$$

In the above, $\equiv_{\mathrm{v}}$ denotes visible equivalence which is based on the visible Herbrand base $\mathrm{HB}_{\mathrm{v}}(P)$ of the program P.

Visible Equivalence

- Visible equivalence $P \equiv_v Q$ requires $\mathrm{HB_v}(P) = \mathrm{HB_v}(Q)$ and a bijection $f : \mathrm{AS}(P) \to \mathrm{AS}(Q)$ such that $\forall S \in \mathrm{AS}(P)$,

$$S \cap \mathrm{HB_v}(P) = f(S) \cap \mathrm{HB_v}(Q).$$



- A newer variant of $\equiv_v$ insists on the coherence of f.

A PFM Translation from SAT to ASP

- A clause $A \vee \neg B$ is translated [Niemelä, 1999] into

$$\mathrm{Tr_N}(A \vee \neg B) = \{a \leftarrow \mathbf{not}\,\overline{a}.\;\; \overline{a} \leftarrow \mathbf{not}\,a. \mid a \in A \cup B\} \cup \{\leftarrow \mathbf{not}\,A,\, \mathbf{not}\,\overline{B}\}.$$

- For a set of clauses S,

$$\mathrm{Tr_N}(S) = \bigcup \{\mathrm{Tr_N}(A \vee \neg B) \mid A \vee \neg B \in S\}.$$

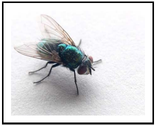
Theorem

For any sets of clauses S, S_1, and S_2,

1. $\mathrm{Tr}(S)$ *can be computed in linear time,*
2. $S \equiv_{\mathrm{v}} \mathrm{Tr_N}(S)$, *and*
3. $\mathrm{Tr_N}(S_1 \cup S_2) \equiv_{\mathrm{v}} \mathrm{Tr_N}(S_1) \cup \mathrm{Tr_N}(S_2)$.

Non-Modularity in Natural Language

Finnish idiom: “**Tehdä kärpäsestä härkänen.**”

 $\longmapsto$ 

Translation into English: “**To make a bull out of a fly.**”

 $\longmapsto$ 

Correct translation: “**To make a mountain out of a molehill.**”

[www.eluova.fi] [en.wikipedia.org]

Intranslatability Results

- There is no modular translation from logic programs to propositional theories [Niemelä, 1999].

Program	Answer sets		Theory
$P_1 = \{a\}$	$\{a\}$	$\mapsto$	$T_1 \models a$
$P_2 = \{a \leftarrow \mathbf{not}\, a\}$	-	$\mapsto$	$T_2 \models \bot$
$P_1 \cup P_2$	$\{a\}$	$\mapsto$	$T_1 \cup T_2 \models \bot$

- Such a translation is (likely) to be exponential if auxiliary atoms are not allowed [Lifschitz and Razborov, 2006]
- Systematic analysis leads to an expressive power hierarchy for classes of logic programs [T.J., 2006].

Expressive Power Hierarchy

- The (non)existence of PFM/FM translations induces:

Normal rules:	$a \leftarrow b_1, \ldots, b_n, \textbf{not}\, c_1, \ldots, \textbf{not}\, c_m.$
$\Downarrow_{\text{PFM}}$	$\cup\vert$
Binary rules:	$a \leftarrow b_1, b_2, \textbf{not}\, c_1, \ldots, \textbf{not}\, c_m.$
$\not\Downarrow_{\text{FM}}$	$\cup\vert$
Unary rules:	$a \leftarrow b, \textbf{not}\, c_1, \ldots, \textbf{not}\, c_m.$
$\not\Downarrow_{\text{FM}}$	$\cup\vert$
Atomic rules:	$a \leftarrow \textbf{not}\, c_1, \ldots, \textbf{not}\, c_m.$
$\not\Downarrow_{\text{FM}}$	$\Uparrow_{\text{PFM}}$
Clauses:	$a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_m$

- Any faithful translation from ASP to SAT is non-modular.
- Strict relationships do not depend on translation length!

Existing Translations

- The translation of [Ben-Eliyahu and Dechter, 1994] is not faithful in the strict sense of visible equivalence ($\equiv_{\mathrm{v}}$).
- In the worst case, an exponential number of loop formulas [Lin and Zhao, 2002] is required (incrementally).
- The translation of [Lin and Zhao, 2003] is faithful but quadratic.
- Level numberings [T.J., 2004] enable a faithful and sub-quadratic translation of length of

  $$O(\|P\| \times \log_2 n)$$

  where n is the size of the largest strongly connected component in the positive dependency graph of P.

Positive Dependency Graph

- Given a program P, the positive dependency graph G_P^+
  1. has $\mathrm{HB}(P)$ as the set of nodes and
  2. there is an edge $\langle a, b\rangle$ in G_P^+ whenever there is a rule $r \in P$ such that $a = \mathrm{H}(r)$ and $b \in \mathrm{B}^+(r)$.
- A strongly connected component (SCC) $S \subseteq \mathrm{HB}(P)$ of G_P^+ is a maximal subset of $\mathrm{HB}(P)$ such that every pair $a, b \in S$ is mutually reachable in G_P^+.

Example

$a \leftarrow b.$

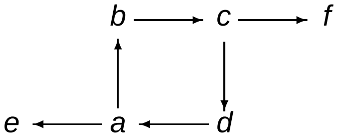
$a \leftarrow e.$

$b \leftarrow c.$

$c \leftarrow d.$

$c \leftarrow f.$

$d \leftarrow a.$



$S_1 = \{e\}$

$S_2 = \{f\}$

$S_3 = \{a, b, c, d\}$

Program Completion

- The idea [Clark, 1978] is to rewrite the defining rules $a \leftarrow B_1, \ldots, a \leftarrow B_n$ of an atom as an equivalence

$$a \leftrightarrow (\bigwedge B_1) \lor \cdots \lor (\bigwedge B_n)$$

where $\bigwedge B_i$ denotes the conjunction of literals in B_i.

- Program completion is faithful for tight programs under answer set semantics but not faithful in general:

$$\mathrm{CM}(\mathrm{Comp}(\{a \leftarrow a.\,\})) = \mathrm{CM}(\{a \leftrightarrow a\}) = \{\emptyset, \{a\}\}.$$

Example

In case of Niemelä's counter-example, we obtain:

1. $\mathrm{CM}(\mathrm{Comp}(\{a.\,\})) = \mathrm{CM}(\{a \leftrightarrow \top\}) = \{\{a\}\}$.
2. $\mathrm{CM}(\mathrm{Comp}(\{a \leftarrow \mathbf{not}\, a.\,\})) = \mathrm{CM}(\{a \leftrightarrow \neg a\}) = \emptyset$.
3. $\mathrm{CM}(\mathrm{Comp}(\{a.\ a \leftarrow \mathbf{not}\, a.\,\})) = \mathrm{CM}(\{a \leftrightarrow \top \lor \neg a\}) = \{\{a\}\}$.

Supported Sets (a.k.a. Supported Models)

- A supported set $S \subseteq \mathrm{HB}(P)$ of P [Apt et al., 1988] is
  - closed under the rules of P, i.e., for every $r \in P$, $S \models \mathrm{B}(r)$ implies $\mathrm{H}(r) \in S$, and
  - for each $a \in S$ there is a supporting rule $r \in P$ such that $\mathrm{H}(r) = a$ and $S \models \mathrm{B}(r)$.
- The set of supported sets of P is denoted by $\mathrm{SuppS}(P)$.
- For a set $S \subseteq \mathrm{HB}(P)$, define the set of supporting rules

$$\mathrm{SuppR}(P, S) = \{r \in P \mid S \models \mathrm{B}(r)\}.$$

Theorem (Marek and Subrahmanian, 1992)

For any normal program P,

1. $\mathrm{AS}(P) \subseteq \mathrm{SuppS}(P)$ *and*
2. $\mathrm{SuppS}(P) = \mathrm{CM}(\mathrm{Comp}(P))$.

Level Numbers

- Let S be a supported set of a normal program P.
- A function $\lambda : S \to \mathbb{N}$ is a level numbering for S iff for all atoms $a \in S$,

$$\lambda(a) = \min\{\lambda(B) \mid a \leftarrow B, \textbf{not}\, C \in \mathrm{SuppR}(P, M)\}$$

where

$$\lambda(B) = \max\{\lambda(b) \mid b \in B\} + 1.$$

- A level numbering λ of a supported set S is unique.

Theorem (T.J., 2004)

A supported set S of P is an answer set of P iff it has a level numbering $\lambda : S \to \mathbb{N}$.

Example

Consider a positive normal program P

$$a \leftarrow b. \quad b \leftarrow a.$$

and its supported sets $S_1 = \emptyset$ and $S_2 = \{a, b\}$:

1. There is a trivial level numbering $\lambda_1 : S_1 \to \mathbb{N}$ for S_1.
2. The requirements for a level numbering $\lambda_2 : S_2 \to \mathbb{N}$ are:

$$\begin{cases} \lambda_2(a) = \lambda_2(b) + 1 \\ \lambda_2(b) = \lambda_2(a) + 1 \end{cases}$$

$\implies$ There is no such level numbering λ_2.

Therefore, the only answer set of P is S_1.

Translation into Atomic Programs

- A faithful and polynomial-time translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ of a normal program P into an atomic normal program

$$\mathrm{Tr}_{\mathrm{SUPP}}(P) \cup \mathrm{Tr}_{\mathrm{CTR}}(P) \cup \mathrm{Tr}_{\mathrm{MIN}}(P) \cup \mathrm{Tr}_{\mathrm{MAX}}(P)$$

  where the parts of the translation

  1. $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ captures a supported set S and supporting rules,
  2. $\mathrm{Tr}_{\mathrm{CTR}}(P)$ chooses level numbers using binary counters,
  3. $\mathrm{Tr}_{\mathrm{MIN}}(P)$ ensures the minimality of $\lambda(a)$ for $a \in S$, and
  4. $\mathrm{Tr}_{\mathrm{MAX}}(P)$ ensures the maximality of $\lambda(\mathrm{B}^{+}(r))$ for $r \in \mathrm{SuppR}(P, S)$.

- A number of subprograms for counters are needed.
- The translation $\mathrm{Tr}_{\mathrm{AT}}$ is inherently non-modular but $\mathrm{Tr}_{\mathrm{AT}}(P)$ is always tight so that $P \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{AT}}(P) \equiv_{\mathrm{v}} \mathrm{Comp}(\mathrm{Tr}_{\mathrm{AT}}(P))$.

Example

For $P = \{a \leftarrow b.\quad b \leftarrow a.\ \}$, the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ contains:

$$
\begin{array}{l}
a \leftarrow \mathbf{not}\,\overline{\mathrm{bt}(r_1)}.\ \ \overline{\mathrm{bt}(r_1)} \leftarrow \mathbf{not}\,\mathrm{bt}(r_1).\ \ \mathrm{bt}(r_1) \leftarrow \mathbf{not}\,\overline{b}.\\
b \leftarrow \mathbf{not}\,\overline{\mathrm{bt}(r_2)}.\ \ \overline{\mathrm{bt}(r_2)} \leftarrow \mathbf{not}\,\mathrm{bt}(r_2).\ \ \mathrm{bt}(r_2) \leftarrow \mathbf{not}\,\overline{a}.\\
\overline{a} \leftarrow \mathbf{not}\,a.\ \ \overline{b} \leftarrow \mathbf{not}\,b.\\
\leftarrow \mathbf{not}\,\overline{a}, \mathbf{not}\,\mathrm{min}(a).\\
\leftarrow \mathbf{not}\,\overline{b}, \mathbf{not}\,\mathrm{min}(b).\\
\leftarrow \mathbf{not}\,\overline{\mathrm{bt}(r_1)}, \mathbf{not}\,\overline{\mathrm{lt}(\mathrm{nxt}(b), \mathrm{ctr}(a))_1}.\\
\leftarrow \mathbf{not}\,\overline{\mathrm{bt}(r_2)}, \mathbf{not}\,\overline{\mathrm{lt}(\mathrm{nxt}(a), \mathrm{ctr}(b))_1}.\\
\mathrm{min}(a) \leftarrow \mathbf{not}\,\overline{\mathrm{bt}(r_1)}, \mathbf{not}\,\overline{\mathrm{eq}(\mathrm{nxt}(b), \mathrm{ctr}(a))}\\
\mathrm{min}(b) \leftarrow \mathbf{not}\,\overline{\mathrm{bt}(r_2)}, \mathbf{not}\,\overline{\mathrm{eq}(\mathrm{nxt}(a), \mathrm{ctr}(b))}
\end{array}
$$

in addition to the required subprograms for counters.

The only answer set of $\mathrm{Tr}_{\mathrm{AT}}P$ is $N = \{\overline{a}, \overline{b}, \overline{\mathrm{bt}(r_1)}, \overline{\mathrm{bt}(r_2)}\}$.

3. FURTHER TRANSLATIONS

In this part, we will consider a number of translations from normal/SMODELS programs to

- difference logic [Niemelä, 2008; T.J. et al., 2009],
- fixed-width bit-vector theories [Nguyen et al., 2011],
- propositional satisfiability [T.J. and Niemelä, 2011] which improves the translation of [T.J., 2004] by
  1. covering extended rule types such as choice rules, cardinality rules, and weight rules [Simons, 1999];
  2. compacting the translation using ranking constraints; and
  3. removing the asymmetry of positive/negative subgoals.

Difference Logic

- The syntax of formulas in difference logic [Nieuwenhuis and Oliveras, 2005] is based on
  - atomic propositions a, b, c, $\ldots$,
  - simple linear constraints of the form $x_i + k \geq x_j$, and
  - propositional connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, and $\leftrightarrow$.
- On the semantical side, each interpretation assigns
  - a truth value $\top$ or $\bot$ to every propositional variable a, and
  - an integer value i to each integer variable x_j.
- Models are defined in the standard way.

Example

For instance, the formula

$$(x_1 + 2 \geq x_2) \leftrightarrow (p_1 \rightarrow \neg(x_2 + 2 \geq x_1))$$

is satisfied in an interpretation with $p_1 = \bot$, $x_1 = 1$, and $x_2 = 1$.

Representing the Completion

- A normal rule $r = a \leftarrow b_1, \ldots, b_n, \mathbf{not}\, c_1, \ldots, \mathbf{not}\, c_m$ in the definition $\mathrm{Def}_P(a)$ of an atom $a \in \mathrm{HB}(P)$ is written

$$\mathrm{bt}(r) \leftrightarrow b_1 \wedge \cdots \wedge b_n \wedge \neg c_1 \wedge \cdots \wedge \neg c_m.$$

- The atom a itself is defined by $a \leftrightarrow \bigvee_{r \in \mathrm{Def}_P(a)} \mathrm{bt}(r)$.
- E.g., for $\mathrm{Def}_P(a) = \{a \leftarrow a, b. \quad a \leftarrow \mathbf{not}\, d.\ \}$, we introduce:

$$\mathrm{bt}(r_1) \leftrightarrow a \wedge b, \quad \mathrm{bt}(r_2) \leftrightarrow \neg d, \quad a \leftrightarrow \mathrm{bt}(r_1) \vee \mathrm{bt}(r_2).$$

- Given $\mathrm{SCC}(a)$, the definition $\mathrm{Def}_P(a)$ splits into two disjoint, external and internal parts:

$$\begin{array}{rcl} \mathrm{Ext}_P(a) & = & \{r \in \mathrm{Def}_P(a) \mid \mathrm{B}^+(r) \cap \mathrm{SCC}(a) = \emptyset\} \text{ and} \\ \mathrm{Int}_P(a) & = & \{r \in \mathrm{Def}_P(a) \mid \mathrm{B}^+(r) \cap \mathrm{SCC}(a) \neq \emptyset\}. \end{array}$$

Weak Ranking Constraints in Difference Logic

- The external and internal support of $a \in \text{HB}(P)$ having a non-trivial $\text{SCC}(a)$ depend on $\text{Def}_P(a) = \text{Ext}_P(a) \sqcup \text{Int}_P(a)$:

$$\text{ext}(a) \leftrightarrow \bigvee_{r \in \text{Ext}_P(a)} \text{bt}(r),$$

$$\text{int}(a) \leftrightarrow \bigvee_{r \in \text{Int}_P(a)} [\text{bt}(r) \wedge \bigwedge_{b \in \text{B}^+(r) \cap \text{SCC}(a)} (x_a - 1 \geq x_b)],$$

$$a \rightarrow \text{ext}(a) \vee \text{int}(a), \quad \neg\text{ext}(a) \vee \neg\text{int}(a), \quad \text{ext}(a) \rightarrow (x_a = z).$$

Example

In the context of $P = \{a \leftarrow \textbf{not}\, c. \quad a \leftarrow b. \quad b \leftarrow a. \}$, we obtain:

$$\text{ext}(a) \leftrightarrow \text{bt}(r_1), \quad \text{int}(a) \leftrightarrow \text{bt}(r_2) \wedge (x_a - 1 \geq x_b),$$
$$\text{ext}(b) \leftrightarrow \bot, \quad \text{int}(b) \leftrightarrow \text{bt}(r_3) \wedge (x_b - 1 \geq x_a).$$

Strong Ranking Constraints in Difference Logic

- For an atom $a \in \mathrm{HB}(P)$ and $\mathrm{Def}_P(a) = \mathrm{Ext}_P(a) \sqcup \mathrm{Int}_P(a)$, the local and global strong ranking constraints are

$$\bigwedge_{r \in \mathrm{Int}_P(a)} [\mathrm{bt}(r) \rightarrow \bigvee_{b \in \mathrm{B}^+(r) \cap \mathrm{SCC}(a)} (x_b + 1 \geq x_a)],$$

$$\mathrm{int}(a) \rightarrow \bigvee_{r \in \mathrm{Int}_P(a)} [\mathrm{bt}(r) \wedge \bigvee_{b \in \mathrm{B}^+(r) \cap \mathrm{SCC}(a)} (x_b + 1 = x_a)].$$

Example

Consider again the program $P = \{a \leftarrow \textbf{not}\, c.\ \ a \leftarrow b.\ \ b \leftarrow a.\ \}$.

For the atom $a \in \mathrm{HB}(P)$, the strong ranking constraints are:

$$\mathrm{bt}(r_2) \rightarrow (x_b + 1 \geq x_a),$$
$$\mathrm{int}(a) \rightarrow [\mathrm{bt}(r_2) \wedge (x_b + 1 = x_a)].$$

(Weak) Correspondence of Models

- Ranking constraints (RCs) are compatible—giving rise to
  - $\mathrm{Tr}^{\mathrm{w}}_{\mathrm{DIFF}}(P)$ is the completion $\mathrm{CompN}(P)$ plus weak RCs,
  - $\mathrm{Tr}^{\mathrm{wl}}_{\mathrm{DIFF}}(P)$ extends $\mathrm{Tr}^{\mathrm{w}}_{\mathrm{DIFF}}(P)$ with local strong RCs,
  - $\mathrm{Tr}^{\mathrm{wg}}_{\mathrm{DIFF}}(P)$ extends $\mathrm{Tr}^{\mathrm{w}}_{\mathrm{DIFF}}(P)$ with global strong RCs, and
  - $\mathrm{Tr}^{\mathrm{wlg}}_{\mathrm{DIFF}}(P)$ extends $\mathrm{Tr}^{\mathrm{w}}_{\mathrm{DIFF}}(P)$ with both local and global strong RCs.
- A 1-to-1 correspondence of $\mathrm{AS}(P)$ and $\mathrm{MT}(\mathrm{Tr}^{*}_{\mathrm{DIFF}}(P))$ is impossible due to the properties of difference logic.

Theorem (Niemelä, 2008; T.J. et al., 2009)

Let P be a normal logic program.

1. *If $S \in \mathrm{AS}(P)$, then there is a model $\langle M, \tau \rangle \in \mathrm{MT}(\mathrm{Tr}^{*}_{\mathrm{DIFF}}(P))$ such that $S = M \cap \mathrm{HB}(P)$.*
2. *If $\langle M, \tau \rangle \in \mathrm{MT}(\mathrm{Tr}^{*}_{\mathrm{DIFF}}(P))$, then $S = M \cap \mathrm{HB}(P) \in \mathrm{AS}(P)$.*

Bit-Vector Logic

- Fixed-width bit-vector logic (cf. SMT-LIB format) uses free functional constants x to denote m-bit vectors $x[1 \dots m]$.
- It extends propositional logic with constraints such as

$$t_1 =_m t_2 \text{ and } t_1 <_m t_2$$

  where t_1 and t_2 are well-formed m-bit terms.
- For instance, a bit-vector constraint $t_1 <_m t_2$ is satisfied in an interpretation $\langle I, \tau \rangle$, denoted by $\langle I, \tau \rangle \models t_1 <_m t_2$, iff

$$\tau(t_1) < \tau(t_2).$$

- Other bit-vector primitives are treated similarly.

Example

Consider the theory $T = \{a \rightarrow (x <_2 y),\ b \rightarrow (y <_2 x)\}$.

Weak Ranking Constraints in Bit-Vector Logic

- The external and internal support of an atom $a \in \mathrm{HB}(P)$ can be formalized in analogy to difference logic:

$$\mathrm{ext}(a) \leftrightarrow \bigvee_{r \in \mathrm{Ext}_a(P)} \mathrm{bt}(r),$$

$$\mathrm{int}(a) \leftrightarrow \bigvee_{r \in \mathrm{Int}_a(P)} [\mathrm{bt}(r) \wedge \bigwedge_{b \in \mathrm{B}^+(r) \cap \mathrm{SCC}(a)} (x_b <_m x_a)],$$

$$a \rightarrow \mathrm{ext}(a) \vee \mathrm{int}(a),\ \neg\mathrm{ext}(a) \vee \neg\mathrm{int}(a),\ \mathrm{ext}(a) \rightarrow (x_a =_m \overline{0}).$$

Example

In the context of $P = \{a \leftarrow \mathbf{not}\, c.\quad a \leftarrow b.\quad b \leftarrow a.\}$, we get:

$$\begin{aligned} &\mathrm{ext}(a) \leftrightarrow \mathrm{bt}(r_1), && \mathrm{int}(a) \leftrightarrow \mathrm{bt}(r_2) \wedge (x_b <_2 x_a), \\ &\mathrm{ext}(b) \leftrightarrow \bot, && \mathrm{int}(b) \leftrightarrow \mathrm{bt}(r_1) \wedge (x_a <_2 x_b). \end{aligned}$$

Difference Logic versus Bit-Vector Logic

Translation time/length:

- The translation from ASP to both logics is basically linear.
- Bit-vector solvers such as BOOLECTOR [Brummayer and Biere, 2009] reduce bit vectors into Boolean vectors.
  $\Longrightarrow$ The logarithmic factor of $\mathrm{Tr}_{\mathrm{AT}}(P)$ recurs.

Faithfulness:

- A 1-to-1 correspondence between answer sets and the models of the translation is impossible in difference logic.
- The translations $\mathrm{Tr}^{\mathrm{wl}}_{\mathrm{BV}}(P)$, $\mathrm{Tr}^{\mathrm{wg}}_{\mathrm{BV}}(P)$, and $\mathrm{Tr}^{\mathrm{wlg}}_{\mathrm{BV}}(P)$ are faithful in the strict sense, i.e., $P \equiv_{\mathrm{v}} \mathrm{Tr}^{\mathrm{o}}_{\mathrm{BV}}(P)$ for $\mathrm{o} \in \{\mathrm{wl}, \mathrm{wg}, \mathrm{wlg}\}$.

Extended Rule Types

- The class of weight constraint programs supported by LPARSE and GRINGO is based on atoms of form:

$$l \leq \{b_1, \ldots, b_n, \textbf{not}\, c_1, \ldots, \textbf{not}\, c_m\} \leq u$$

$$l \leq [b_1 = w_{b_1}, \ldots, b_n = w_{b_n}, \textbf{not}\, c_1 = w_{c_1}, \ldots, \textbf{not}\, c_m = w_{c_m}] \leq u$$

- Rules involving such constraints are straightforward to translate into cardinality and weight rules of forms

$$a \leftarrow l \leq \{b_1, \ldots, b_n, \textbf{not}\, c_1, \ldots, \textbf{not}\, c_m\}.$$

$$a \leftarrow l \leq [b_1 = w_{b_1}, \ldots, b_n = w_{b_n} \textbf{not}\, c_1 = w_{c_1}, \ldots, \textbf{not}\, c_m = w_{c_m}].$$

- It is also easy to translate ground weight rules into difference/bit-vector logic as part of $\mathrm{Tr}^*_{\mathrm{DIFF}}/\mathrm{Tr}^*_{\mathrm{BV}}$ translations.

A Native Translation of Weight Constraints

- A weight constraint of form

$$l \leq [b_1 = w_{b_1}, \ldots, b_n = w_{b_n}, \textbf{not}\, c_1 = w_{c_1}, \ldots, \textbf{not}\, c_m = w_{c_m}]$$

can be evaluated with the following case analysis formulas:

$$\begin{array}{ll}
b_1 \rightarrow (s_1 =_k \overline{w_{b_1}}), & \neg b_1 \rightarrow (s_1 =_k \overline{0}), \\
b_2 \rightarrow (s_2 =_k s_1 +_k \overline{w_{b_2}}), & \neg b_2 \rightarrow (s_2 =_k s_1), \\
\vdots & \vdots \\
b_n \rightarrow (s_n =_k s_{n-1} +_k \overline{w_{b_n}}), & \neg b_n \rightarrow (s_n =_k s_{n-1}), \\
c_1 \rightarrow (s_{n+1} =_k s_n), & \neg c_1 \rightarrow (s_{n+1} =_k s_n +_k \overline{w_{c_1}}), \\
\vdots & \vdots \\
c_m \rightarrow (s_{n+m} =_k s_{n+m-1}), & \neg c_m \rightarrow (s_{n+m} =_k s_{n+m-1} +_k \overline{w_{c_m}}).
\end{array}$$

- The formula $\neg(s_{n+m} <_k \overline{l})$ checks the lower bound l.

New Translation from ASP to SAT

1. Remove cardinality and weight rules as well as choice rules under answer-set semantics.
2. Capture answer sets with supported sets.
3. Apply Clark's completion and clausify in Tseitin's style.

Input	Output	Semantics
SMODELS program P	$\mathrm{Normal}(P)$	$\mathrm{AS}(\mathrm{Normal}(P))$
Normal program P	$\mathrm{LP2LP}(P)$	$\mathrm{SuppS}(\mathrm{LP2LP}(P))$
Normal program P	$\mathrm{CompC}(P)$	$\mathrm{CM}(\mathrm{CompC}(P))$

Theorem (T.J. and Niemelä, 2011)

For an SMODELS *program,* $P \equiv_{\mathrm{v}} \mathrm{CompC}(\mathrm{LP2LP}(\mathrm{Normal}(P)))$.

Removing Cardinality Rules

- Eén and Sörensson [2006] translate cardinality constraints into clauses—trying to share structure as far as possible.
- However, in the case of an ASP to SAT translation, the preservation of positive dependencies becomes crucial.

Example

The rule $a \leftarrow 3 \leq \{b_1, b_2, b_3, \textbf{not}\, c_1, \textbf{not}\, c_2\}$ is captured by:

$$
\begin{array}{ccccccc}
a & \longleftarrow & \textbf{cnt}(3,1) & \longleftarrow & \textbf{cnt}(3,2) & \longleftarrow & \textbf{cnt}(3,3) \\
 & & \uparrow b_1 & & \uparrow b_2 & & \uparrow b_3 \\
 & & \textbf{cnt}(2,2) & \longleftarrow & \textbf{cnt}(2,3) & \longleftarrow & \textbf{cnt}(2,4) \\
 & & \uparrow b_2 & & \uparrow b_3 & & \uparrow \textbf{not}\, c_1 \\
 & & \textbf{cnt}(1,3) & \longleftarrow & \textbf{cnt}(1,4) & \longleftarrow & \textbf{cnt}(1,5) \\
 & & \uparrow b_3 & & \uparrow \textbf{not}\, c_1 & & \uparrow \textbf{not}\, c_2
\end{array}
$$

Capturing Answer Sets with Supported Ones

- The syntax of normal logic programs is preserved.
- The shift in semantics is achieved by adding rules which require the existence of a level ranking [Niemelä, 2008].
- The extra rules make Clark's completion sound.

Example

For $P = \{a \leftarrow \textbf{not}\, c. \quad a \leftarrow b. \quad b \leftarrow a.\}$, we introduce:

$$
\begin{aligned}
\mathrm{just}(a) &\leftarrow \textbf{not}\, c. \\
\mathrm{just}(a) &\leftarrow b, \mathrm{lt}(\mathrm{ctr}(b), \mathrm{ctr}(a)). \\
\mathrm{just}(b) &\leftarrow a, \mathrm{lt}(\mathrm{ctr}(a), \mathrm{ctr}(b)). \\
&\leftarrow a, \textbf{not}\, \mathrm{just}(a). \\
&\leftarrow b, \textbf{not}\, \mathrm{just}(b).
\end{aligned}
$$

4. IMPLEMENTATION AND EXPERIMENTS

- The file format of SMODELS system is assumed.
- We have implemented a number of translators:

Translator	Output specification for a program P
LP2NORMAL	Normal(P)
LP2ATOMIC	$\mathrm{Tr}_{\mathrm{AT}}(P)$
LP2LP2	$\mathrm{LP2LP}^{*}(P)$
LP2SAT	CompC(P)
LP2DIFF	$\mathrm{Tr}^{*}_{\mathrm{DIFF}}(P)$
LP2BV	$\mathrm{Tr}^{*}_{\mathrm{BV}}(P)$

- Strong local/global ranking constraints can be included by command line options `-l` and `-g` (when appropriate).

Using The Tools

These tools can be combined in shell pipelines:

```
lparse program.lp \
| lp2normal | lp2lp2 | lp2sat -n | minisat -

lparse program.lp | lp2diff | z3 -smt -m /dev/stdin

lparse program.lp | lp2bv | boolector --smt

gringo program.lp \
| smodels -internal -nolookahead \
| lpcat | lp2normal | igen \
| smodels -internal -nolookahead \
| lpcat -s=symbols.sm \
| lp2lp2 \
| lp2sat -n \
| minisat /dev/stdin model.txt
```

Experiments

- The NP-complete problems from the 2nd ASP Competition:

  15-Puzzle, Blocked *n*-Queens, Channel Routing, Connected Dominating Set, Disjunctive Scheduling, Edge Matching, Fastfood, Generalized Slitherlink, Graph Colouring, Graph Partitioning, Hamiltonian Path, Hanoi, Hierarchical Clustering, Knight Tour, Labyrinth, Maze Generation, Schur Numbers, Sokoban, Solitaire, Sudoku, Travelling Salesperson, Weight Bounded Dominating Set, Wire Routing.
- GRINGO (version 2.0.5) was used to ground all program instances to provide an identical input for all systems.
- The parameters and options of solvers were not tuned.
- All answers sets found were verified using SMODELS 2.34.

Systems Subject to Comparison

Native ASP solvers:

1. CLASP [Gebser et al., 2007]
2. CMODELS [Giunchiglia et al., 2006] calling ZCHAFF

Translation-based ASP solving:

1. LP2ATOMIC+LP2SAT and MINISAT [Eén and Sörensson]
2. LP2LP2+LP2SAT and MINISAT [Eén and Sörensson]
3. LP2DIFF and Z3 [de Moura and Bjørner, 2008]
4. LP2BV and BOOLECTOR [Brummayer and Biere, 2009]

Summary of Results

Number of solved instances (out of 516 possible):

System	W	L	G	LG
CLASP	465			
CMODELS	387			
LP2NORMAL+LP2SAT+MINISAT	387			
LP2DIFF+Z3	360	349	324	324
LP2NORMAL+LP2DIFF+Z3	364	357	349	349
LP2BV+Z3	217	216	194	204
LP2BV+BOOLECTOR	276	244	261	256
LP2NORMAL+LP2BV+BOOLECTOR	381	343	379	381
LP2NORMAL+LP2BV+Z3	346	330	325	331
LP2NORMAL+LP2SAT2+MINISAT	404	429	427	424
LP2NORMAL+CLASP	459			

Based on [Nguyen et al., 2011; T.J. and Niemelä, 2011].

5. LANGUAGE INTEGRATION

- Non-Boolean variables are important primitives in logical modeling in a number of disciplines: ASP, CP, LP, MIP, ...
- The SMT framework enriches Boolean satisfiability checking in terms of a background theory.
- Logic programs under answer sets can be translated into
  - difference logic [Niemelä, 2008],
  - bit-vector logic [Nguyen et al., 2011], and
  - mixed integer programming [Liu et al., 2012].
- Translations in the other direction are impeded if infinite-domain variables are involved.
- There are approaches combining ASP and CP [Balduccini, 2009; Gebser et al., 2009; Mellarkord et al., 2008].

Objectives for the Integration

- Our goal is to integrate ASP and SMT so that non-Boolean variables of these formalisms can be used together.
- We aim at a rule-based language ASP(SMT) which is enriched by theory atoms from a particular SMT dialect.

Example

Let us formalize the n-queens problem in ASP(DL):

$$\text{queen}(1..n). \qquad \text{int}(row(X)) \leftarrow \text{queen}(X). \qquad \text{int}(zero).$$

$$row(X) - zero > 0 \leftarrow \text{queen}(X).$$
$$row(X) - zero \leq n \leftarrow \text{queen}(X).$$
$$\leftarrow row(X) - row(Y) = 0, \text{queen}(X), \text{queen}(Y), X < Y.$$
$$\leftarrow row(X) - row(Y) = |X - Y|, \text{queen}(X), \text{queen}(Y), X < Y.$$

Integrated Language: Syntax

- A program P in ASP(SMT) is a finite set of rules of forms

$$a \leftarrow \; b_1, \; \ldots, \; b_m, \; \textbf{not}\, c_1, \; \ldots, \; \textbf{not}\, c_n, \; t_1, \; \ldots, \; t_l$$
$$t \leftarrow \; b_1, \; \ldots, \; b_m, \; \textbf{not}\, c_1, \; \ldots, \; \textbf{not}\, c_n, \; t_1, \; \ldots, \; t_l$$

  where

  - $a, b_1, \ldots, b_m$, and $c_1, \ldots, c_n$ are propositional atoms, and
  - $t_1, \ldots, t_l$ are theory atoms of the SMT fragment.

- The latter form is viewed as a shorthand for a constraint

$$\leftarrow b_1, \; \ldots, \; b_m, \; \textbf{not}\, c_1, \; \ldots, \; \textbf{not}\, c_n, \; t_1, \; \ldots, \; t_l, \; \neg t$$

  where $\neg t$ denotes the negation/complement of t.

- For instance, we have $\neg(x - y < 6) = (x - y \geq 6)$.

Integrated Language: Semantics

- The theory base of an ASP(SMT) program P consists of theory atoms that appear in the rules of P.
- An interpretation of an ASP(SMT) program P is defined as a pair $\langle S, T \rangle$ where $S \subseteq \mathrm{HB}(P)$ and $T \subseteq \mathrm{TB}(P)$.

Definition

An interpretation $\langle S, T \rangle$ is an *answer set* of P iff

1. $\langle S, T \rangle \models P$,
2. the propositional part S is the least subset closed under

$$P^M = \{\mathrm{H}(r) \leftarrow \mathrm{B}^+(r) \mid$$
$$r \in P,\ \mathrm{B}^-(r) \cap S = \emptyset,\ \text{and } \mathrm{B}^{\mathrm{t}}(r) \subseteq T\}, \text{ and}$$

3. the theory part $T \cup \overline{T}$ where $\overline{T} = \{\neg t \mid t \in \mathrm{TB}(P) \setminus T\}$ is satisfiable in the SMT fragment in question.

Example

Consider an ASP(DL) program P

$$\leftarrow \textbf{not}\, s. \quad s \leftarrow x > z. \quad p \leftarrow x \leq y. \quad p \leftarrow q. \quad q \leftarrow p, y \leq z.$$

and the following candidates that superficially satisfy P:

S_i	T_i	$\overline{T_i}$	SAT?
$\{s\}$	$\{x > z\}$	$\{x > y, y > z\}$	Yes
$\{s, p, q\}$	$\{x > z, x \leq y, y \leq z\}$	$\emptyset$	No
$\{s, p, q\}$	$\{x > z, y \leq z\}$	$\{x > y\}$	Yes

$P^{\langle S_i, T_i \rangle}$	$\mathrm{cl}(P^{\langle S_i, T_i \rangle})$	Stable?
$\{s.\ p \leftarrow q.\ \}$	$\{s\}$	Yes
$\{s.\ p.\ p \leftarrow q.\ q \leftarrow p.\ \}$	$\{s, p, q\}$	Yes
$\{s.\ p \leftarrow q.\ q \leftarrow p.\ \}$	$\{s\}$	No

$\Longrightarrow$ The pair $\langle \{a\}, \{x > z\} \rangle$ is the only answer set!

ASP versus ASP(DL)

- In pure ASP encodings, variables appearing in a rule are instantiated over the Herbrand universe of the program.
- The number of instances can be reduced by treating some variables as integer variables in difference logic.
- If a rule involves n variables ranging over a set D of integers, savings up to a factor of $|D|^n$ can be possible.

Example

Compare the two constraints below in this respect:

$$\leftarrow \text{start}(P, T_1), \text{end}(P, T_2), T_2 - T_1 < D, \text{process}(P, D), \text{time}(T_1), \text{time}(T_2).$$

$$\leftarrow e(P) - s(P) < D, \text{process}(P, D).$$

where $e(P)$ and $s(P)$ are integer variables associated with P.

Example: A Scheduling Problem

- A predicate read(P, N, T) is used to encode the time T required by a person P to read a newspaper N.
- Integer variables $s(P, N)$ and $e(P, N)$ capture the respective starting and ending times.

$$
\begin{aligned}
s(P, N) \geq 0 &\leftarrow \text{read}(P, N, T). \\
e(P, N) - s(P, N) = T &\leftarrow \text{read}(P, N, T). \\
e(P, N) \leq \mathit{deadline} &\leftarrow \text{read}(P, N, T).
\end{aligned}
$$

$$
\begin{aligned}
\leftarrow\ & \text{s}(P, N_1) < \text{s}(P, N_2),\ \text{s}(P, N_2) - \text{s}(P, N_1) < T_1, \\
& \text{read}(P, N_1, T_1),\ \text{read}(P, N_2, T_2),\ N_1 \neq N_2.
\end{aligned}
$$

$$
\begin{aligned}
\leftarrow\ & s(P_1, N) < s(P_2, N),\ s(P_2, N) - s(P_1, N) < T_1, \\
& \text{read}(P_1, N, T_1),\ \text{read}(P_2, N, T_2),\ P_1 \neq P_2.
\end{aligned}
$$

Prototype Implementation

- Theory atoms are represented with special predicates like

$$\text{dl\_lt}(X, Y, D)$$

  for a constraint $x - y < d$ in difference logic.
- Special domain predicates such as $\text{int}(V)$ for DL are used to declare the domains of theory constants.
- Our prototype exploits off-the-shelf ASP and SMT components for grounding (GRINGO) and model search.

Example

$$\text{int}(at(X)) \leftarrow \text{edge}(X, Y, W).$$

$$\text{int}(at(Y)) \leftarrow \text{edge}(X, Y, W).$$

$$\leftarrow \text{route}(X, Y), \text{edge}(X, Y, W), \text{dl\_lt}(at(Y), at(X), W).$$

Performance in the Newspaper Benchmark

Deadline	DINGO		CLINGO	
	time	size ratio	time	size ratio
100	0.09	1.0	2.10	1.0
200	0.11	1.1	9.00	3.1
300	0.11	1.3	21.32	6.3
400	0.10	1.4	36.68	15
500	0.12	1.5	61.15	23
600	0.12	1.7	93.51	34
700	0.11	1.8	–	44
800	0.11	1.9	–	60
900	0.12	2.1	–	74
1000	0.13	2.2	–	81

6. CONCLUSIONS

SAT and SMT for Answer Set Programming

- SAT/SMT solvers develop rapidly—providing a promising computational platform to implement ASP systems.
- The functionality of SMODELS-compatible solvers can be implemented using
  1. a compact translation of a cardinality/weight constraint program into an appropriate theory and
  2. a suitable SAT/SMT solver for model search.
- The performance obtained in this way is surprisingly close to that of the top state-of-the-art ASP solver CLASP.
- Tools LP2LP2, LP2SAT, LP2DIFF, and LP2BV implement the required translations of SMODELS programs into SAT/SMT.

Conclusions

Answer Set Programming for SAT and SMT

- Our translators provide an easy way to generate challenging, highly structural or partly randomized, benchmark instances.
- The integrated the languages ASP(SMT) enrich rules with extra conditions—enabling more concise modeling.
- Our approach enables the use of standard ASP grounders for the creation of SMT theories of interest declaratively.
- Our first experiments using these encodings also show reduced solving times in certain problem domains.
- It is also possible to develop ASP(SMT) encodings in a modular way using LPCAT for linking.

Ongoing/Future Work

- There are further ways to optimize the translation-based approach from ASP to SAT and its extensions:
  - Simplification of the rule-based and clausal representations.
  - Trying out the new (versions of) SAT/SMT solvers.
  - Proper parametrization of the tools involved.
  - Linear transformations are possible for SMT solvers.
- We are developing new translations into further formalisms such as mixed integer programming [Liu et al., 2012].
- Also, new ways to extend rules are of interest.
- We plan to participate in the 4th ASP Competition in 2013.
- Submission of ASP-based benchmark sets to future SAT/SMT competitions.

REFERENCES (ASP)

C. Baral: *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge Univ. Press, 2003.

M. Gelfond and N. Leone: *Logic programming and knowledge representation – The A-Prolog perspective.* Artificial Intelligence, 138(1-2), 3–38, 2002.

G. Brewka, T. Eiter, and M. Truszczyński: *Answer set programming at a glance.* CACM, 54(12), 92–103, 2011.

I. Niemelä: *Logic Programming with Stable Model Semantics as a Constraint Programming Paradigm.* Annals of Mathematics and Artificial Intelligence, 25(3-4), 241–273, 1999.

V. Marek and M. Truszczyński: *Stable models and an alternative logic programming paradigm.* In Logic Programming Paradigm: A 25-Year Perspective, 375–398, 1999.

REFERENCES (Grounders)

M. Gebser, T. Schaub, and S. Thiele: *GrinGo : A New Grounder for Answer Set Programming.* In Proceedings of LPNMR'07, 266–271, 2007.

S. Perri, F. Scarcello, G. Catalano, and N. Leone: *Enhancing DLV instantiator by backjumping techniques.* Annals of Mathematics and Artificial Intelligence, 51(2-4), 195–228, 2007.

T. Syrjänen: *Omega-Restricted Logic Programs.* In Proceedings of LPNMR'01, 267–279, 2001.

REFERENCES (Solvers)

R. Brummayer and A. Biere. *Boolector: An efficient SMT solver for bitvectors and arrays.* In Proceedings of TACAS'09, 174–177, 2009.

M. Gebser, B. Kaufmann, A. Neumann and, T. Schaub: CLASP *: A Conflict-Driven Answer Set Solver.* In Proceedings of LPNMR'07, 260–265, 2007.

L. de Moura and N. Bjørner. *Z3: An efficient SMT solver.* In Proceedings of TACAS'08, 337–340, 2008.

E. Giunchiglia, Y. Lierler, M. Maratea: *Answer Set Programming Based on Propositional Satisfiability.* Journal of Automated Reasoning 36(4), 345–377, 2006.

F. Lin and Y. Zhao: *ASSAT: Computing answer sets of a logic program by SAT solvers.* Artificial Intelligence, 157(1-2), 115–137, 2004.

REFERENCES (Translations)

R. Ben-Eliyahu and R. Dechter: *Propositional Semantics for Disjunctive Logic Programs.* Annals of Mathematics and AI, 12(1-2), 53–87, 1994.

N. Eén and N. Sörensson: *Translating Pseudo-Boolean Constraints into SAT.* Journal on Satisfiability, Boolean Modeling and Computation, 2(1-4), 1–26, 2006.

T. Janhunen: *Representing normal programs with clauses.* In Proceedings of ECAI'04, 358–362, 2004.

T. Janhunen: *Some (in)translatability results for normal logic programs and propositional theories.* Journal of Applied Non-Classical Logics, 16(1-2):35–86, June 2006.

T. Janhunen and I. Niemelä: *Compact Translations of Non-Disjunctive Answer Set Programs to Propositional Clauses.* In Gelfond Festschrift, 111–130, 2011.

REFERENCES (Translations)

T. Janhunen, I. Niemelä, and M. Sevalnev: *Computing stable models via reductions to difference logic.* In Proceedings of LPNMR'09, pages 142–154, 2009.

F. Lin, J. Zhao: *On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic.* In Proceedings of IJCAI'03, 853–858, 2003.

G. Liu, T. Janhunen, and Ilkka Niemelä: *Answer Set Programming via Mixed Integer Programming.* In Proceedings of KR'12, 32–42, 2012.

M. Nguyen, T. Janhunen, and I. Niemelä: *Translating Answer-Set Programs into Bit-Vector Logic.* In Proceedings of INAP'11, 105–116, 2011.

I. Niemelä: *Stable models and difference logic.* Annals of Mathematics and AI, 53(1-4):313–329, 2008.

REFERENCES (General)

K. Apt and H. Blair and A. Walker: *Towards a theory of declarative knowledge.* In Foundations of Deductive Databases, Chapter 2, Morgan Kaufmann, 1988.

M. Balduccini: *Industrial-Size Scheduling with ASP+CP.* In Proceedings of LPNMR'11, 284–296, 2011.

K. Clark: *Negation as failure.* In Logics and Databases, 293–322, Plenum Press, 1978.

M. Gebser, M. Ostrowski, and T. Schaub: *Constraint Answer Set Solving.* In Proceedings of ICLP'09, 235–249, 2009.

M. Gelfond and V. Lifschitz: *The Stable Model Semantics for Logic Programming.* In Proc. of ICLP'88, 1070–1080, 1988.

V. Lifschitz, A. Razborov: *Why are there so many loop formulas?* ACM TOCL 7(2), 261–268, 2006.

REFERENCES (General)

V. Marek and V. S. Subrahmanian: *The Relationship between Stable, Supported, Default and Autoepistemic Semantics for General Logic Programs.*
Theoretical Computer Science, 103, 365–386, 1992.

V. Mellarkod, M. Gelfond, and Y. Zhang: *Integrating Answer Set Programming and Constraint Logic Programming.*
Annals of Mathematics and AI, 53(1-4), 251–287, 2008.

R. Nieuwenhuis and A. Oliveras: *DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic.*
In Proceedings of CAV’05, 321–334, 2005.

P. Simons: *Extending the Stable Model Semantics with More Expressive Rules.*
In Proceedings of LPNMR’99, 305–316, 1999.