Aalto University
School of Science

Integrating Answer Set Programming and Satisfiability Modulo Theories

Ilkka Niemelä

Department of Information and Computer Science
Aalto University
Ilkka.Niemela@aalto.fi
(Joint work with Tomi Janhunen)

Deduction at Scale, Schloss Ringberg, March 2011

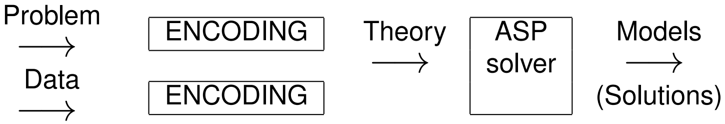
Answer Set Programming (ASP)

- Term coined by Vladimir Lifschitz in the late 1990s
- An approach to modeling and solving **knowledge intensive search problems** with defaults, exceptions, definitions:
  planning, configuration, model checking, network management, linguistics, bioinformatics, combinatorics, ...
- Solving a problem in ASP:
  Encode the problem as a logic program such that **solutions** to the problem are given by stable models (**answer sets**) of the program.

Problem $\longrightarrow$ $\boxed{\text{ENCODING}}$
Data $\longrightarrow$ $\boxed{\text{ENCODING}}$
Program $\longrightarrow$ $\boxed{\text{ASP solver}}$ Models $\longrightarrow$ (Solutions)

ASP

- Expressive KR language for problem encoding
- Uniform encoding:
  separate problem specification and problem instance data
- Compact, easily maintainable representation
- Program development, reusable modules, debugging, testing, optimization
- Integrating KR, DDB, and search techniques
- Handling dynamic, knowledge intensive applications:
  data, frame axioms, exceptions, defaults, definitions



Coloring Problem

Uniform Encoding: the problem specification is a fixed program working for every input graph given as a set of facts.

```
% Problem specification
1 { colored(V,C):color(C) } 1 :- vtx(V).
:- edge(V,U), colored(V,C), colored(U,C).

% Data
vtx(a). ...
edge(a,b). ...
color(r). color(g). ...
```

☞ Legal colorings of the graph given as data and stable models of the problem encoding and data correspond: a vertex v colored with a color c iff `colored`(v, c) holds in a stable model.

ASP Solver Technology

- ASP solvers need to handle two challenging tasks: complex data and search
- Current systems employ a two level architecture with two steps:
- **Grounding** step handles complex data:
  - Given program *P* with variables, generate a set of ground instances of the rules preserving stable models.
  - LP and DDB techniques employed
- **Model search** for ground programs

☞ **ASP = KR + DDB + Search**

Integrating ASP and SMT

- Solvers for the **propositional satisfiability problem (SAT)** are used widely as model search engines.
- Extensions of SAT emerging:
  Satisfiability Modulo Theories (SMT)
- Efficient SMT solvers for expressive theories (integers, reals, uninterpreted function with equality, bit vectors, arrays, ...) are becoming available
- Is it possible to integrate ASP and SMT to exploit the strengths of both approaches?

Integrating ASP and SMT

Two interrelated lines of work:

- Using **SMT solvers as model search engines** for ground programs
- **Combining** ASP and SMT modelling **languages**

Outline

- Stable models and propositional satisfiability
- Stable models and linear constraints
- Satisfiability Modulo Theories
- Translating LPs to SMT
- Integrating ASP and SMT

Stable models

- For a logic program consisting of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n.$$

  a **stable model** is a set of atoms (i) satisfying the rules where (ii) each atom is **justified** by the rules (negation by default).

- **Example.** P: $b \leftarrow .$
  $f \leftarrow b, \text{not } eb.$
  $eb \leftarrow p.$

- $\{b, eb\}$ is not a stable model of P but
  $\{b, f\}$ is the (unique) stable model of P.

- For rules with variables **Herbrand interpretation** used (UNA, DCA):
  stable models of a set of rules are defined to be those of the Herbrand instantiation of the rules.

Stable Models and SAT

- LPs with stable models are closely related to SAT through program completion.
  Example.

P :	Completion $\mathrm{CC}(P)$:
$a \leftarrow b, \mathrm{not}\ c$	$(a \leftrightarrow ((b \wedge \neg c) \vee (\neg b \wedge d))) \wedge$
$a \leftarrow \mathrm{not}\ b, d$	$\neg b \wedge \neg c \wedge \neg d$

- For **tight programs** (no positive recursion) **models of the completion and stable models** coincide (Fages 1994).
- SAT solvers provide an interesting platform for implementing ASP solvers.

Stable Models and SAT

- However, translating general (non-tight) LPs to SAT is challenging
  - Modular translation not possible (I.N. 1999)
  - Without new atoms exponential blow-up (Lifschitz & Razborov 2006)
- There are one pass translations:
  - Polynomial size (Ben-Eliyahu & Dechter 1994; Lin & Zhao 2003)
  - $O(\|P\| \times \log|At(P)|)$ size (Janhunen 2004)
- Also incremental translations have been developed extending the completion dynamically with **loop formulas** (Lin & Zhao 2002)

  ☞ ASSAT and CMODELS ASP solvers

Stable Models and SAT

- Question: what needs to be added to SAT to allow a compact linear size translation of LPs to SAT?
- A possibility: stable models can be characterized using **orderings** (Elkan 1990; Fages 1994).
- Such an ordering can be captured with a restricted set of linear constraints on integers using **level rankings** (I.N. AMAI 2008)
- A suitable simple extension of propositional logic with such restricted linear constraints called **difference logic** is supported by most SMT solvers.

Stable Models and Linear Constraints

- A **level ranking** of a model M is a function assigning positive integers to atoms such that for each atom $a \in M$ there is supporting rule with (i) a as the head, (ii) body true in M and (iii) for each **positive body** atom the **level ranking is smaller** than that of a.
- **Example.** Consider a program P

$p_1 \leftarrow .$
$p_2 \leftarrow p_1.$
$p_3 \leftarrow p_1. \quad p_3 \leftarrow p_4.$
$p_4 \leftarrow p_2. \quad p_4 \leftarrow p_3.$

Function $\mathrm{lr}_1(p_i) = i$ is a level ranking of $M = \{p_1, p_2, p_3, p_4\}$

Theorem (I.N, AMAI 2008)

Let M be a model of the completion of a ground program P. Then M is a stable model of P iff there is a level ranking of M for P.

Satisfiability Modulo Theories

- **Satisfiability Modulo Theories (SMT) problem**: a first-order theory T is given and the problem is to determine whether a formula **F is T-satisfiable** (whether $T \wedge F$ is satisfiable in the usual first-order sense).
- Some restrictions are typically assumed:
  - F is a **ground** (quantifier-free) formula that can contain **free constants** not in the signature of T but all other predicate and function symbols are in the signature of T.
  - **T-satisfiability of a conjunction** of such **ground literals** is **decidable**.

Example: Difference Logic

- T is the theory of integers
- F is limited to contain only linear difference constraints of the form

$$x_i + k \geq x_j \text{ (or equivalently } x_j - x_i \leq k\text{)}$$

  where k is an arbitrary integer constant and $x_i, x_j \in \mathcal{X}$ are free constants (which can be seen as integer valued variables).
- **Difference logic = propositional logic + linear difference constraint**
- For example,

$$(x_1 + 2 \geq x_2) \leftrightarrow (p_1 \rightarrow \neg(x_2 - 3 \geq x_1))$$

  is a formula in difference logic where $2, 3$ are integer constants, x_1, x_2 **free function constants**, and p_1 **a free predicate constant**.

Translating LPs to Difference Logic

- Mapping $T_{\mathrm{diff}}(P)$ of a logic program P to difference logic consists of two parts:
  - completion $\mathrm{CC}(P)$ of P and
  - **ranking constraints** $\mathrm{R}(P)$.
- The completion $\mathrm{CC}(P)$ contains for each atom a having k rules in P, the formula

$$a \leftrightarrow bd_a^1 \vee \cdots \vee bd_a^k$$

and for each such rule a formula

$$bd_a^i \leftrightarrow b_1 \wedge \cdots \wedge b_m \wedge \neg c_1 \wedge \cdots \wedge \neg c_n$$

Ranking Constraints

- The ranking constraints R(P) contain for each atom a which has $k \geq 1$ rules in P, a formula in difference logic

$$a \rightarrow \bigvee_{i=1}^{k} (bd_a^i \wedge (x_a - 1 \geq x_{b_1}) \wedge \cdots \wedge (x_a - 1 \geq x_{b_m}))$$

  where x_a, x_{b_i} are free function constants denoting the rankings of atoms a, b_i.

Example.

P:	CC(P):	R(P):
$p \leftarrow q, \text{not } r.$	$\neg r$	$p \rightarrow (bd_p^1 \wedge (x_p - 1 \geq x_q))$
$q \leftarrow p, \text{not } r.$	$p \leftrightarrow bd_p^1$	$q \rightarrow (bd_q^1 \wedge (x_q - 1 \geq x_p)).$
	$bd_p^1 \leftrightarrow q \wedge \neg r$	
	$q \leftrightarrow bd_q^1$	
	$bd_q^1 \leftrightarrow p \wedge \neg r$	

Difference Logic Captures Stable Models

Theorem (I.N., AMAI 2008)

- *If a set of atoms M is a stable model of a finite normal program P, then there is a satisfying valuation τ of $\mathrm{T}_{\mathrm{diff}}(P)$ such that $M = \{a \in \mathrm{At}(P) \mid \tau(a) = \top\}$.*
- *If there is a satisfying valuation τ of $\mathrm{T}_{\mathrm{diff}}(P)$, then $M = \{a \in \mathrm{At}(P) \mid \tau(a) = \top\}$ is a stable model of P.*

☞ A solver for difference logic can be used for computing stable models.

Observations

- The translation is compact (of linear size).
- It uses a limited subset of difference logic:
  - Level rankings can be captured with constraints of the form $x_i - 1 \geq x_j$
- The translation can be made even more compact and the number of required linear constraints can be reduced dramatically in typical cases by exploiting **strongly connected components** given by the **positive dependency graph** of the program (I.N., AMAI 2008).
- The translation provides a **rich source of benchmarks** for difference logic solvers (wide range of ASP applications, for example, in the ASP competitions).

Experiments

- A translator from ground programs to difference logic which supports a number of variants of the translation available (Janhunen & I.N. & Sevalnev, LPNMR 2009). `http://www.tcs.hut.fi/Software/lp2diff/`
- Any state-of-the-art SMT solver supporting difference logic can be used without modification as the backend solver.
  ☞ The performance obtained by current SMT solvers (Z3, BARCELOGIC, YICES) surprisingly close to the best native ASP solvers (clasp).
  ☞ The same (or slightly better performance) is obtained by carefully bounding the integers, translating the linear constraints to CNF and then using state-of-the-art SAT solvers.

Integrating ASP and SMT

- Goal: combining KR and DDB modelling language features with rich theories offered by SMT solvers.
- A direct approach: rules of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n, t_1, \ldots, t_l.$$

  where $t_1, \ldots, t_l$ are theory literals
- Semantics combines
  - Hebrand interpretation for rules with variables (UNA, DCA)
  - "classical" interpretation for theory atoms

Example: Routing with Real Time Constraints

```
% Data
vtx(a). ... edge(a,b,10.8). ... critical(c,122.5). ...

% Problem specification
{ route(X,Y) } :- edge(X,Y,T).
:- 2 { route(X,Y):edge(X,Y,T) }, vtx(X).
:- 2 { route(X,Y):edge(X,Y,T) }, vtx(Y).
:- route(X,Y), edge(X,Y,T), at(Y) - at(X) < T.
:- vtx(X), at(X) < 0.
r(start).
r(Y) :- r(X), route(X,Y).
missing_critical :- critical(X,T), not r(X).
missing_critical :- critical(X,T), r(X), at(X) > T.
:- missing_critical.
```

Integrating ASP and SMT

- The approach can be implemented by combining
  - ASP grounding techniques
  - the proposed translation to difference logic and
  - an SMT solver supporting difference logic as the model search engine
- For example,

  ```
  :- route(X,Y), edge(X,Y,T), at(Y) - at(X) < T.
  ```

  can be seen as a shorthand for a set of ground rules

  ```
  :- route(s1,s2), edge(s1,s2,s3), at(s2) - at(s1) < s3.
  ```

  where $s1, s2, s3$ range over Herbrand terms and $at(s1), at(s2), s_3$ are treated as **free constants** of the background theory.
- The grounder computes a sufficient set of such ground instances.

Integrating ASP and SMT

- Now the proposed translation can be used to map ground rules to SMT with the following extension in the completion:
- For a rule r of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n, t_1, \ldots, t_l.$$

the formula capturing the satisfaction of the body is now

$$bd_a^i \leftrightarrow b_1 \wedge \cdots \wedge b_m \wedge \neg c_1 \wedge \cdots \wedge \neg c_n \wedge t_1 \wedge \cdots \wedge t_l$$

- Then any SMT solver supporting difference logic + the theory used in the theory literals $t_1, \ldots, t_l$ can be used as the **model search engine**.

Conclusions

- Difference logic allows for a compact translation of rules capturing stable models.
- The translation to difference logic opens up the possibility of using difference logic solvers as a computational platform for implementing ASP.
- The performance obtained by the translation and current SMT solvers is already surprisingly close to the best state-of-the-art ASP solvers.
- The translation provides a basis for combining ASP and SMT modelling languages

References

- I.N. Stable Models and Difference Logic. Annals of Mathematics and Artificial Intelligence 1-4 (2008) 53, 313-329.
- T. Janhunen, I.N., M. Sevalnev. Computing Stable Models via Reductions to Difference Logic. LPNMR 2009.
- T. Janhunen and I.N. Compact Translations of Non-Disjunctive Answer Set Programs to Propositional Clauses. In M. Balduccini and T. C. Son, editors, in the Proceedings of Symposium on Constructive Mathematics in Computer Science, Organized in Honour of Michael Gelfond's 65th Anniversary, Lexington, Kentucky, October 2010.