
Answer Set Programming

an Approach to Declarative Problem Solving

Ilkka Niemelä

`Ilkka.Niemela@hut.fi`, `http://www.tcs.hut.fi/~ini/`

Laboratory for Theoretical Computer Science
Helsinki University of Technology



Contents

- Introduction to Answer Set Programming (ASP)
- ASP with logic programs
- Implementation techniques
- Available systems
- Applications



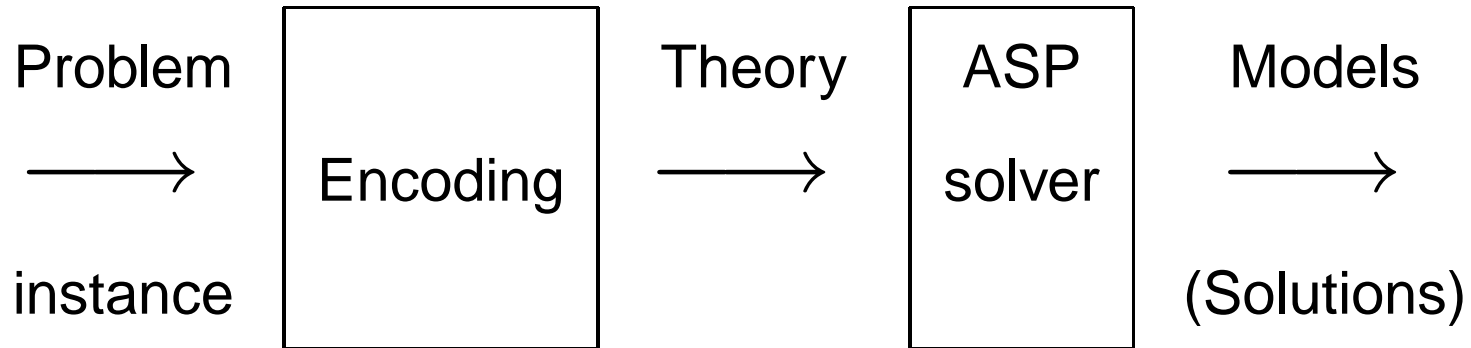
Answer Set Programming

- Term coined by Vladimir Lifschitz
- Roots: KR, logic programming, nonmonotonic reasoning
- Based on some formal system with semantics that assigns a theory a collection of answer sets (models).
- An **ASP solver**: computes answer sets for a theory
- Solving a problem in ASP:
Encode the problem as a theory such that **solutions** to the problem are given by the **answer sets** of the theory.



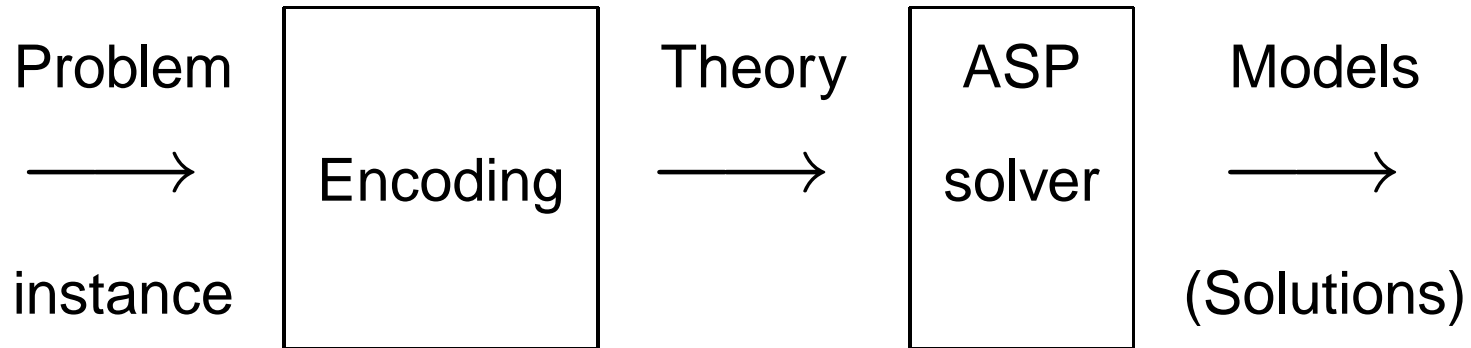
ASP—cont'd

- Solving a problem using ASP



ASP—cont'd

- Solving a problem using ASP



- Possible formal system Models

Propositional logic

Truth assignments

CSP

Variable assignments

Logic programs

Stable models



Example. k -coloring problem

- Given a graph (V, E) find an assignment of one of k colors to each vertex such that no two adjacent vertices share a color.
- Encoding 3-coloring using propositional logic

For each vertex $v \in V$:

$$v(1) \vee v(2) \vee v(3)$$

$$\neg v(1) \vee \neg v(2)$$

$$\neg v(1) \vee \neg v(3)$$

$$\neg v(2) \vee \neg v(3)$$

For each edge $(v, u) \in E$:

$$\neg v(1) \vee \neg u(1)$$

$$\neg v(2) \vee \neg u(2)$$

$$\neg v(3) \vee \neg u(3)$$



Example. k -coloring problem

- Given a graph (V, E) find an assignment of one of k colors to each vertex such that no two adjacent vertices share a color.

- Encoding 3-coloring using propositional logic

For each vertex $v \in V$:

$$v(1) \vee v(2) \vee v(3)$$

$$\neg v(1) \vee \neg v(2)$$

$$\neg v(1) \vee \neg v(3)$$

$$\neg v(2) \vee \neg v(3)$$

For each edge $(v, u) \in E$:

$$\neg v(1) \vee \neg u(1)$$

$$\neg v(2) \vee \neg u(2)$$

$$\neg v(3) \vee \neg u(3)$$

- 3-colorings of a graph (V, E) and models of the encoding correspond:

vertex v colored with color i iff $v(i)$ true in the model.



What is ASP Good for?

Search problems:

- Constraint satisfaction
- Planning, routing
- Computer-aided verification
- Security analysis
- Product configuration
- Combinatorics
- Diagnosis

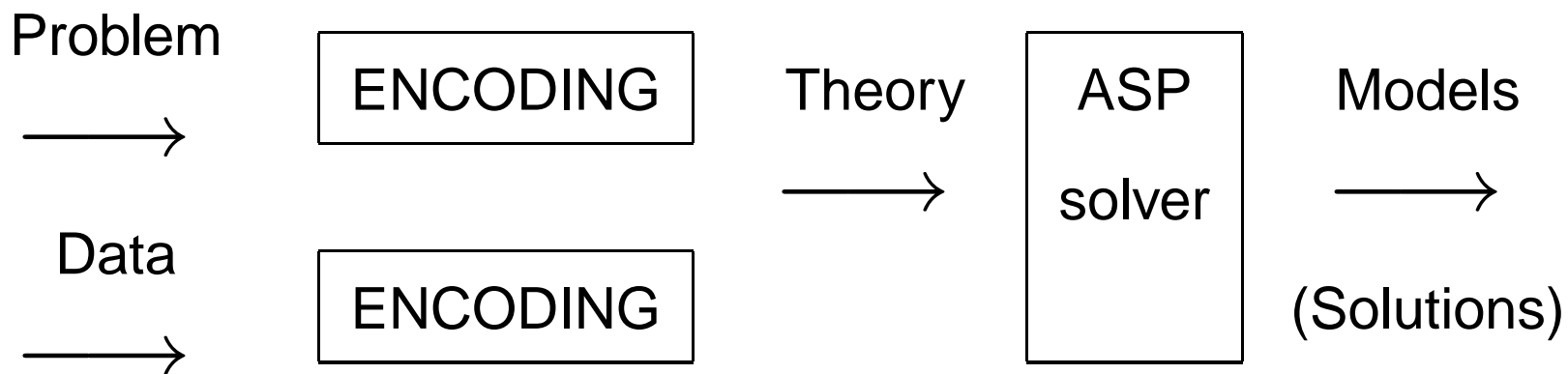


Declarative problem solving



Towards ASP in Practice

- Uniform encoding:
separate problem specification and data
- Compact, easily maintainable representation
- Integrating KR, DB, and search techniques
- Handling dynamic, knowledge intensive applications:
data, frame axioms, exceptions, defaults, closures



ASP Using Logic Programs



ASP Using Logic Programs

- Logic programming: framework for merging KR, DB, and search
- PROLOG style logic programming systems not directly suitable for ASP:
 - they search for proofs (not models) and produce answer substitutions
 - they are not entirely declarative
- In late 80s new semantical basis for “negation-as-failure” in LPs based on nonmonotonic logics: **Stable model semantics**
- Implementations of stable model semantics led to ASP



Example. 3-coloring

Problem: $clrd(V, 1) \leftarrow \text{not } clrd(V, 2), \text{not } clrd(V, 3), vtx(V)$
 $clrd(V, 2) \leftarrow \text{not } clrd(V, 1), \text{not } clrd(V, 3), vtx(V)$
 $clrd(V, 3) \leftarrow \text{not } clrd(V, 1), \text{not } clrd(V, 2), vtx(V)$
 $\leftarrow edge(V, U), clrd(V, C), clrd(U, C)$


Data: $vtx(v) \quad vtx(u) \quad \dots$
 $edge(v, u) \quad edge(u, w) \quad \dots$



Example. 3-coloring

Problem: $clrd(V, 1) \leftarrow \text{not } clrd(V, 2), \text{not } clrd(V, 3), vtx(V)$
 $clrd(V, 2) \leftarrow \text{not } clrd(V, 1), \text{not } clrd(V, 3), vtx(V)$
 $clrd(V, 3) \leftarrow \text{not } clrd(V, 1), \text{not } clrd(V, 2), vtx(V)$
 $\leftarrow edge(V, U), clrd(V, C), clrd(U, C)$

Data: $vtx(v) \quad vtx(u) \quad \dots$
 $edge(v, u) \quad edge(u, w) \quad \dots$

 3-colorings and stable models of the encoding correspond: v colored i iff $clrd(v, i)$ in the model.



LPs with Stable Models Semantics

- Consider normal logic program rules

$$A \leftarrow B_1, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n$$

- Seen as constraints on an answer set (stable model):
 - if B_1, \dots, B_m are in the set and
 - none of C_1, \dots, C_n is included,then A must be included in the set
- A stable model is a set of atoms
 - (i) which satisfies the rules and
 - (ii) where each atom is **justified** by the rules.



Stable Models — cont'd

■ Program:

$b \leftarrow$

$f \leftarrow b, \text{ not } eb$

$eb \leftarrow p$

Stable model:

$\{b, f\}$



Stable Models — cont'd

■ Program:

$b \leftarrow$

$f \leftarrow b, \text{ not } eb$

$eb \leftarrow p$

Stable model:

$\{b, f\}$

■ Another candidate model: $\{b, eb\}$

satisfies the rules but is not a proper stable model:
 eb is included for no reason.



Stable Models — cont'd

■ Program:

$b \leftarrow$

$f \leftarrow b, \text{ not } eb$

$eb \leftarrow p$

Stable model:

$\{b, f\}$

■ Another candidate model: $\{b, eb\}$

satisfies the rules but is not a proper stable model:
 eb is included for no reason.

■ Justifiability of stable models is captured by the notion of a **reduct** of a program



The stable model semantics
[Gelfond/Lifschitz, 1988].



Example. Stable models

- A program can have **none**, one, or **multiple** stable models.

- Program:

$$p_1 \leftarrow \text{not } q_1$$

$$q_1 \leftarrow \text{not } p_1$$

Stable models:

$$\{p_1\}$$

$$\{q_1\}$$

- Program:

$$p_1 \leftarrow \text{not } q_1$$

$$q_1 \leftarrow \text{not } p_1$$

$$\leftarrow \text{not } p_1$$

$$\leftarrow \text{not } q_1$$

Stable models:

None



Variables

- Variables are needed for uniform encodings

Program:

$$clrd(V, 1) \leftarrow \text{not } clrd(V, 2), \text{not } clrd(V, 3), vtx(V)$$
$$clrd(V, 2) \leftarrow \text{not } clrd(V, 1), \text{not } clrd(V, 3), vtx(V)$$
$$clrd(V, 3) \leftarrow \text{not } clrd(V, 1), \text{not } clrd(V, 2), vtx(V)$$
$$\leftarrow edge(V, U), clrd(V, C), clrd(U, C)$$

Data:

$$vtx(v) \quad vtx(u) \quad \dots$$
$$edge(v, u) \quad edge(u, w) \quad \dots$$


Variables — cont'd

- Semantics: Herbrand models
- A rule is seen as a shorthand for the set of its ground instantiations.

Example.

$$\text{clrd}(V, 1) \leftarrow \text{not } \text{clrd}(V, 2), \text{not } \text{clrd}(V, 3), \text{vtx}(V)$$

is a shorthand for

$$\text{clrd}(v, 1) \leftarrow \text{not } \text{clrd}(v, 2), \text{not } \text{clrd}(v, 3), \text{vtx}(v)$$

$$\text{clrd}(u, 1) \leftarrow \text{not } \text{clrd}(u, 2), \text{not } \text{clrd}(u, 3), \text{vtx}(u)$$

$$\text{clrd}(1, 1) \leftarrow \text{not } \text{clrd}(1, 2), \text{not } \text{clrd}(1, 3), \text{vtx}(1)$$

...



Stable Models — cont'd

- A stratified program has a unique stable model (canonical model).
- It is **linear time to check** whether a set of atoms is a stable model of a ground program.
- It is **NP-complete to decide** whether a ground program has a stable model.
- Normal programs (without function symbols) give a **uniform solution** to every NP search problem.



Extensions to Normal Rules

- Encoding of choices

choice rules: $\{a\} \leftarrow b, \text{ not } c$

disjunctive rules: $a_1 \vee a_2 \leftarrow b, \text{ not } c$

- Cardinality constraints

$2 \{hd_1, \dots, hd_n\} 4$

- Weight constraints

$20 [hd_1 = 6, \dots, hd_n = 13]$

- Optimization

minimize $[hd_1 = 100, \dots, hd_n = 600]$

- Preferences, soft constraints, aggregates, ...



Generate-and-test programming

- Basic methodology:
 - **Generator rules**: provide candidate answer sets (typically encoded using choice constructs)
 - **Tester rules**: eliminate non-valid candidates (typically encoded using integrity constraints)
 - **Optimization statements**: Criteria for preferred answer sets (typically encoded using cost functions)



Example. k -coloring problem

- k -coloring: an assignment of one of k colors to each vertex such that no two adjacent vertices share a color.
- Input: available colors and a graph
 - `color(1) , color(k) .`
 - `vtx(v)`
 - `edge(v, u)`



k -coloring — cont'd

- An assignment of colors is represented by ground atoms of the form $\text{clrd}(v, c)$ where v is a vertex and c is an available color.
- The basic idea of the encoding:
 - (i) generator rules produce candidate stable models (assignments)
 - (ii) tester rules eliminate candidates which do not satisfy the coloring condition.



k -coloring — cont'd

```
% Encoding of the k-coloring problem
% Generator: producing candidate stable models
1 {clrd(V,C):color(C)} 1 :- vtx(V).

% Tester: eliminate candidates
% not satisfying the coloring condition.
:- edge(V,U), color(C), clrd(V,C), clrd(U,C).
```

- Given the encoding program (the input facts and the generator and tester rules):
 k -colorings and stable models correspond.
- k -coloring: facts $\text{clrd}(v, c)$ in the stable model.



Example: Review assignment

```
% DATA:
reviewer(r1). ...
paper(p1). ...
classA(r1,p1). ... % Preferred papers
classB(r1,p2). ... % Doable papers
coi(r1,p3). ... % Conflicts of interest

% PROBLEM
% Each paper is assigned 3 reviewers
3 { assigned(P,R):reviewer(R) } 3 :- paper(P).
% No paper assigned to a reviewer with coi
:- assigned(P,R), coi(R,P).
```



Review Assignment — cont'd

```
% No reviewer has an unwanted paper.
:- paper(P), reviewer(R),
   assigned(P,R), not classA(R,P), not classB(R,P).

% No reviewer has more than 8 papers
:- 9 { assigned(P,R): paper(P) }, reviewer(R).

% Each reviewer has at least 7 papers
:- { assigned(P,R): paper(P) } 6, reviewer(R).

% No reviewer has more than 2 classB papers
:- 3 { assignedB(P1,R): paper(P1) }, reviewer(R).

assignedB(P,R) :- classB(R,P), assigned(P,R).

% Minimize the number of classB papers
minimize [ assignedB(P,R):paper(P):reviewer(R) ].
```



ASP vs Other Approaches

- SAT, CSP, (M)IP
 - Similarities: search for models (assignments to variables) satisfying a set of constraints
 - Differences: no logical variables, database, DDB or KR techniques available, search space given by variable domains
- LP, CLP:
 - Similarities: database and DDB techniques
 - Differences: Search for proofs (not models), non-declarative features



Implementing ASP Solvers



ASP Solvers

- ASP solvers need to handle two challenging tasks
 - complex data
 - search
- The approach has been to use
 - **logic programming and deductive data base techniques** for the former
 - **SAT/CSP related search techniques** for the latter
- In the current systems: separation of concerns
 - ☞ A two level architecture



Architecture of ASP Solvers

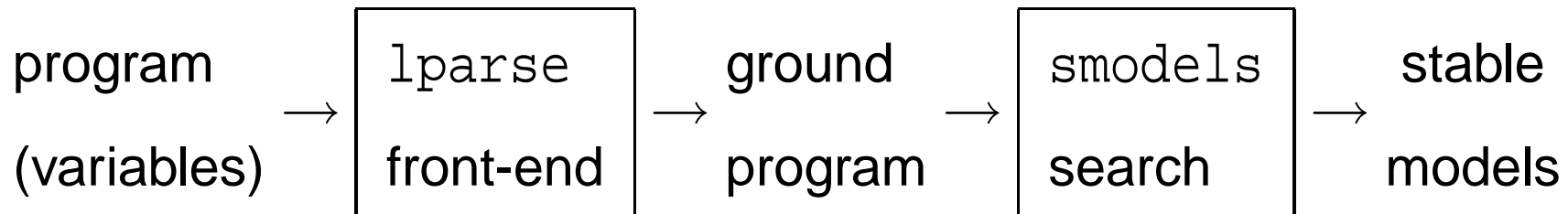
Typically a two level architecture employed

- **Grounding** step handles complex data:
 - Given program P with variables, generate a set of ground instances of the rules which preserves the models.
 - LP and DDB techniques employed
- **Model search** for ground programs:
 - Special-purpose search procedures
 - Translation to SAT



SMODELS system

(<http://www.tcs.hut.fi/Software/smodels>)



- Front-end: (deductive) DB techniques for stratified programs
- Special purpose search engine:
 - array data structures (Dowling-Gallier type)
 - local computations for large rule sets
 - linear space requirements
 - optimization built-in



Other ASP Implementations

dlv	http://www.dbai.tuwien.ac.at/proj/dlv/
GnT	http://www.tcs.hut.fi/Software/gnt/
CMODELS	http://www.cs.utexas.edu/users/tag/cmodels.html
ASSAT	http://assat.cs.ust.hk/
NoMoRe	http://www.cs.uni-potsdam.de/~linke/nomore/
XASP	distributed with XSB v2.6 http://xsb.sourceforge.net
aspps	http://www.cs.engr.uky.edu/ai/aspps/
ccalc	http://www.cs.utexas.edu/users/tag/cc/



Example. SOKOBAN game

■ Data

```
square(1, 1).      initial_at(4,3).
square(2, 1).      initial_box(3, 4).
square(3, 1).      target_square(2, 3).
...
```

■ Program

```
1 { move_to(X_2, Y_2, I) :
    same_segment(X_1, Y_1, X_2, Y_2, Dir) } 1 :-
    push(X_1, Y_1, Dir, I),
    has_neighbor(X_1, Y_1, Dir),
    time(I), I < n.
```



Applications



Applications

- Planning
 - USAdvisor project at Texas Tech:
A decision support system for the flight controllers of space shuttles
- Product configuration
 - Intelligent software configurator for Debian/Linux
 - WeCoTin project (Web Configuration Technology)
 - Spin-off (<http://www.variantum.com/>)
- Computer-aided verification
 - Partial order methods
 - Bounded model checking



Applications—cont'd

- VLSI routing
- Planning
- Combinatorial problems, network management, network security, security protocol analysis, linguistics . . .
- C. Baral. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.
- Applying ASP
 - as a stand alone system
 - as an embedded solver



Conclusions

ASP = KR + DB + search

- ASP emerging as a viable KR tool
- Efficient implementations under development
(Smodels, aspps, dlV, XASP, CMODELS, ASSAT, ...)
- Expanding functionality and ease of use
- Growing range of applications



Topics for Further Research

- Intelligent grounding
- Model computation without full grounding
- Program transformations, optimizations
- Model search: learning, restarting, backjumping, heuristics, local search techniques
- Language extensions
- Programming methodology
- Tool support



EU funded WASP working group

