Answer Set Programming
an Approach to Declarative Problem Solving

Ilkka Niemelä

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.
Solving a problem using ASP

Problem \rightarrow \text{Encoding} \rightarrow \text{Theory} \rightarrow \text{ASP solver} \rightarrow \text{Models} \rightarrow (\text{Solutions})
ASP—cont’d

- Solving a problem using ASP

  Problem $\rightarrow$ Encoding

  instance $\rightarrow$ Theory $\rightarrow$ ASP solver

  Models $\rightarrow$ (Solutions)

- Possible formal system

<table>
<thead>
<tr>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional logic</td>
</tr>
<tr>
<td>Truth assignments</td>
</tr>
<tr>
<td>CSP</td>
</tr>
<tr>
<td>Variable assignments</td>
</tr>
<tr>
<td>Logic programs</td>
</tr>
<tr>
<td>Stable models</td>
</tr>
</tbody>
</table>
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 \( v \in V \):
  \[
  v(1) \lor v(2) \lor v(3) \\
  \neg v(1) \lor \neg v(2) \\
  \neg v(1) \lor \neg v(3) \\
  \neg v(2) \lor \neg v(3)
  \]

  For each edge \((v, u) \in E\):
  \[
  \neg v(1) \lor \neg u(1) \\
  \neg v(2) \lor \neg u(2) \\
  \neg v(3) \lor \neg u(3)
  \]
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 $v \in V$:
  
  $v(1) \lor v(2) \lor v(3)$
  
  $\neg v(1) \lor \neg v(2)$
  
  $\neg v(1) \lor \neg v(3)$
  
  $\neg v(2) \lor \neg v(3)$

  For each edge $(v, u) \in E$:
  
  $\neg v(1) \lor \neg u(1)$
  
  $\neg v(2) \lor \neg u(2)$
  
  $\neg v(3) \lor \neg u(3)$

- 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

Problem → ENCODING → Theory → ASP solver → Models → (Solutions)
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:  
\[ \text{clrd}(V, 1) \leftarrow \neg \text{clrd}(V, 2), \neg \text{clrd}(V, 3), \text{vtx}(V) \]
\[ \text{clrd}(V, 2) \leftarrow \neg \text{clrd}(V, 1), \neg \text{clrd}(V, 3), \text{vtx}(V) \]
\[ \text{clrd}(V, 3) \leftarrow \neg \text{clrd}(V, 1), \neg \text{clrd}(V, 2), \text{vtx}(V) \]
\[ \leftarrow \text{edge}(V, U), \text{clrd}(V, C), \text{clrd}(U, C) \]

Data:  
\[ \text{vtx}(v) \quad \text{vtx}(u) \quad \ldots \]
\[ \text{edge}(v, u) \quad \text{edge}(u, w) \quad \ldots \]
Example. 3-coloring

Problem:  
\[
\begin{align*}
\text{clrd}(V, 1) & \leftarrow \neg \text{clrd}(V, 2), \neg \text{clrd}(V, 3), \text{vtx}(V) \\
\text{clrd}(V, 2) & \leftarrow \neg \text{clrd}(V, 1), \neg \text{clrd}(V, 3), \text{vtx}(V) \\
\text{clrd}(V, 3) & \leftarrow \neg \text{clrd}(V, 1), \neg \text{clrd}(V, 2), \text{vtx}(V) \\
& \leftarrow \text{edge}(V, U), \text{clrd}(V, C), \text{clrd}(U, C) \\
\end{align*}
\]

Data:  
\[
\begin{align*}
\text{vtx}(v) & \quad \text{vtx}(u) & \ldots \\
\text{edge}(v, u) & \quad \text{edge}(u, w) & \ldots \\
\end{align*}
\]

3-colorings and stable models of the encoding correspond: \( v \) colored \( i \) iff \( \text{clrd}(v, i) \) in the model.
LWs with Stable Models Semantics

- Consider normal logic program rules
  \[ A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n \]

- Seen as constraints on an answer set (stable model):
  - if \( B_1, \ldots, B_m \) are in the set and
  - none of \( C_1, \ldots, C_n \) 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:
  
  $b \leftarrow$
  
  $f \leftarrow b, \text{not } eb$
  
  $eb \leftarrow p$

- Another candidate model: $\{b, eb\}$ satisfies the rules but is not a proper stable model: $eb$ is included for no reason.

- Stable model: $\{b, f\}$
Stable Models — cont’d

- Program:
  \[
  \begin{align*}
  b & \leftarrow \\
  f & \leftarrow b, \text{not } eb \\
  eb & \leftarrow p
  \end{align*}
  \]

  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 redact 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:
\[
\text{clrd}(V, 1) \leftarrow \text{not clrd}(V, 2), \text{not clrd}(V, 3), \text{vtx}(V) \\
\text{clrd}(V, 2) \leftarrow \text{not clrd}(V, 1), \text{not clrd}(V, 3), \text{vtx}(V) \\
\text{clrd}(V, 3) \leftarrow \text{not clrd}(V, 1), \text{not clrd}(V, 2), \text{vtx}(V) \\
\text{vtx}(V) \leftarrow \text{edge}(V, U), \text{clrd}(V, C), \text{clrd}(U, C)
\]

Data:
\[
\text{vtx}(v) \quad \text{vtx}(u) \quad \ldots \\
\text{edge}(v, u) \quad \text{edge}(u, w) \quad \ldots
\]
Variables — cont’d

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

**Example.**

\[
clrd(V, 1) \leftarrow \neg clrd(V, 2), \neg clrd(V, 3), \text{vtx}(V)
\]

is a shorthand for

\[
clrd(v, 1) \leftarrow \neg clrd(v, 2), \neg clrd(v, 3), \text{vtx}(v)
\]
\[
clrd(u, 1) \leftarrow \neg clrd(u, 2), \neg clrd(u, 3), \text{vtx}(u)
\]
\[
clrd(1, 1) \leftarrow \neg clrd(1, 2), \neg clrd(1, 3), \text{vtx}(1)
\]
\[
\ldots
\]
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: \( \{a\} \leftarrow b, \neg c \)
  - disjunctive rules: \( a_1 \lor a_2 \leftarrow b, \neg c \)

- Cardinality constraints
  \( 2 \{hd_1, \ldots, hd_n\} 4 \)

- Weight constraints
  \( 20 [hd_1 = 6, \ldots, hd_n = 13] \)

- Optimization
  \( \text{minimize} [hd_1 = 100, \ldots, hd_n = 600] \)

- 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
  - \texttt{color(1),...}, \texttt{color(k)}.  
  - \texttt{vtx(v),...}.  
  - \texttt{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.
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).
% 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]

program (variables) → lparse front-end → ground program → smodels search → stable models

- 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
  USAAdvisor 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)

- 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 . . .
- 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