Aalto University
School of Science

# Answer Set Programming: A Declarative Approach to Solving Challenging Search Problems

Ilkka Niemelä

Department of Information and Computer Science
School of Science
Aalto University
Ilkka.Niemela@aalto.fi

# Aalto University

- Aalto University (http://www.aalto.fi/en/) established in 2010
- Merger of three leading Finnish universities in their fields:
  - Helsinki School of Economics
  - Helsinki University of Technology
  - University of Art and Design Helsinki.
- Unique combination of science and art with business and technology.

# Answer Set Programming (ASP)

- Basic principles outlined in the late 1990s
- Now well represented at research conferences and workshops (IJCAI, AAAI, ECAI, KR, ...)
- Competive implementations available (winning first places even in SAT competitions 2009, 2011)
- Growing number of applications
- An approach to modeling and solving **knowledge intensive search problems** with defaults, exceptions, definitions:
  planning, configuration, model checking, network management, linguistics, bioinformatics, combinatorics, ...

# Content

- Introduction to Answer Set Programming (ASP)
- Stable Model Semantics
- Solving Problems with ASP
- ASP Solver Technology
- Systems, Applications, Literature

# Part I

# Introduction to ASP

# Answer Set Programming

- Term coined by Vladimir Lifschitz in the late 1990s.
- Roots: KR, logic programming, nonmonotonic reasoning.
- Based on some formal system with semantics that assigns a theory (program/set of constraints) 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 **answer sets** of the theory.

# ASP—cont'd

- Solving a problem using ASP

  Problem instance $\longrightarrow$ $\boxed{\text{Encoding}}$ Theory $\longrightarrow$ $\boxed{\text{ASP solver}}$ Models (Solutions) $\longrightarrow$

- | Possible formal system | Models |
  |---|---|
  | Propositional logic | Truth assignments |
  | CSP | Variable assignments |
  | Logic programs | Stable models |
  | Model expansion | First-order structures |

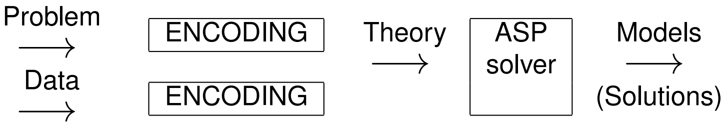
- Similar to constraint programming, MIP, SAT applications (SAT planning, symbolic model checking), ...

# Example. $k$-coloring problem with SAT

- 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 $v \in V$ include the clauses:
    $v_1 \lor v_2 \lor v_3$
    $\neg v_1 \lor \neg v_2$
    $\neg v_1 \lor \neg v_3$
    $\neg v_2 \lor \neg v_3$
  - and for each edge $(v, u) \in E$ the clauses:
    $\neg v_1 \lor \neg u_1$
    $\neg v_2 \lor \neg u_2$
    $\neg v_3 \lor \neg u_3$
- 3-colorings of a graph $(V, E)$ and models of the encoding correspond: vertex $v$ colored with color $i$ iff $v_i$ true in a model.

# ASP Using Logic Programs

- 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, inductive definitions



# Coloring Problem (Uniform Encoding)

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

# What is ASP Good for?

**Knowledge intensive search problems** with defaults, exceptions, inductive definitions:

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

# ASP Using Logic Programs

- Logic programming: framework for merging KR, DB, and search
- PROLOG style logic programming systems not directly suitable for ASP:
  - search for proofs (not models) and produce answer substitutions
  - 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
  - Smodels [N. and Simons 1996]
  - Basic ASP principles [N. 1999; Marek and Truszczyński 1999]
  - The term ASP coined by V. Lifschitz in 1999

Part II

# Stable Model Semantics

# LPs with Stable Models Semantics

- Consider first 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
  (negation by default; CWA)

# 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].

# Definite Programs

- For the reduct we need to consider first definite programs, i.e. normal programs without negation (not ).
- Such a program $P$ has a unique least model $LM(P)$ satisfying the rules.
- $LM(P)$ can be constructed, e.g., by forward chaining.

**Examples.**

| $P_1$ : | $P_2$ : | $P_3$ : |
|---|---|---|
| $p \leftarrow$ | $p \leftarrow q$ | $p \leftarrow q$ |
| $q \leftarrow p$ | $q \leftarrow p$ | $q \leftarrow p$ |
| | | $p \leftarrow$ |
| $LM(P_1) = \{p, q\}$ | $LM(P_2) = \{\}$ | $LM(P_2) = \{p, q\}$ |

# Stable Models — cont'd

- Consider the propositional (variable free) case:
  $P$ — ground program
  $S$ — set of ground atoms
- Reduct $P^S$ (Gelfond-Lifschitz)
  - delete each rule having a body literal not $C$ with $C \in S$
  - remove all negative body literals from the remaining rules
- $P^S$ is a definite program (and has a unique least model $LM(P^S)$)
- **$S$ is a stable model of $P$ iff $S = LM(P^S)$.**

# Example. Stable models

| $S$ | $P$ | $P^S$ | $LM(P^S)$ |
|---|---|---|---|
| $\{b, f\}$ | $b \leftarrow$ | $b \leftarrow$ | $\{b, f\}$ |
| | $f \leftarrow b, \text{not } eb$ | $f \leftarrow b$ | |
| | $eb \leftarrow p$ | $eb \leftarrow p$ | |
| $\{b, eb\}$ | $b \leftarrow$ | $b \leftarrow$ | $\{b\}$ |
| | $f \leftarrow b, \text{not } eb$ | | |
| | $eb \leftarrow p$ | $eb \leftarrow p$ | |

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

# Example. Stable models

- A program can have **none**, one, or **multiple** stable models.
- Program:
  $p \leftarrow \text{not } q$
  $q \leftarrow \text{not } p$

  Two stable models:
  $\{p\}$
  $\{q\}$
- Program:
  $p \leftarrow \text{not } p$

  No stable models

# Programs with variables

- Variables are needed for uniform encodings
- Semantics: **Herbrand models**
- A rule is seen as a shorthand for the set of its ground instantiations over the Herbrand universe of the program
- The **Herbrand universe** is the set of terms built from the constants and functions in the program

☞ Database assumptions: unique names, domain closure

# Example. Programs with variables

- For the program $P$:

```
edge(1,2).
edge(1,3).
edge(2,4).
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

  The Herbrand universe is { `1,2,3,4` }.

- Hence, the rule `path(X,Y) :- edge(X,Y).` in $P$ represents the set of ground instantiations:

```
path(1,1) :- edge(1,1).
path(1,2) :- edge(1,2).
path(2,1) :- edge(2,1).
path(2,2) :- edge(2,2).
path(1,3) :- edge(1,3).
...
```

# Stable Models — cont'd

- A stratified program (no recursion through negation) 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 encoding** to every NP search problem.

# Extensions to Normal Programs

- An **integrity constraint** is a rule without a head:

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

  - It can be seen as a shorthand for

$$F \leftarrow \text{not } F, B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n$$

  - and it eliminates stable models where the body $B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n$ is satisfied.
- **Classical negation**
  can be handled by normal programs (renaming):

| | | |
|---|---|---|
| $p \leftarrow \text{not } \neg p$ | corresponds to | $p \leftarrow \text{not } p'$ |
| | | $\leftarrow p, p'$ |

# Extensions to Normal Programs

- **Encoding of choices**
  - A key point in ASP
  - Choices can be encoded using normal rules with unstratified negation

$$
\begin{aligned}
a &\leftarrow \text{not } a', b, \text{not } c \\
a' &\leftarrow \text{not } a
\end{aligned}
$$

  - **Choice rules**, however, provide a much more intuitive encoding:

$$\{a\} \leftarrow b, \text{not } c$$

  - Disjunctive rules: $a \vee a' \leftarrow b, \text{not } c$
    - Higher expressivity and complexity ($\Sigma_2^p$)
    - Special purpose implementations (`dlv`,`claspD`)
    - Can be implemented also using an ASP solver for normal programs as the **core engine** (`GnT`)

# Extensions — cont'd

- Many extensions implemented using an ASP solver as the **core engine**:
  - preferences
  - nested logic programs
  - circumscription, planning, diagnosis, ...
  - HEX-programs
  - DL-programs
- Aggregates (count, sum, ...)
- Optimization
- Function symbols
- Built-in predicates and functions:

```
nextstate(Y,X) :- time(X), time(Y), Y = X + 1.
```

# Example. Rules in `lparse`

- Cardinality constraints

  ```
  2 { hd_1,...,hd_n } 4
  ```

- Weight constraints

  ```
  200 [ hd_1 = 60,...,hd_n = 130]
  ```

  A.k.a. **pseudo-Boolean constraints**:

$$60hd_1 + \cdots + 130hd_n \geq 200$$

- Optimization

  ```
  minimize [ hd_1 = 100,...,hd_n = 180 ].
  ```

- Conditional literals:
  expressing sets in cardinality and weight constraints

  ```
  1 {colored(V,C):color(C)} 1 :- vtx(V).
  ```

Part III

# Solving Problems using ASP

# Programming Methodology

- Uniform encodings: separate data and problem encoding
- Basic methodology: **generate and test**
  - **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 using cost functions)

# Example: Coloring

```
% Problem encoding

% Generator rule
1 {colored(V,C):color(C)} 1 :- vtx(V).

% Tester rule
:- edge(V,U), color(C), colored(V,C), colored(U,C).

% Optimization statement
minimize {colored(V,y):vtx(V)}.

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

# 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 encoding

% Generator rule
% Each paper is assigned 3 reviewers
3 { assigned(P,R):reviewer(R) } 3 :- paper(P).
```

# Review Assignment — cont'd

```
% Tester rules
% 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) ].
```

# Fixed Points

- The stable model semantics captures inherently **minimal fixed points** enabling compact encodings of **closures and inductive definitions**
- **Example.** Reachability from node $s$.

```
r(s).
r(V) :- edge(U,V), r(U).
edge(a,b). ...
```

- The program captures reachability:
  it has a unique stable model $S$ s.t. $v$ is reachable from $s$ iff $r(v) \in S$.
- **Example.** Transitive closure of a relation $q(X, Y)$

```
t(X,Y) :- q(X,Y).
t(X,Y) :- q(X,Z), t(Z,Y).
```

# ASP vs Other Approaches

- SAT, CSP, (M)IP
  - Similarities: search for models (assignments to variables) satisfying a set of constraints.
  - Differences: no logical variables, fixed points, 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.

Part IV

# ASP Solver Technology

# 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
  - Exploiting SAT/SMT solver technology

# Typical ASP System Tool Chain

program (variables) $\rightarrow$ Grounder $\rightarrow$ ground program $\rightarrow$ Model finder $\rightarrow$ stable models

- Grounder:
  - (deductive) DB techniques
  - built-in predicates/functions (e.g. arithmetic)
  - function symbols
- Model finder:
  - SAT technology (propagation, conflict driven clause learning)
  - Special propagation rules for rules
  - Support for cardinality and weight constraints and optimization built-in

# SAT and ASP

- ASP systems have much more expressive modelling languages than SAT: variables, built-ins, aggregates, optimization
- For model finding for ground normal programs results carry over: efficient unit propagation techniques, conflict driven learning, backjumping, restarting, ...
- ASP model finders have special (unfounded set based) propagation rules for recursive rules
- ASP model finders have **built-in support for aggregates** (cardinality and weight constraints) and optimization
- One pass compact translations to SAT and SMT available: progress in SAT and SMT solver technology can also be exploited directly in ASP model finding.

# Part V

# Systems, Applications, Literature

# Some ASP Systems

**Grounders:**

| | |
|---|---|
| `dlv` | `http://www.dbai.tuwien.ac.at/proj/dlv/` |
| `gringo` | `http://potassco.sourceforge.net/` |
| `lparse` | `http://www.tcs.hut.fi/Software/smodels/` |
| `XASP` | with XSB `http://xsb.sourceforge.net` |

**Model finders (disjunctive programs):**

| | |
|---|---|
| `claspD` | `http://potassco.sourceforge.net/` |
| `dlv` | `http://www.dbai.tuwien.ac.at/proj/dlv/` |
| `GnT` | `http://www.tcs.hut.fi/Software/gnt/` |

# Some ASP Systems

## Model finders (non-disjunctive programs):

| | |
|---|---|
| ASSAT | http://assat.cs.ust.hk/ |
| clasp | http://potassco.sourceforge.net/ |
| CMODELS | http://userweb.cs.utexas.edu/users/tag/cmodels/ |
| LP2DIFF | http://www.tcs.hut.fi/Software/lp2diff/ |
| LP2SAT | http://www.tcs.hut.fi/Software/lp2sat/ |
| Smodels | http://www.tcs.hut.fi/Software/smodels/ |
| SUP | http://userweb.cs.utexas.edu/users/tag/sup/ |

- For systems, performance, benchmarks, and examples, see for instance the latest **ASP competition**:
  `http://dtai.cs.kuleuven.be/events/ASP-competition/`

# Applications

- Planning
  For example, 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

- Data and information Integration
- Semantic web reasoning
- Team building at Gioia Tauro Seaport
- Repairing large-scale biological networks
- ASP-based music composition system (`anton-demo.wav`)
- VLSI routing, planning, combinatorial problems, network management, network security, security protocol analysis, linguistics ...
- WASP Showcase Collection
  `http://www.kr.tuwien.ac.at/research/projects/WASP/showcase.html`

# Some Literature


- C. Baral. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.
- V. Lifschitz. Foundations of Logic Programming. `http://www.cs.utexas.edu/~vl/papers/flp.ps`
- V. Lifschitz. Introduction to Answer Set Programming. `http://www.cs.utexas.edu/~vl/papers/esslli.ps`
- T. Eiter, G. Ianni, and T. Krennwallner. A Primer on Answer Set Programming. `http://www.kr.tuwien.ac.at/staff/tkren/pub/2009/rw2009-asp.pdf`
- M. Gebser and T. Schaub. Answer Set Solving in Practice. A Tutorial at IJCAI'11. `http://www.cs.uni-potsdam.de/~torsten/ijcai11tutorial/`

# Conclusions

**ASP = KR + DB + search**

- ASP emerging as a viable KR tool
- Efficient implementations under development
- 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
- Distributed and parallel implementation techniques
- Language extensions
- Programming methodology
- Testing techniques
- Tool support: debuggers, IDEs