Answer Set Programming

- Answer Set Programs
- Answer Set Semantics
- Implementation Techniques
- Using Answer Set Programming
Example ASP: 3-Coloring

Problem: For a graph \((V, E)\) find an assignment of one of 3 colors to each vertex such that no adjacent vertices share a color.

\[
\begin{align*}
\text{clrd}(V, 1) & : \text{not clrd}(V, 2), \text{not clrd}(V, 3), \text{vtx}(V). \\
\text{clrd}(V, 2) & : \text{not clrd}(V, 1), \text{not clrd}(V, 3), \text{vtx}(V). \\
\text{clrd}(V, 3) & : \text{not clrd}(V, 1), \text{not clrd}(V, 2), \text{vtx}(V). \\
& : \text{edge}(V, U), \text{clrd}(V, C), \text{clrd}(U, C).
\end{align*}
\]

\text{vtx}(a). \text{vtx}(b). \text{vtx}(c). \text{edge}(a, b). \text{edge}(a, c). \ldots
ASP in Practice

- Compact, easily maintainable representation
- Roots: logic programming
- Solutions = Answer sets to logic program
Some Applications

- Constraint satisfaction
- Planning, Routing
- Computer-aided verification
- Security analysis
- Configuration
- Diagnosis
ASP vs. Prolog

- Prolog not directly suitable for ASP
  - Models vs. proofs + answer substitutions
  - Prolog not entirely declarative

- **Answer set semantics**: alternative semantics for negation-as-failure

- Existing ASP Systems: CLINGO, SMODELS, DLV and others
Answer Set Semantic

- A logic program clause

\[ A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n \quad (m \geq 0, n \geq 0) \]

is seen as constraint on an answer (model): if \( B_1, \ldots, B_m \) are in the answer and none of \( C_1, \ldots, C_m \) is, then must \( A \) be included in the answer.

- Answer sets should be **minimal**

- Answer sets should be **justified**
Answer Sets: Example (1)

\[
p :\neg q.
\]
\[
r :\neg p.
\]
\[
s :\neg r, \neg p.
\]

The answer set is \{p, r\}

- \{p\} is not an answer (because it's not a model)
- \{r, s\} is not an answer (because \(r\) included for no reason)
Answer Sets: Example (2)

\[
p :\neg q.
p :\neg r.
q :\neg \text{not } r.
p :\neg \text{not } r.
\]

There are two answers: \{p, q\} and \{p, r\}.

Note that in Prolog, p is not derivable.
Answer Sets: Definition

Consider a program $P$ of ground clauses

$$A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n \quad (m \geq 0, n \geq 0)$$

Let $S$ be a set of ground atoms.

- **Reduct $P^S$**: 
  - delete each clause with some $\text{not } C_i$ such that $C_i \in S$
  - delete each $\text{not } C_i$ such that $C_i \not\in S$

- $S$ answer set (also called stable model) : $\iff S = \text{least-model}(P^S)$
Properties

- Programs can have multiple answer sets

\[
\begin{align*}
p_1 & :\neg q_1. & q_1 & :\neg p_1. \\
\vdots & & \vdots \\
p_n & :\neg q_n. & q_n & :\neg p_n.
\end{align*}
\]

This program has \(2^n\) answers

- Programs can have no answers

\[
\begin{align*}
p & :\neg q. \\
q & :\neg p.
\end{align*}
\]
Properties (ctd)

- A **stratified** program has a unique answer (= the **standard** model).
- Checking whether a set of atoms is a stable model can be done in linear time.
- Deciding whether a program has a stable model is NP-complete.
Programs with Variables and Functions

Semantics: Herbrand models

Clause seen as shorthand for all its ground instances

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

stands for

\[ \text{clrd}(a,1) : \neg \text{clrd}(a,2), \neg \text{clrd}(a,3), \text{vtx}(a). \]
\[ \text{clrd}(b,1) : \neg \text{clrd}(b,2), \neg \text{clrd}(b,3), \text{vtx}(b). \]
\[ \vdots \]

Constraint

\[ \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n \]

shorthand for \[ \text{false} \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n, \neg \text{false} \]
Example ASP: 3-Coloring

\begin{verbatim}
clrd(V,1) :- not clrd(V,2), not clrd(V,3), vtx(V).
clrd(V,2) :- not clrd(V,1), not clrd(V,3), vtx(V).
clrd(V,3) :- not clrd(V,1), not clrd(V,2), vtx(V).
:- edge(V,U), clrd(V,C), clrd(U,C).

vtx(a). vtx(b). vtx(c). edge(a,b). edge(a,c).
\end{verbatim}

Each answer set is a valid coloring, for example:

\{clrd(a,1), clrd(b,2), clrd(c,2)\}
Generalization: Classical Negation

- Rules built using classical literals (not just atoms)
- Answers are sets of literals
- Example:

  \[ p :\neg q \]
  \[ \neg p :\neg q \]

  An answer is \(\{\neg q\}\)
Generalization: Classical Negation (ctd)

- Classical negation can be handled by normal programs:
  - treat $\neg A$ as a new atom (renaming)
  - add the constraint $\leftarrow A, \neg A$

- Example:

```
p :- not q'
q' :- not p
   :- p, p'
   :- q, q'
```

has the answer \{q'\}
Generalization: Disjunction

- Rules can have disjunctions in the head
- Direct generalization of answer set semantics
- Example:
  \[ p \lor q : \neg \ p \]
  has the only answer \{q\}
- Another example:
  \[ p \lor q : \neg \ p \]
  \[ p : \neg \ q \]
  has no answer
ASP Solver: Architecture

Two challenging tasks: handle complex data; search

Two-layer architecture:

- **Grounding** handles complex data: A set of ground clauses is generated which preserves the models

- **Model search** uses special-purpose search procedures
Grounding: Domain Restrictions

- Domain-restricted programs guarantee decidability.

- Domain-restricted programs consist of two parts:
  1. Domain predicate definitions (a stratified clause set), where each variable occurs in a positive domain predicate defined in an earlier stratum;
  2. Clauses where each variable occurs in a positive domain predicate in the body.

- The domain predicate definitions have a unique answer, which is subset of every solution to the program.

- Only those ground instances of clauses need to be generated where the domain predicates in the body are true.
Example: Domain Predicate Definitions

col(1).  col(2).  col(3).

r(a,b).  r(a,c).  ...

d(U) :- r(V,U).

tr(V,U) :- r(V,U).

tr(V,U) :- r(V,Z), tr(Z,U), d(U).

edge(t(V), t(U)) :- tr(V,U), not tr(U,U), not tr(V,V).

vtx(V) :- edge(V,U).

vtx(U) :- edge(V,U).
Example: Domain-Restricted Clauses

\[
\begin{align*}
\text{clrd}(V,1) & : - \text{not clrd}(V,2), \text{not clrd}(V,3), \text{vtx}(V). \\
\text{clrd}(V,2) & : - \text{not clrd}(V,1), \text{not clrd}(V,3), \text{vtx}(V). \\
\text{clrd}(V,3) & : - \text{not clrd}(V,1), \text{not clrd}(V,2), \text{vtx}(V). \\
& : - \text{edge}(V,U), \text{col}(C), \text{clrd}(V,C), \text{clrd}(U,C).
\end{align*}
\]
Example: Grounding

Suppose that the unique stable model for the definition of the domain predicate $\text{vtx}(V)$ contains $\text{vtx}(v_1), \ldots, \text{vtx}(v_n)$

Then for the clause

$$\text{clrd}(V,1) : - \text{not clrd}(V,2), \text{not clrd}(V,3), \text{vtx}(V).$$

grounding produces

$$\text{clrd}(v_1,1) : - \text{not clrd}(v_1,2), \text{not clrd}(v_1,3).$$

$$\ldots$$

$$\text{clrd}(v_n,1) : - \text{not clrd}(v_n,2), \text{not clrd}(v_n,3).$$
Search

- Backtracking over truth-values for atoms

![Diagram of a search tree with nodes labeled a, not a, not b, b, and c. The node labeled c is marked as stable.]

- Each node consists of a model candidate (set of literals)

- Propagation rules are applied after each choice
Propagation Rules

- A propagation rule extends a model candidate by one or more new literals.

- Example: Given \( q \leftarrow p_1, \text{not } p_2 \) and candidate \( \{p_1, \text{not } q\} \): derive \( p_2 \)

- Propagation rules need to be correct: If \( L \) is derived from model candidate \( A \) then \( L \) holds in every stable model compatible with \( A \).
Example: Propagation Rule “Upper Bound”

Consider program $P$ and candidate model $A$

Let $P'$ be all clauses in $P$

- whose body is not false under $A$
- without negative body literals

If $p \notin$ least-model ($P'$) derive not $p$

\[
\begin{align*}
P: & \quad p_2 := p_1, \text{ not } q_1. \quad A: \{q_2\} \quad P': p_2 := p_1. \\
& \quad p_1 := p_2, \text{ not } q_1. \quad p_1 := p_2. \\
& \quad p_2 := \text{ not } q_2. \\
\text{Derive:} & \quad \text{not } p_1, \text{ not } p_2, \text{ not } q_1, \text{ not } q_2
\end{align*}
\]
### Schema of Local Propagation Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Only clauses for q</th>
<th>Candidate</th>
<th>Derive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1)$</td>
<td>$q \leftarrow p_1, \text{not } p_2$</td>
<td>$p_1, \text{not } p_2$</td>
<td>$q$</td>
</tr>
</tbody>
</table>
| $(R_2)$ | $q \leftarrow p_1, \text{not } p_2$
$q \leftarrow p_3, \text{not } p_4$ | $p_2, \text{not } p_3$ | $\text{not } q$ |
| $(R_3)$ | $q \leftarrow p_1, \text{not } p_2$ | $q$ | $p_1, \text{not } p_2$ |
| $(R_4)$ | $q \leftarrow p_1, \text{not } p_2$ | $\text{not } q, p_1$ | $p_2$ |
Example

\[ f : - \neg g, \neg h \]
\[ g : - \neg f, \neg h \]
\[ f : - g \]
Lookahead

Given a program $P$ and a candidate model $A$.

If, for a literal $L$, $\text{propagate}(P, A \cup \{L\})$ contains a conflict (some $p$ together with $\text{not } p$), derive the complement of $L$. 
Search Heuristics

Heuristics to select the next atom for splitting the search tree:

- an atom with the maximal number of occurrences in clauses of minimal size
- an atom with the maximal number of propagations after the split
- an atom with the smallest remaining search space after split + propagation
Using ASPs (Example 1): Hamiltonian Cycles

- **A Hamiltonian cycle**: a closed path that visits all vertices of a graph exactly once
- **Input**: a graph
  - vtx(a), ...
  - edge(a, b), ...
  - initialvtx(a)

- **Weight** atoms in ASP:

  \[ m \{ p : \text{d}(x) \} \leq n \]

  means that an answer contains at least \( m \) and at most \( n \) different \( p \)-instances which satisfy \( \text{d}(x) \). If \( m \) is omitted, there is no lower bound; if \( n \) is omitted, there is no upper bound.
Hamiltonian Cycles (ctd)

- Candidate answer sets: subsets of edges
- Generator (using a weight atom):
  \[
  \{ \text{hc}(X, Y) \} : \text{edge}(X, Y)
  \]
- Answer sets for the generator given a graph:
  
  input graph
  + a subset of the ground facts \( \text{hc}(a, b) \) for which there is \( \text{edge}(a, b) \)
Hamiltonian Cycles (ctd)

- Tester(1): Each vertex has at most one chosen incoming and one outcoming edge

  
  \[- \text{hc}(X, Y), \ \text{hc}(X, Z), \ \text{edge}(X, Y), \ \text{edge}(X, Z), \ Y \neq Z. \]
  
  \[- \text{hc}(Y, X), \ \text{hc}(Z, X), \ \text{edge}(Y, X), \ \text{edge}(Z, X), \ Y \neq Z. \]

- Only subsets of chosen edges \text{hc}(a, b) forming paths (possibly closed) pass this test
Hamiltonian Cycles (ctd)

• Tester(2): Every vertex is reachable from a given initial vertex through chosen \(hc(a,b)\) edges

\[
\begin{align*}
\texttt{\tt :-} & \quad \texttt{vtx}(X), \ \texttt{not} \ \texttt{r}(X). \\
\texttt{r}(Y) & \quad \texttt{:­} \ \texttt{hc}(X,Y), \ \texttt{edge}(X,Y), \ \texttt{initialvtx}(X). \\
\texttt{r}(Y) & \quad \texttt{:­} \ \texttt{hc}(X,Y), \ \texttt{edge}(X,Y), \ \texttt{r}(X), \ \texttt{not} \ \texttt{initialvtx}(X).
\end{align*}
\]

• Only Hamiltonian cycles pass both tests
Hamiltonian Cycles (ctd)

- Using more weight atoms enables even more compact encoding

- Tester(1) using 2 variables:

  :- 2 { hc(X,Y) : edge(X,Y) }, vtx(X).
  :- 2 { hc(X,Y) : edge(X,Y) }, vtx(Y).
Hamiltonian Cycles (ctd): Undirected Cycles

- **Instance** \((V,E)\):

\[
\begin{align*}
\text{vtx}(v). \\
\text{edge}(v,u). & \quad \% \text{one fact for each edge in } E
\end{align*}
\]

- **Generator**:

\[
2 \{ \text{hc}(V,U) : \text{edge}(V,U), \\
\text{hc}(W,V) : \text{edge}(W,V) \} 2 :- \text{vtx}(V).
\]

- **Tester**:

\[
\begin{align*}
\text{r}(V) :- \text{initialvtx}(V). \\
\text{r}(V) :- \text{hc}(V,U), \text{edge}(V,U), \text{r}(U). \\
\text{r}(V) :- \text{hv}(U,V), \text{edge}(U,V), \text{r}(U). \\
:- \text{vtx}(V), \text{not} \text{ r}(V).
\end{align*}
\]
Using ASPs (Example 2): Verification

- Verify, on the basis of a given formal specification, that a dynamic system satisfies desirable properties
- Example:

Given a formal specification of Tic-Tac-Toe, ASP can be used to verify that it is a turn-taking game and that no cell ever contains two symbols.
Formal Specification: Initial State

\texttt{init(cell(1,1,b))}.
\texttt{init(cell(1,2,b))}.
\texttt{init(cell(1,3,b))}.
\texttt{init(cell(2,1,b))}.
\texttt{init(cell(2,2,b))}.
\texttt{init(cell(2,3,b))}.
\texttt{init(cell(3,1,b))}.
\texttt{init(cell(3,2,b))}.
\texttt{init(cell(3,3,b))}.
\texttt{init(\texttt{control(xplayer)})}.
Formal Specification: State Transitions

\[
\text{legal}(P, \text{mark}(X,Y)) :\text{-} \begin{align*}
\text{true}(\text{cell}(X,Y,b)) , \\
\text{true}(\text{control}(P)).
\end{align*}
\]

\[
\text{legal}(\text{xplayer}, \text{noop}) :\text{-} \begin{align*}
\text{true}(\text{cell}(X,Y,b)) , \\
\text{true}(\text{control}(\text{oplayer})).
\end{align*}
\]

\[
\text{legal}(\text{oplayer}, \text{noop}) :\text{-} \begin{align*}
\text{true}(\text{cell}(X,Y,b)) , \\
\text{true}(\text{control}(\text{xplayer})).
\end{align*}
\]
Formal Specification: State Change

\[
\text{next(cell}(M,N,x)) \; : \; \text{does(xplayer,mark}(M,N))\).
\]
\[
\text{next(cell}(M,N,o)) \; : \; \text{does(oplayer,mark}(M,N))\).
\]
\[
\text{next(cell}(M,N,W)) \; : \; \text{true(cell}(M,N,W))\), \; W!=b.
\]
\[
\text{next(cell}(M,N,b)) \; : \; \text{true(cell}(M,N,b))\),
\text{does(P,mark}(J,K))\), \; M!=J.
\]
\[
\text{next(cell}(M,N,b)) \; : \; \text{true(cell}(M,N,b))\),
\text{does(P,mark}(J,K))\), \; N!=K.
\]
\[
\text{next(control}(xplayer)) \; : \; \text{true(control}(oplayer))\).
\]
\[
\text{next(control}(oplayer)) \; : \; \text{true(control}(xplayer))\).
\]
Verification (ctd)

- Properties of dynamic systems are verified inductively
- Induction base:

\[
\text{player(xplayer)}. \\
\text{player(oplayer)}. \\
t0 : - 1 \{ \text{init(control(X)) : player(X) } \} 1. \\
: - t0.
\]

- This program has no answer set, which proves the fact that initially exactly one player has the control.
Verification (ctd)

- State generator for the induction step:

\[
\begin{align*}
\text{coordinate}(1..3). \\
\text{symbol}(x). \text{symbol}(o). \text{symbol}(b).
\end{align*}
\]

\[
\begin{align*}
\text{tdomain}(&\text{cell}(X,Y,C)) \leftarrow \text{coordinate}(X), \text{coordinate}(Y), \\
&\quad \text{symbol}(C).
\end{align*}
\]

\[
\begin{align*}
\text{tdomain}(&\text{control}(X)) \leftarrow \text{player}(X).
\end{align*}
\]

\[
\{ \text{true}(T) : \text{tdomain}(T) \}.
\]

- Transition generator for the induction step:

\[
\begin{align*}
\text{ddomain}(&\text{mark}(X,Y)) \leftarrow \text{coordinate}(X), \text{coordinate}(Y).
\end{align*}
\]

\[
\begin{align*}
\text{ddomain}(&\text{noop}).
\end{align*}
\]

\[
\begin{align*}
1 \{ \text{does}(P,M) : \text{ddomain}(M) \} 1 \leftarrow \text{player}(P).
\end{align*}
\]
Verification (ctd)

- Tester(1): Every transition must be legal
  
  \[- \text{does}(P,M), \text{not legal}(P,M). \]

- Tester(2): Induction hypothesis
  
  \[t_0 :- 1 \{ \text{true}(\text{control}(X)) \text{ : player}(X) \} 1. \]  
  
  \[- \text{not } t_0. \]

- Induction step
  
  \[t :- 1 \{ \text{next}(\text{control}(X)) \text{ : player}(X) \} 1. \]  
  
  \[- t. \]

- This program has no answer, which proves the claim that in every reachable state exactly one player has the control.
Verification (ctd)

- Induction base to prove that cells have unique contents:

  t0(X,Y) :- 1 { init(cell(X,Y,Z)) : symbol(Z) } 1.
  t0 :- not t0(X,Y).
  :- not t0.

- This program has no answer set, which proves the claim.
Verification (ctd)

- Induction hypothesis

\[ t_0(X,Y) \ :- \ 1 \ \{ \ true(cell(X,Y,Z)) \ : \ symbol(Z) \ \} \ 1. \]
\[ t_0 \ :- \ not \ t_0(X,Y). \]
\[ :- \ t_0. \]

- Induction step to prove that cells have unique contents

\[ t(X,Y) \ :- \ 1 \ \{ \ next(cell(X,Y,Z)) \ : \ symbol(Z) \ \} \ 1. \]
\[ t \ :- \ not \ t(X,Y). \]
\[ :- \ not \ t. \]

This program has an answer set! Need to add uniqueness-of-control:

\[ p \ :- \ 1 \ \{ \ true(control(X)) \ : \ player(X) \ \} \ 1. \]
\[ :- \ not \ p. \]

Now the program has no answer set, which proves the claim.