Systems and Solving Techniques for Knowledge Representation

– Guess & Check –

Marco Maratea
University of Genoa, Italy

066 011 Double degree programme Computational Logic
066 931 Computational Intelligence
066 937 Software Engineering & Internet Computing
Institute of Information Systems

ASP Basics

ASP:

Datalog ← done!

+ Default negation ← done!
+ Disjunction ← done!
+ Integrity Constraints ← done!
+ Weak Constraints ← done!
+ Aggregate atoms ← done!

How to program in ASP?

- Programming methodology

ASP Basics

ASP:

Datalog $\leftarrow$ done!

+ Default negation $\leftarrow$ done!
+ Disjunction $\leftarrow$ done!
+ Integrity Constraints $\leftarrow$ done!
+ Weak Constraints $\leftarrow$ done!
+ Aggregate atoms $\leftarrow$ done!

How to program in ASP?

- Programming methodology

Problem solving in ASP

The idea of ASP:

1. Write a program representing a computational problem

   → i.e., such that answer sets correspond to solutions

2. Use a solver to find solutions

Programming Steps:

1. Model your domain

   → Single out input/output predicates

2. Write a logic program modeling your problem

   → Use predicates representing relevant entities

   → **Hint:** take input data separated from derived ones

Problem solving in ASP

The idea of ASP:

1. Write a program representing a computational problem
   → i.e., such that answer sets correspond to solutions
2. Use a solver to find solutions

Programming Steps:

1. Model your domain
   → Single out input/output predicates
2. Write a logic program modeling your problem
   → Use predicates representing relevant entities
   → **Hint:** take input data separated from derived ones

Problem solving in ASP

The idea of ASP:

1. Write a program representing a computational problem
   $\rightarrow$ i.e., such that answer sets correspond to solutions
2. Use a solver to find solutions

Programming Steps:

1. Model your domain
   $\rightarrow$ Single out input/output predicates
2. Write a logic program modeling your problem
   $\rightarrow$ Use predicates representing relevant entities
   $\rightarrow$ **Hint:** take input data separated from derived ones

Problem solving in ASP

The idea of ASP:

1. Write a program representing a computational problem

   $\rightarrow$ i.e., such that answer sets correspond to solutions

2. Use a solver to find solutions

Programming Steps:

1. Model your domain

   $\rightarrow$ Single out input/output predicates

2. Write a logic program modeling your problem

   $\rightarrow$ Use predicates representing relevant entities

   $\rightarrow$ **Hint:** take input data separated from derived ones

Direct Encodings when...

Use a "Direct" Encoding with Datalog rules for

- Polynomial Problems, etc.

Example (Reachability)

Problem: Find all nodes reachable from the others.
Input: *edge*(_, _).

% X is reachable from Y if an edge (X,Y) exists
$reachable(X, Y) \text{ :- } edge(X, Y).$

% Reachability is transitive
$reachable(X, Y) \text{ :- } reachable(X, Z), edge(Z, Y).$

The method in often unfeasible for search problems from NP and beyond: need for a programming methodology

Direct Encodings when...

Use a "Direct" Encoding with Datalog rules for

- Polynomial Problems, etc.

Example (Reachability)

Problem: Find all nodes reachable from the others.
Input: *edge*(_, _).

% X is reachable from Y if an edge (X,Y) exists
reachable(*X*, *Y*) :- *edge*(*X*, *Y*).

% Reachability is transitive
reachable(*X*, *Y*) :- *reachable*(*X*, *Z*), *edge*(*Z*, *Y*).

The method in often unfeasible for search problems from NP and beyond: need for a programming methodology

Programming Methodology

Guess & Check & Optimize (GCO)

1. Guess solutions → using disjunctive rules
2. Check admissible ones → using strong constraints

Optimization problem?

3. Specify Preference criteria → using weak constraints

In other words...

1. disjunctive rules → generate candidate solutions
2. constraints → test solutions discarding unwanted ones
3. weak constraints → single out optimal solutions

Programming Methodology

Guess & Check & Optimize (GCO)

1. Guess solutions → using disjunctive rules
2. Check admissible ones → using strong constraints

Optimization problem?

3. Specify Preference criteria → using weak constraints

In other words...

1. disjunctive rules → generate candidate solutions
2. constraints → test solutions discarding unwanted ones
3. weak constraints → single out optimal solutions

Programming Methodology

Guess & Check & Optimize (GCO)

1. Guess solutions → using disjunctive rules
2. Check admissible ones → using strong constraints

Optimization problem?

3. Specify Preference criteria → using weak constraints

In other words...

1. disjunctive rules → generate candidate solutions
2. constraints → test solutions discarding unwanted ones
3. weak constraints → single out optimal solutions

Guess and Check (Example 1)

Example (Group Assignments)

Problem: We want to partition a set of persons in two groups, while avoiding that father and children belong to the same group.
Input: persons and fathers are represented by *person*(_) and *father*(_, _).

% a disjunctive rule to “guess” all the possible assignments

$$group(P, 1) \mid group(P, 2) \text{ :- } person(P).$$

% a constraint to discard unwanted solutions
% i.e., father and children cannot belong to the same group

$$\text{:- } group(P1, G), group(P2, G), father(P1, P2).$$

...so how does it work really?

Guess and Check (Example 1)

Example (Group Assignments)

Problem: We want to partition a set of persons in two groups,
while avoiding that father and children belong to the same group.
Input: persons and fathers are represented by *person*(_) and *father*(_, _).

% a disjunctive rule to “guess” all the possible assignments

$$group(P, 1) \mid group(P, 2) \mathrel{:-} person(P).$$

% a constraint to discard unwanted solutions
% i.e., father and children cannot belong to the same group

$$\mathrel{:-} group(P1, G), group(P2, G), father(P1, P2).$$

...so how does it work really?

Guess and Check (Example 1)

Example (Group Assignments)

Problem: We want to partition a set of persons in two groups, while avoiding that father and children belong to the same group.
Input: persons and fathers are represented by *person*(_) and *father*(_, _).

% a disjunctive rule to "guess" all the possible assignments

$$group(P, 1) \mid group(P, 2) \text{ :- } person(P).$$

% a constraint to discard unwanted solutions
% i.e., father and children cannot belong to the same group

$$\text{:- } group(P1, G), group(P2, G), father(P1, P2).$$

...so how does it work really?

Guessing part explained

Consider: $group(P, 1) \mid group(P, 2) \text{ :- } person(P).$

If the input is: $person(john).\ \ person(joe).\ \ father(john, joe).$

Then, the answer set of this single-rule program are:

$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 1)\}$
$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 2)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 1)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 2)\}$

Guessing part explained

Consider: $group(P, 1) \mid group(P, 2) \text{ :- } person(P).$

If the input is: $person(john).\ \ person(joe).\ \ father(john, joe).$

Then, the answer set of this single-rule program are:

$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 1)\}$
$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 2)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 1)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 2)\}$

Guessing part explained

Consider: $group(P, 1) \mid group(P, 2) \text{ :- } person(P).$

If the input is: $person(john).\ \ person(joe).\ \ father(john, joe).$

Then, the answer set of this single-rule program are:

$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 1)\}$
$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 2)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 1)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 2)\}$

Checking part explained

Consider: $group(P, 1) \mid group(P, 2) \text{ :- } person(P).$

Now add: $\text{:- } group(P1, G), group(P2, G), father(P1, P2).$

If the input is: $person(john).\ \ person(joe).\ \ father(john, joe).$

The constraint "discards" two non admissible answers:

~~$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 1)\}$~~

$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 2)\}$

$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 1)\}$

~~$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 2)\}$~~

Checking part explained

Consider: $group(P,1) \mid group(P,2) \text{ :- } person(P).$

Now add: $\text{:- } group(P1,G), group(P2,G), father(P1,P2).$

If the input is: $person(john).\ \ person(joe).\ \ father(john,joe).$

The constraint “discards” two non admissible answers:

~~$\{person(john), person(joe), father(john,joe), group(john,1), group(joe,1)\}$~~

$\{person(john), person(joe), father(john,joe), group(john,1), group(joe,2)\}$

$\{person(john), person(joe), father(john,joe), group(john,2), group(joe,1)\}$

~~$\{person(john), person(joe), father(john,joe), group(john,2), group(joe,2)\}$~~

Guess & Check explained

Consider: $group(P, 1) \mid group(P, 2) \text{ :- } person(P).$
$\text{:- } group(P1, G), group(P2, G), father(P1, P2).$

If the input is: $person(john).\ \ person(joe).\ \ father(john, joe).$

The answer sets are:

$\{person(john), person(joe), father(john, joe), group(john, 1), group(joe, 2)\}$
$\{person(john), person(joe), father(john, joe), group(john, 2), group(joe, 1)\}$

G&C = Define search space + specify desired solutions

Guess and Check (Example 2)

Example (3-col)

Problem: Given a graph assign one color out of 3 colors to each node such that two adjacent nodes have always different colors.
Input: a Graph is represented by *node*(_) and *edge*(_, _).

% guess a coloring for the nodes
$(r) \quad col(X, red) \mid col(X, yellow) \mid col(X, green) \ {:\!-}\ node(X).$

% discard colorings where adjacent nodes have the same color
$(c) \quad {:\!-}\ edge(X, Y), col(X, C), \ col(Y, C).$

% NB: answer sets are subset minimal → only one color per node

Guess and Check (Example 2)

Example (3-col)

Problem: Given a graph assign one color out of 3 colors to each node such that two adjacent nodes have always different colors.
Input: a Graph is represented by *node*(_) and *edge*(_, _).

% guess a coloring for the nodes
$(r)\quad col(X, red) \mid col(X, yellow) \mid col(X, green) \;\text{:-}\; node(X).$

% discard colorings where adjacent nodes have the same color
$(c)\quad \text{:-}\; edge(X, Y), col(X, C), col(Y, C).$

% NB: answer sets are subset minimal → only one color per node

Guess and Check (Example 2)

Example (3-col)

Problem: Given a graph assign one color out of 3 colors to each node such that two adjacent nodes have always different colors.
Input: a Graph is represented by *node*(_) and *edge*(_, _).

% guess a coloring for the nodes
$(r) \quad col(X, red) \mid col(X, yellow) \mid col(X, green) \text{ :- } node(X).$

% discard colorings where adjacent nodes have the same color
$(c) \quad \text{:- } edge(X, Y), col(X, C), \; col(Y, C).$

% NB: answer sets are subset minimal → only one color per node

Guess and Check (Example 2)

Example (3-col)

Problem: Given a graph assign one color out of 3 colors to each node such that two adjacent nodes have always different colors.
Input: a Graph is represented by *node*(_) and *edge*(_, _).

% guess a coloring for the nodes
$(r)\quad col(X, red) \mid col(X, yellow) \mid col(X, green) \ \text{:-}\ node(X).$

% discard colorings where adjacent nodes have the same color
$(c)\quad \text{:-}\ edge(X, Y), col(X, C), col(Y, C).$

% NB: answer sets are subset minimal → only one color per node

Guess and Check (Example 3)

Example (Hamiltonian Path)

Problem: Find a path in a Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _), and *start*(_).

% Guess a path
$inPath(X, Y) \mid outPath(X, Y) \text{ :- } edge(X, Y).$ | Guess

% A node can be reached only once |
$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$
$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$ | Check
% All nodes must be reached |
$\text{:- } node(X), \text{not } reached(X).$ |
% The path is not cyclic |
$\text{:- } inPath(X, Y), start(Y).$

$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules
$reached(X) \text{ :- } start(X).$ |

Guess and Check (Example 3)

Example (Hamiltonian Path)

Problem: Find a path in a Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _), and *start*(_).

% Guess a path | Guess
$inPath(X, Y) \mid outPath(X, Y) \text{ :- } edge(X, Y).$

% A node can be reached only once | Check
$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$
$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$
% All nodes must be reached
$\text{:- } node(X), \text{not } reached(X).$
% The path is not cyclic
$\text{:- } inPath(X, Y), start(Y).$

$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules
$reached(X) \text{ :- } start(X).$

Guess and Check (Example 3)

Example (Hamiltonian Path)

Problem: Find a path in a Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _), and *start*(_).

```
% Guess a path                                            | Guess
inPath(X, Y) | outPath(X, Y) :- edge(X, Y).

% A node can be reached only once                         |
:- inPath(X, Y), inPath(X, Y1), Y <> Y1.                  | Check
:- inPath(X, Y), inPath(X1, Y), X <> X1.
% All nodes must be reached                               |
:- node(X), not reached(X).                               |
% The path is not cyclic                                  |
:- inPath(X, Y), start(Y).

reached(X) :- reached(Y), inPath(Y, X).                   | Aux. Rules
reached(X) :- start(X).                                   |
```

Guess and Check (Example 3)

Example (Hamiltonian Path)

Problem: Find a path in a Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _), and *start*(_).

% Guess a path
$inPath(X, Y) \mid outPath(X, Y) \text{ :- } edge(X, Y).$ | Guess

% A node can be reached only once |
$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$ | Check
$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$ |
% All nodes must be reached |
$\text{:- } node(X), \text{not } reached(X).$ |
% The path is not cyclic |
$\text{:- } inPath(X, Y), start(Y).$

$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules
$reached(X) \text{ :- } start(X).$ |

Guess and Check (Example 3)

Example (Hamiltonian Path)

Problem: Find a path in a Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _), and *start*(_).

```
% Guess a path
inPath(X, Y) | outPath(X, Y) :- edge(X, Y).                | Guess

% A node can be reached only once                          |
:- inPath(X, Y), inPath(X, Y1), Y <> Y1.
:- inPath(X, Y), inPath(X1, Y), X <> X1.                   | Check
% All nodes must be reached                                |
:- node(X), not reached(X).                                |
% The path is not cyclic                                   |
:- inPath(X, Y), start(Y).

reached(X) :- reached(Y), inPath(Y, X).                    | Aux. Rules
reached(X) :- start(X).                                    |
```

Guess, Check and Optimize (Example 4)

Example (Traveling Salesman Person)

Problem: Find a path of minimum length in a Weighted Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _, _), and *start*(_).

% Guess a path | Guess

$inPath(X, Y) \mid outPath(X, Y) \text{ :- } edge(X, Y, \_).$

% Ensure that it is Hamiltonian (as before) |

$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$ | Check

$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$ |

$\text{:- } node(X), \text{not } reached(X).\ \text{:- } inPath(X, Y), start(Y).$ |

$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules

$reached(X) \text{ :- } start(X).$ |

% Minimize the sum of distances

$:\sim inPath(X, Y), edge(X, Y, C).\ [C@0, X, Y, C]$ | Optimize

Guess, Check and Optimize (Example 4)

Example (Traveling Salesman Person)

Problem: Find a path of minimum length in a Weighted Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _, _), and *start*(_).

% Guess a path | Guess

$inPath(X, Y) \mid outPath(X, Y) \text{ :- } edge(X, Y, \_).$

% Ensure that it is Hamiltonian (as before) |

$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$ | Check

$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$ |

$\text{:- } node(X), \text{not } reached(X). \text{ :- } inPath(X, Y), start(Y).$ |

$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules

$reached(X) \text{ :- } start(X).$ |

% Minimize the sum of distances

$:\sim inPath(X, Y), edge(X, Y, C). [C@0, X, Y, C]$ | Optimize

Guess, Check and Optimize (Example 4)

Example (Traveling Salesman Person)

Problem: Find a path of minimum length in a Weighted Graph beginning at the starting node which contains all nodes of the graph.
Input: *node*(_) and *edge*(_, _, _), and *start*(_).

% Guess a path | Guess

$inPath(X, Y) \mid outPath(X, Y) \text{ :- } edge(X, Y, \_).$

% Ensure that it is Hamiltonian (as before) | Check

$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$

$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$

$\text{:- } node(X), \text{not } reached(X). \text{ :- } inPath(X, Y), start(Y).$

$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules

$reached(X) \text{ :- } start(X).$

% Minimize the sum of distances

$:\sim inPath(X, Y), edge(X, Y, C). [C@0, X, Y, C]$ | Optimize

And what's about abstract solvers for this lecture?

- ∼ Abstract solvers for disjunctive ASP with bj and learning
- ∼ Abstract solvers for cautious ASP reas. with bj and lear
- × Abstract solvers for ASP with aggregates
- × Abstract solvers for finding "optimal" ASP solutions

And what's about abstract solvers for this lecture?

- $\sim$ Abstract solvers for disjunctive ASP with bj and learning
- $\sim$ Abstract solvers for cautious ASP reas. with bj and lear
- $\times$ Abstract solvers for ASP with aggregates
- $\times$ Abstract solvers for finding "optimal" ASP solutions

And what's about abstract solvers for this lecture?

- ∼ Abstract solvers for disjunctive ASP with bj and learning
- ∼ Abstract solvers for cautious ASP reas. with bj and lear
- × Abstract solvers for ASP with aggregates
- × Abstract solvers for finding "optimal" ASP solutions

And what's about abstract solvers for this lecture?

- $\sim$ Abstract solvers for disjunctive ASP with bj and learning
- $\sim$ Abstract solvers for cautious ASP reas. with bj and lear
- $\times$ Abstract solvers for ASP with aggregates
- $\times$ Abstract solvers for finding "optimal" ASP solutions

And what's about abstract solvers for this lecture?

- $\sim$ Abstract solvers for disjunctive ASP with bj and learning
- $\sim$ Abstract solvers for cautious ASP reas. with bj and lear
- $\times$ Abstract solvers for ASP with aggregates
- $\times$ Abstract solvers for finding "optimal" ASP solutions

Thanks to Francesco Ricca for a preliminary version of these slides