Deep Reasoning in AI with Answer Set Programming

ASP Basics

Francesco Ricca and Mario Alviano

Department of Mathematics and Computer Science

University of Calabria

ESSAI 2024 - Athens

Outline

Deep Reasoning

Learning vs Reasoning

- From the particular to the universal (induction)
- From the universal to the particular (deduction)

Why... this course?

- Spotlight Seminars in AI, June 24, 2022
  → Machine Learning and Logic: Fast and Slow Thinking
  → Prof. Moshe Vardi - Rice University
  `https://www.youtube.com/watch?v=K-wfD5SKaLc`
- "Reasoning" has a problem of marketing and adoption
  → ESSAI23 - Pills of ASP → Deep Reasoning with ASP - ESSAI24

Deep Reasoning

Learning vs Reasoning

- From the particular to the universal (induction)
- From the universal to the particular (deduction)

Why... this course?

- Spotlight Seminars in AI, June 24, 2022

  → Machine Learning and Logic: Fast and Slow Thinking

  → Prof. Moshe Vardi - Rice University

  `https://www.youtube.com/watch?v=K-wfD5SKaLc`
- "Reasoning" has a problem of marketing and adoption

  → ESSAI23 - Pills of ASP → Deep Reasoning with ASP - ESSAI24

Deep Reasoning

Learning vs Reasoning

- From the particular to the universal (induction)
- From the universal to the particular (deduction)

Why... this course?

- Spotlight Seminars in AI, June 24, 2022

  → Machine Learning and Logic: Fast and Slow Thinking

  → Prof. Moshe Vardi - Rice University

  `https://www.youtube.com/watch?v=K-wfD5SKaLc`

- "Reasoning" has a problem of marketing and adoption

  → ESSAI23 - Pills of ASP → Deep Reasoning with ASP - ESSAI24

Context

Answer Set Programming (ASP) [BET11]

- Declarative programming paradigm
- Non-monotonic reasoning and logic programming
- Stable model semantics [GL91]

Expressive KR Language

- Roots in Datalog and Nonmonotonic Logic
- Default negation, Disjunction, Aggregates,
- Hard and Weak constraint, ...
- Basic ASP models up to Δ_3^P [DEGV01]

  $\rightarrow$ i.e., problems not (polynomially) translatable to SAT or CSP

Context (2)

Robust and efficient implementations

- Clasp, Wasp, lp2*, DLV, Cmodels, IDP, etc [GLM+18].
- Performance improvements in the last years [GMR17]

Applications in several fields

- Artificial Intelligence, Knowledge Representation & Reas.,
- Information Integration, Bioinformatics, Robotics... industrial ones!
- see [EGL16]

Goals of the course

Introduce ASP from the basics to sophisticated use-cases

1. Illustrate the language of ASP[1]
2. Show how to model and solve problems with ASP
3. Provide details on the working principle of ASP systems
4. Show how to use ASP systems proficiently
5. Present some recent Explainable AI technology
6. Secure and modular development with ASP

[1] The coverage is not extensive, and may reflect our own biased view.

Presentation roadmap

The language of ASP is:

Datalog

+ Default negation
+ Disjunction
+ Integrity Constraints
+ Weak Constraints
+ Aggregate atoms
+ Choice Rules

Presentation roadmap

The language of ASP is:

Datalog

+ Default negation
+ Disjunction
+ Integrity Constraints
+ Weak Constraints
+ Aggregate atoms
+ Choice Rules
+ Some other solver-specific extensions

What is Datalog?

Datalog

- A logic language for querying databases

  $\rightarrow$ The name is de combination of “Data + Logic”
- Overcomes some limits of Relational Algebra/ SQL

  $\rightarrow$ Recursive definitions
- Deductive database applications, query answering

  $\rightarrow$ The basic fragment of ASP

Datalog Syntax

Rule

$$head(\overline{H}) \;\text{:-}\; body_1(\overline{X_1}), \ldots, body_n(\overline{X_n}).$$

Intuitive meaning

"Infer $head(\overline{h})$ if $body_1(\overline{x_1}), \ldots, body_n(\overline{x_n})$ is true"

Terms, Variables, Atoms

- Numbers, Strings and Variables (Prolog-like syntax)
- Variables occur in some body atom (Safety)
- Atoms: $head(\overline{h})$, $body_1(\overline{x_1})$, ..., $body_n(\overline{x_n})$

Datalog Syntax

Example

Program and query:

$father(X) :- parent(X, Y), male(X).$ father(X)?

Database:

$male(rob).$ ← Facts have empty body
$parent(rob, ann).$ ← symbol :- omitted
$parent(mary, ann).$ ← "True by definition"
$parent(lucy, terry).$

Query Result:

$father(rob).$

Semantics by Example

Example

$father(X) \text{ :- } parent(X, Y), male(X).$

$male(rob).\ parent(rob, ann).\ parent(mary, ann).$
$parent(lucy, terry).$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = I^0 \cup \{\ male(rob),\ parent(rob, ann),\ parent(mary, ann),\ parent(lucy, terry)\}$
3. $I^2 = I^1 \cup \{father(rob)\}$
4. *No match is possible given* $I^2 = I$... STOP!

Result:

$$I = \{male(rob),\ parent(rob, ann),\ parent(mary, ann),\ parent(lucy, terry),\ father(rob)\}$$

Semantics by Example

Example

$father(X) \text{ :- } parent(X,Y), male(X).$

$male(rob).\ parent(rob,ann).\ parent(mary,ann).$
$parent(lucy,terry).$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = I^0 \cup \{\ male(rob),\ parent(rob,ann),\ parent(mary,ann),\ parent(lucy,terry)\}$
3. $I^2 = I^1 \cup \{father(rob)\}$
4. No match is possible given $I^2 = I$... STOP!

Result:

$I = \{male(rob),\ parent(rob,ann),\ parent(mary,ann),\ parent(lucy,terry),\ father(rob)\}$

Semantics by Example

Example

$father(X) \text{ :- } parent(X, Y), male(X).$

$male(rob).\ parent(rob, ann).\ parent(mary, ann).$
$parent(lucy, terry).$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = I^0 \cup \{ male(rob),\ parent(rob, ann),$
   $parent(mary, ann),\ parent(lucy, terry)\}$
3. $I^2 = I^1 \cup \{father(rob)\}$
4. *No match is possible given* $I^2 = I$... STOP!

Result:

$I = \{male(rob),\ parent(rob, ann),\ parent(mary, ann),$
$parent(lucy, terry),\ father(rob)\}$

Semantics by Example

Example

$father(X) \coloneq parent(X, Y), male(X).$

$male(rob).\ parent(rob, ann).\ parent(mary, ann).$
$parent(lucy, terry).$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = I^0 \cup \{\ male(rob),\ parent(rob, ann),\ parent(mary, ann),\ parent(lucy, terry)\}$
3. $I^2 = I^1 \cup \{father(rob)\}$
4. *No match is possible given* $I^2 = I$... STOP!

Result:

$I = \{male(rob),\ parent(rob, ann),\ parent(mary, ann),\ parent(lucy, terry),\ father(rob)\}$

Fully declarative language

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes (transitive closure of *edge*).

Can you write an SQL[a]??

[a] Using a Select - Project - Join query, or say SQL92

Fully declarative language

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes (transitive closure of *edge*).

% if there is an edge from X to Y, then X is reachable from Y
$reachable(X, Y) :- edge(X, Y)$

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

Fully declarative language

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes (transitive closure of *edge*).

% if there is an edge from X to Y, then X is reachable from Y
$reachable(X, Y) :- edge(X, Y)$

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

Intuitive meaning: *(bottom-up evaluation)*

→ *Start with the facts in the DB*
→ *Iteratively derive facts from rules until no new fact is derived*
→ *Obtain the **unique minimal** (perfect, well-founded, stable...) **model**!*

Fully declarative language

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes (transitive closure of *edge*).

% if there is an edge from X to Y, then X is reachable from Y
$reachable(X, Y) :- edge(X, Y)$

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

Example (Fully Declarative Language!)

% if there is an edge from X to Y, then X is reachable from Y
$reachable(X, Y) :- edge(X, Y)$

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

Fully declarative language

Example (Reachability)

Input: a graph encoded by relation *edge*(_, _).
Problem: Find all pairs of reachable nodes (transitive closure of *edge*).

% if there is an edge from X to Y, then X is reachable from Y
$reachable(X, Y) :- edge(X, Y)$

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

Example (Atom's and Rule's order is immaterial!)

% Reachability is transitive
$reachable(X, Y) :- edge(Z, Y), reachable(X, Z)$
$\updownarrow$
% if there is an edge from X to Y, then X is reachable from Y
$reachable(X, Y) :- edge(X, Y)$

Exercise limits (1)

Given the following relational database schema

- $beers(beername*, manufacturer)$
- $sells(bar*, beername*, price)$
- $associate(bar, bar)$

(* indicates primary key)

Write (if possible) in SQL^{92}, and Datalog

1. Manufacturers of beers sold by “John’s bar”
2. Beers sold by "John’s bar" that are not sold by “Annie’s” bar
3. Bars that sell more than three beers
4. Bars that sell exactly two beers
5. Number of beers sold by “John’s bar”
6. Bars that are associated trough a chain of bar associations to “John’s bar”

Presentation roadmap

The language of ASP is:

Datalog ← Done!

- \+ Default negation
- \+ Disjunction
- \+ Integrity Constraints
- \+ Weak Constraints
- \+ Aggregate atoms
- \+ Choice Rules

Default Negation

Limits of Datalog

- Datalog programs are positive
- Already is not enough for expressing simple queries!

Example (Anti join)

% The airports that are **not** reachable from El Paso

Can you write a Datalog query??

Default Negation

Limits of Datalog

- Datalog programs are positive
- Already is not enough for expressing simple queries!

Example (Anti join)

% The airports that are **not** reachable from El Paso

You need some form of negation!

Default Negation

Limits of Datalog

- Datalog programs are positive
- Already is not enough for expressing simple queries!

Example (Anti join)

% The airports that are **not** reachable from El Paso

$reachableFromElPaso(X) \text{ :- } reachable("ElPaso", X).$

$query(X) \text{ :- } airport(X), \text{not } reachableFromElPaso(X)$

Default Negation

Often, it is desirable to express negation in the following sense:

"If we do not have evidence that X holds, conclude Y."

This is expressed by default negation: the operator **not.**

Example (Cross railroad)

An agent could act according to the following rule:

% If the grass is not wet then it did not rain.

did_not_rain :– not *wet_grass*.

Negation might be problematic (1)

Example (Bad negation (1))

$$p(X) :- q(X), \text{not } p(X).$$
$$q(1).\ q(2).$$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = \{q(1),\ q(2)\}$
3. $I^2 = \{q(1),\ q(2),\ p(1),\ p(2).\}$
4. $I^3 = \{q(1),\ q(2)\} = I^1$

Negation might be problematic (1)

Example (Bad negation (1))

$p(X) :- q(X), \text{not } p(X).$
$q(1).\ q(2).$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = \{q(1),\ q(2)\}$
3. $I^2 = \{q(1),\ q(2),\ p(1),\ p(2).\}$
4. $I^3 = \{q(1),\ q(2)\} = I^1$

Negation might be problematic (1)

Example (Bad negation (1))

$$p(X) :\!-\ q(X), \text{not } p(X).$$
$$q(1).\ q(2).$$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = \{q(1),\ q(2)\}$
3. $I^2 = \{q(1),\ q(2),\ p(1),\ p(2).\}$ $\leftarrow$ What?!?
4. $I^3 = \{q(1),\ q(2)\} = I^1$

Negation might be problematic (1)

Example (Bad negation (1))

$p(X) \text{ :- } q(X), \text{not } p(X).$

$q(1).\ q(2).$

Evaluation:

1. $I^0 = \{\}$
2. $I^1 = \{q(1),\ q(2)\}$
3. $I^2 = \{q(1),\ q(2),\ p(1),\ p(2).\}$
4. $I^3 = \{q(1),\ q(2)\} = I^1$ So... it does not work!

Negation might be problematic (2)

Example (Bad negation (2))

$a(X) :- \text{not } b(X), d(X)$

$b(X) :- \text{not } a(X), d(X)$

$d(1).$

Try to apply the bottom-up evaluation strategy...

Again, it does not work... :-(

Negation might be problematic (2)

Example (Bad negation (2))

$a(X) :- \text{not } b(X), d(X)$

$b(X) :- \text{not } a(X), d(X)$

$d(1).$

Try to apply the bottom-up evaluation strategy...

Again, it does not work... :-(

Stratified Programs

Definition (Dependency Graph)

Given program P, the graph $DG(P) =< V, E >$ s.t.:

- $V = \{p \mid p \text{ is predicate occurring in } P\}$
- $E = \{(p, q, +) \mid p \text{ is in the head and } q \text{ is positive in the body of } r \in P\}$

  $\cup$

  $\{(p, q, -) \mid p \text{ is in the head and } q \text{ is negative in the body of } r \in P\}$

Definition (Recursive Program)

P is recursive if $DG(P)$ is cyclic.

Definition (Stratified Program)

P is stratified if no cycle in $DG(P)$ contains a negative edge.

Examples: stratified programs

Example (Non stratified)

$$p(X) :- q(X), \text{not } p(X).$$

Example (Non stratified)

$$a(X) :- \text{not } b(X), d(X)$$
$$b(X) :- \text{not } a(X), d(X)$$

Example (Cyclic, stratified)

$$reachable(X, Y) :- edge(X, Y)$$
$$reachable(X, Z) :- reachable(X, Z), edge(Z, Y)$$
$$reachableFromElPaso(X) :- reachable(\text{"}ElPaso\text{"}, X).$$
$$query(X) :- airport(X), \text{not } reachableFromElPaso(X)$$

Stratified negation (1)

Example (Good negation)

1. $reachable(X, Y) :- edge(X, Y)$
2. $reachable(X, Z) :- reachable(X, Z), edge(Z, Y)$
3. $reachableFromElPaso(X) :- reachable(''ElPaso'', X).$
4. $query(X) :- airport(X), \text{not } reachableFromElPaso(X)$

Evaluation:

- Evaluate stratum by stratum
  $\rightarrow$ when all the information is present when negation is evaluated
- Stratified Datalog
  $\rightarrow$ One set of answers, one model!

Stratified negation (1)

Example (Good negation)

1. $reachable(X, Y) :- edge(X, Y)$
2. $reachable(X, Z) :- reachable(X, Z), edge(Z, Y)$
3. $reachableFromElPaso(X) :- reachable("ElPaso", X).$
4. $query(X) :- airport(X), \mathrm{not}\ reachableFromElPaso(X)$

Evaluation:

1. Stratum 1:
   - → Rules (1),(2), and (3)
   - → Now *reachableFromElPaso*(·) fully computed!!
2. Stratum 2
   - → Rule (4)

Exercise limits (1)

Given the following relational database schema

- $beers(beername*, manufacturer)$
- $sells(bar*, beername*, price)$
- $associate(bar, bar)$

(* indicates primary key)

Write (if possible) in SQL^{92}, and Stratified Datalog

1. Manufacturers of beers sold by “John’s bar”
2. Beers sold by "John’s bar" that are not sold by “Annie’s” bar
3. Bars that sell more than three beers
4. Bars that sell exactly two beers
5. Number of beers sold by “John’s bar”
6. Bars that are associated trough a chain of bar associations to “John’s bar”

Arithmetic Expressions and Builtins

Applications may require to use numbers, make comparisons, etc.

Example (Comparison operator)

$$older(P1, P2) \text{ :- } person(P1, Age1), person(P2, Age2), Age1 > Age2.$$

Example (Fibonacci numbers)

$fib(0, 1).$
$fib(1, 1).$

$$fib(N + 2, Y1 + Y2) \text{ :- } fib(N, Y1), fib(N + 1, Y2).$$

WARNING: *For recursive definitions an upper bound for integers (system setting) or a domain has to be specified.*

Intuitive definition of model

Definition (Informal)

- **Interpretation:** A set I of *true* ground atoms
- **Satisfaction:** A rule r is satisfied w.r.t. I if the head is true whenever all the body literals are true
- **Model:** An interpretation that satisfies all (the instantiations of the) rules

Intuitive definition of model

Example (Models)

Given:

$$a \,{:}\!-\, b, c.$$
$$c \,{:}\!-\, d.$$
$$d.$$

Interpretations and Models:

- $I_1 = \{b, c, d\}$
- $I_2 = \{a, b, c, d\}$
- $I_3 = \{c, d\}$

Intuitive definition of model

Example (Models)

Given:

$$a \text{ :- } b, c.$$
$$c \text{ :- } d.$$
$$d.$$

Interpretations and Models:

- $I_1 = \{b, c, d\} \leftarrow$ not a model!
- $I_2 = \{a, b, c, d\} \leftarrow$ model!
- $I_3 = \{c, d\} \leftarrow$ minimal model!

Unrestricted negation (1)

Example (Stable models)

$$a :- \text{not } b$$

$$b :- \text{not } a$$

Some Observation:

- What if we assume *a* is true and *b* is false? ..OK!
- What if we assume *a* is false and *b* is true? ..OK!
- There is no problem if you fix a “good interpretation”!

Unrestricted negation (1)

Example (Stable models)

$$a :- \text{not } b$$

$$b :- \text{not } a$$

Some Observation:

- What if we assume *a* is true and *b* is false? ..OK!
- What if we assume *a* is false and *b* is true? ..OK!
- There is no problem if you fix a “good interpretation”!

Unrestricted negation (1)

Example (Stable models)

a :– not b

b :– not a

Some Observation:

- What if we assume *a* is true and *b* is false? ..OK!
- What if we assume *a* is false and *b* is true? ..OK!
- There is no problem if you fix a “good interpretation”!

Unrestricted negation (1)

Example (Stable models)

a :– not b

b :– not a

Some Observation:

- What if we assume *a* is true and *b* is false? ..OK!
- What if we assume *a* is false and *b* is true? ..OK!
- There is no problem if you fix a “good interpretation”!

Unrestricted negation (2)

We know that

- Positive programs have a deterministic behavior
- Some assumptions can be satisfactory

Gelfond-Lifschitz Reduct

- Remove rules with false negative literals in the body
- Remove the remaining negative literals

Stable Model or Answer Set (step by step)

- Given a model m for P
- Compute the reduct P^m
- m is stable if it is the model of P^m

Example 1

Example (Reduct)

Program:

$a :- d, \text{not}\ \ b.$
$b :- \text{not}\ \ d.$

$d.$

Consider: $I = \{a, d\}$

Reduct:

$a :- d.$
$d.$

I is an answer set of P^I and therefore it is an answer set of P.

Example 2

Example (Stable models)

$a \text{ :- not } b \qquad b \text{ :- not } a$

Let's check all possibilities:

- Assume $\{\}$ is not a model
- Assume $\{a, b\}$ is a model but is not stable!
- Assume $\{a\}$, is model, actually a stable one!
- Assume $\{b\}$, is model, actually a stable one!

Example 2

Example (Stable models)

$a \coloneq \text{not } b$ $\qquad$ $b \coloneq \text{not } a$

Let's check all possibilities:

- Assume $\{\}$ is not a model
- Assume $\{a, b\}$ is a model but is not stable!
- Assume $\{a\}$, is model, actually a stable one!
- Assume $\{b\}$, is model, actually a stable one!

Example 2

Example (Stable models)

$a :- \text{not } b \qquad b :- \text{not } a$

Let's check all possibilities:

- Assume $\{\}$ is not a model
- Assume $\{a, b\}$ is a model but is not stable!
- Assume $\{a\}$, is model, actually a stable one!
- Assume $\{b\}$, is model, actually a stable one!

Example 2

Example (Stable models)

$a :\!-\ \text{not}\ b \qquad b :\!-\ \text{not}\ a$

Let's check all possibilities:

- Assume $\{\}$ is not a model
- Assume $\{a, b\}$ is a model but is not stable!
- Assume $\{a\}$, is model, actually a stable one!
- Assume $\{b\}$, is model, actually a stable one!

Example 2

Example (Stable models)

$a :- \text{not } b \qquad b :- \text{not } a$

Let's check all possibilities:

- Assume $\{\}$ is not a model
- Assume $\{a, b\}$ is a model but is not stable!
- Assume $\{a\}$, is model, actually a stable one!
- Assume $\{b\}$, is model, actually a stable one!

Example 2

Example (Stable models)

$a \text{ :- not } b \qquad b \text{ :- not } a$

Let's check all possibilities:

- Assume $\{\}$ is not a model
- Assume $\{a, b\}$ is a model but is not stable!
- Assume $\{a\}$, is model, actually a stable one!
- Assume $\{b\}$, is model, actually a stable one!

Example 3

Example

Program:

$a :- \text{not } b.$

Answer Set: $\{a\}$

Example

Program:

$a :- \text{not } b.$
$b :- \text{not } a.$
$c :- b.$
$c :- a.$

Answer Sets: $\{a, c\}, \{b, c\}$

Example 4

Example

Program:

$$a \text{ :- } \text{not } a.$$

Answer Set: no answer set!

Example

Program:

$$a \text{ :- } \text{not } b.$$
$$b \text{ :- } \text{not } a.$$
$$f \text{ :- } b, \text{not } f$$

Answer Set: $\{a\}$

Supported Models and Answer Sets (1)

Definition (Supported Model)

A model M is supported if for each $a \in M$ there exist rule $r \in P$ such that a is the head and $\forall b$ in the body, b is true w.r.t. M

Intuition:
Something is true if there is a rule “supporting” its truth.

Theorem:

Answer sets are supported models

Supported Models and Answer Sets (2)

Example (Inverse does not hold.)

Program:

$a \text{ :- } a.$

Models: $\{\}, \{a\}$ ← both are supported

Answer Set: $\{\}$

→ **Circular support is not allowed!**

→ **Empty answer set is fine!**

Supported Models and Answer Sets (2)

Example (Inverse does not hold.)

Program:

$a :\!- a.$

Models: $\{\}, \{a\}$ ← both are supported

Answer Set: $\{\}$

→ **Circular support is not allowed!**

→ **Empty answer set is fine!**

Supported Models and Answer Sets (2)

Example (Inverse does not hold.)

Program:

$a \coloneq a.$

Models: $\{\}, \{a\} \leftarrow$ both are supported

Answer Set: $\{\}$

$\rightarrow$ **Circular support is not allowed!**

$\rightarrow$ **Empty answer set is fine!**

Unfounded Sets and Answer Sets (intuition)

Unfounded Set:

A set of ground atoms X is an unfounded set if, for each rule r s.t. $H(r) \in X$, one of the following conditions hold

1. the body of r is false, or
2. some literal in the positive body belongs to X

Example: $a :\!- a.$ and $X = \{a\}$. is unfounded!

Theorem:

Answer sets are unfounded-free interpretations, i.e., no subset is unfounded.

Unfounded Sets and Answer Sets (intuition)

Unfounded Set:

A set of ground atoms X is an unfounded set if, for each rule r s.t. $H(r) \in X$, *one of the following conditions hold*

1. the body of r is false, or
2. some literal in the positive body belongs to X

Example: $a \mathrel{:-} a.$ and $X = \{a\}.$ is unfounded!

Theorem:

Answer sets are unfounded-free interpretations, i.e., no subset is unfounded.

Multiple models...

Observation:

- Several stable models might represent several possible solutions
- Stable models are sets... answer sets
- No answer set... no solution

Idea [Lif99]:

1. Represent a computational problem by a Logic program
2. Answer sets correspond to problem solutions
3. Use an ASP solver to find these solutions

Multiple models...

Observation:

- Several stable models might represent several possible solutions
- Stable models are sets... answer sets
- No answer set... no solution

Idea [Lif99]:

1. Represent a computational problem by a Logic program
2. Answer sets correspond to problem solutions
3. Use an ASP solver to find these solutions

Exercise SAT

Given a propositional formula Φ in 3 CNF, compute an assignment to variables that satisfies Φ if it exists.

Write a logic program $P(\Phi)$ such that answer sets of $P(\Phi)$ correspond to satisfying assignments of Φ

Exercise SAT

Given a propositional formula Φ in 3 CNF, compute an assignment to variables that satisfies Φ if it exists.

Write a logic program $P(\Phi)$ such that answer sets of $P(\Phi)$ correspond to satisfying assignments of Φ

Hints...

Example (Ingredient 1)

$$a :- \text{not } b$$

$$b :- \text{not } a$$

Example (Ingredient 2)

$$p :- \text{not } p$$

Example (A simple 3SAT formula)

$$(A \lor B \lor \neg C) \land (\neg A \lor B \lor C) \land (\neg A \lor \neg B \lor \neg C)$$

Presentation roadmap

The language of ASP is:

Datalog ← Done!

+ Default negation ← Done!
+ Disjunction
+ Integrity Constraints
+ Weak Constraints
+ Aggregate atoms
+ Choice Rules

Disjunction

There is a more intuitive way of expressing multiple models.

Often we just desire to express disjunctive information.

"We want to model several alternative scenarios"

This is expressed by disjunctive rules: the operator $|$.

Example (Datalog + Disjunction)

% Disjunctive knowledge:
%"A parent *P* is either a father or a mother"

$mother(P, S) \mid father(P, S) \text{ :- } parent(P, S).$

Constraints

Many models $\rightarrow$ need to express properties of solutions

"Discard a solution If that conjunction holds."

Constrains are *rules with empty (false) head*

Example (Parent of himself)

% "Ensure that none is the parent of himself."

$:\!-\ mother(P, P). \quad :\!-\ father(P, P).$

ASP Syntax

Rule: $\underbrace{a_1 \mid \ldots \mid a_n}_{head} \;\text{:-}\; \underbrace{b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m.}_{body}$

Atoms and Literals: a_i , b_i, not b_i
Positive Body: $b_1, \ldots, b_k$
Negative Body: not $b_{k+1}, \ldots,$ not b_m.

Fact: A rule with empty body
Constraint: A rule with empty head

Variables: allowed in atom's arguments

- Must occur in the positive body (Safety)
- Are placeholders for constants

ASP Syntax

Rule: $\underbrace{a_1 \mid \ldots \mid a_n}_{head} \;\text{:-}\; \underbrace{b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m.}_{body}$

Atoms and Literals: a_i , b_i, not b_i
Positive Body: $b_1, \ldots, b_k$
Negative Body: not $b_{k+1}, \ldots,$ not b_m.

Fact: A rule with empty body
Constraint: A rule with empty head

Variables: allowed in atom's arguments

- Must occur in the positive body (Safety)
- Are placeholders for constants

Informal Semantics

Rule:

$$\underbrace{a_1 \mid \ldots \mid a_n}_{head} \text{ :- } \underbrace{b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m.}_{body}$$

Informal Semantics:

"If all $b_1, \ldots, b_k$ are true and all $b_{k+1}, \ldots, b_m$ are not true, then at least one among $a_1, \ldots, a_n$ is true".

Informal Semantics

Rule: $\underbrace{a_1 \mid \ldots \mid a_n}_{head} \text{ :- } \underbrace{b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m.}_{body}$

Informal Semantics:

"If all $b_1, \ldots, b_k$ are true and all $b_{k+1}, \ldots, b_m$ are not true, then at least one among $a_1, \ldots, a_n$ is true".

Example (Disjunction + Constraint)

%"A a node is either in the set or out of the set"
$inSet(N) \mid outSet(N) \text{ :- } node(N).$

% Constrains: "Two adjacent nodes cannot be in the set."
$\text{:- } inSet(N1), inSet(N2), edge(N1, N2).$

Informal Semantics: Variables

Handling variables

- Variables are placeholders for constants
- ***Grounding***: *"Replace variables by constants in all possible ways"*

Example (Ground Instantiation)

Consider:

$isInterestedinASP(X) \mid isCurious(X) \text{ :- } attendsASP(X).$
$attendsASP(john). \quad attendsASP(mary).$

Instantiation:

$isInterestedinASP(john) \mid isCurious(john) \text{ :- } attendsASP(john).$
$isInterestedinASP(mary) \mid isCurious(mary) \text{ :- } attendsASP(mary).$
$attendsASP(john). \quad attendsASP(mary).$

Informal Semantics: Minimal Models

Example (Disjunction)

isInterestedinASP(*john*) | *isCurious*(*john*) :– *attendsASP*(*john*).
attendsASP(*john*).

Two (minimal) models encoding two plausible scenarios:

- M_1:{*isInterestedinASP*(*john*), *attendsASP*(*john*).}
- M_2:{*isCurious*(*john*), *attendsASP*(*john*).}

Example (Constraints)

isInterestedinASP(*john*) | *isCurious*(*john*) :– *attendsASP*(*john*).
:– *hatesASP*(*john*), *isInterestedinASP*(*john*).
attendsASP(*john*). *hatesASP*(*john*).

Only one plausible scenario:

- ~~M_1:{*isInterestedinASP*(*john*), *attendsASP*(*john*), *hatesASP*(*john*).}~~
- M_2:{*isCurious*(*john*), *attendsASP*(*john*), *hatesASP*(*john*).}

Informal Semantics: Minimal Models

Example (Disjunction)

isInterestedinASP(john) | isCurious(john) :– attendsASP(john).
attendsASP(john).

Two (minimal) models encoding two plausible scenarios:

- M_1:{*isInterestedinASP(john), attendsASP(john).*}
- M_2:{*isCurious(john), attendsASP(john).*}

Example (Constraints)

isInterestedinASP(john) | isCurious(john) :– attendsASP(john).
:– hatesASP(john), isInterestedinASP(john).
attendsASP(john). hatesASP(john).

Only one plausible scenario:

- ~~M_1:{*isInterestedinASP(john), attendsASP(john), hatesASP(john).*}~~
- M_2:{*isCurious(john), attendsASP(john), hatesASP(john).*}

Informal Semantics (Disjunction and minimality)

Semantics of disjunction is:

- Minimal

  $a \mid b \mid c. \;\Rightarrow \{a\}, \{b\}, \{c\}$

- Actually subset minimal

  $a \mid b.$

  $a \mid c. \;\Rightarrow \{a\}, \{b, c\}$

- ...but *not exclusive*

  $a \mid b.$

  $a \mid c.$

  $b \mid c. \;\Rightarrow \{a, b\}, \{a, c\}, \{b, c\}$

Informal Semantics (Disjunction and minimality)

Semantics of disjunction is:

- Minimal

  $a \,|\, b \,|\, c. \;\Rightarrow \{a\}, \{b\}, \{c\}$

- Actually subset minimal

  $a \,|\, b.$

  $a \,|\, c. \;\Rightarrow \{a\}, \{b, c\}$

- ...but *not exclusive*

  $a \,|\, b.$

  $a \,|\, c.$

  $b \,|\, c. \;\Rightarrow \{a, b\}, \{a, c\}, \{b, c\}$

Informal Semantics (Disjunction and minimality)

Semantics of disjunction is:

- Minimal

  $a \,|\, b \,|\, c. \;\Rightarrow \{a\}, \{b\}, \{c\}$

- Actually subset minimal

  $a \,|\, b.$

  $a \,|\, c. \;\Rightarrow \{a\}, \{b, c\}$

- ...but *not exclusive*

  $a \,|\, b.$

  $a \,|\, c.$

  $b \,|\, c. \;\Rightarrow \{a, b\}, \{a, c\}, \{b, c\}$

Exercise SAT

Given a propositional formula Φ in 3 CNF, compute an assignment to variables that satisfies Φ if it exists.

Write a disjunctive ASP program $P(\Phi)$ such that answer sets of $P(\Phi)$ correspond to satisfying assignments of Φ

Formal Semantics: just a recap

Answer Set Semantics (aka stable models semantics)

1. Instantiation
2. Positive (Ground) Programs
3. Negative Programs
   - via Gelfong & Lifschitz Reduct [GL91]

Formal Semantics: just a recap

Answer Set Semantics (aka stable models semantics)

1. Instantiation (get rid of variables)
2. Positive (Ground) Programs (minimal models)
3. Negative Programs (stable models)
    - via Gelfong & Lifschitz Reduct [GL91]

Presentation roadmap

The language of ASP is:

Datalog ← Done!

+ Default negation ← Done!
+ Disjunction ← Done!
+ Integrity Constraints ← Done!
+ Weak Constraints
+ Aggregate atoms
+ Choice Rules

Optimum Models

Weak Constraints

- Express desiderata
- *Constraints which should possibly be satisfied (as soft constraints in CSP)*

Syntax $:\sim body(\overline{X}, \overline{Y}).\ [w@p, \overline{X}]$

Intuitive meaning "set *body* as false, if possible"

Optimum Models

Weak Constraints

- Express desiderata
- *Constraints which should possibly be satisfied (as soft constraints in CSP)*

Syntax $:\sim body(\overline{X}, \overline{Y}).\ [w@p, \overline{X}]$

Weight and Priority Level

- higher weights/priorities $\Rightarrow$ higher importance
- "@p" can be omitted

"Minimize the sum of the weights of the violated constraints in the highest priority level, and so on"

Declarative specification of optimization problems

Weak Constraints Example

Example (Exams Scheduling)

Problem: Assign course exams to 3 time slots avoiding overlapping of exams of courses with common students.

Strict Solution:

$assign(X, s1) \mid assign(X, s2) \mid assign(X, s3) \text{ :- } course(X).$
% No overlap is admitted!"
$\text{:- } assign(X, S), assign(Y, S), commonStudents(X, Y, N), N > 0.$

Optimal Solution:

$assign(X, s1) \mid assign(X, s2) \mid assign(X, s3) \text{ :- } course(X).$
% If overlapping is unavoidable, then reduce it "As Much As Possible"
$:\sim assign(X, S), assign(Y, S), commonStudents(X, Y, N), N > 0. \quad [N@0]$

NB: Answer sets minimizing the total number of "lost" exams are preferred.

Aggregates

Aggregate atoms

- Express functions calculated over sets of elements
- Often needed by applications
- Similar to aggregates in SQL

$$L_g <_1 f\{S\} <_2 U_g$$

$$5 < \#count\{EmpId : emp(EmpId, male, Skill, Salary)\} \leq 10$$

The atom is true if the number of male employees is greater than 5 and does not exceed 10.

Aggregate Example (1)

Example (Count beers)

% Number of beers sold by "John's bar"

$numBeers(X) :\!-\ \#count\{B : beers(B, \_), sells(john, B, \_)\} = X.$

Example (Sum salaries)

% Sum of salaries of team members

$sumSal(S) :\!-\ \#sum\{Sa, I : emp(I, Sa), teamMember(I)\} = S.$

Aggregate Example (2)

Example (Team Building)

% An employee is either included in the team or not

$inTeam(I) \mid outTeam(I) \text{ :- } emp(I, Sx, Sk, Sa).$

% The team consists of a certain number of employees

$\text{:- } nEmp(N),\ \#count\{I : inTeam(I)\} \neq N.$

% At least a given number of different skills must be present in the team

$\text{:- } nSkill(M), \#count\{Sk : emp(I, Sx, Sk, Sa), inTeam(I)\} \leq M.$

% The sum of the salaries of the team must not exceed the given budget

$\text{:- } budget(B),\ \#sum\{Sa, I : emp(I, Sx, Sk, Sa), inTeam(I)\} > B.$

% The salary of each individual employee is within a specified limit

$\text{:- } maxSal(M),\ \#max\{Sa : emp(I, Sx, Sk, Sa), inTeam(I)\} > M.$

Choice Rules

Syntax

$$\{a(\overline{X}) : l_1(\overline{X_1}), \cdots, l_k(\overline{X_k})\}\Theta u \ \text{:-}\ b_1(\overline{Y_1}), \cdots, b_n(\overline{X_n}).$$

Intuitive meaning: (Direct modeling of the search space)

"if the body of the rule is true, choose as true an arbitrary subset of n atoms $a(\overline{X})$, such that $l_1(\overline{X_1}), \cdots, l_k(\overline{X_k})$ are true, and the expression $n\Theta u$ is satisfied"

Example (Assign colors)

% Choose exactly one color per each node

$\{col(X, C) : color(C)\} = 1 \text{ :- } node(X).$
$color(red).\ color(blue).\ node(1).\ node(2)$

Choice Rules

Syntax

$$\{a(\overline{X}) : l_1(\overline{X_1}), \cdots, l_k(\overline{X_k})\}\Theta u \ {:}\!- b_1(\overline{Y_1}), \cdots, b_n(\overline{X_n}).$$

Intuitive meaning: (Direct modeling of the search space)

"if the body of the rule is true, choose as true an arbitrary subset of n atoms $a(\overline{X})$, such that $l_1(\overline{X_1}), \cdots, l_k(\overline{X_k})$ are true, and the expression $n\Theta u$ is satisfied"

Example (Assign colors)

% Choose exactly one color per each node

$\{col(1, red), col(1, blue)\} = 1 \ {:}\!- \ node(1).$

$\{col(2, red), col(2, blue)\} = 1 \ {:}\!- \ node(2).$

$color(red).\ color(blue).\ node(1).\ node(2)$

Choice Rules

Syntax

$$\{a(\overline{X}) : l_1(\overline{X_1}), \cdots, l_k(\overline{X_k})\}\Theta u \;\text{:-}\; b_1(\overline{Y_1}), \cdots, b_n(\overline{X_n}).$$

Intuitive meaning: (Direct modeling of the search space)

"if the body of the rule is true, choose as true an arbitrary subset of n atoms $a(\overline{X})$, such that $l_1(\overline{X_1}), \cdots, l_k(\overline{X_k})$ are true, and the expression $n\Theta u$ is satisfied"

Example (Assign colors)

% Choose exactly one color per each node
$\{col(1, red), col(1, blue)\} = 1 \text{ :- } node(1).$
$\{col(2, red), col(2, blue)\} = 1 \text{ :- } node(2).$
$color(red).\ color(blue).\ node(1).\ node(2)$

Admissible choices (combine and get the answer sets):

$\{col(1, red)\},\ \{col(1, blue).\},\ \{col(2, red).\},\ \{col(3, blue).\}$

Aggregate Example (2)

Example (Team Building)

% Select a team of exactly a given number of employees
$\{inTeam(I) : emp(I, Sx, Sk, Sa)\} = N \text{ :- } nEmp(N).$

% At least a given number of different skills must be present in the team
$\text{:- } nSkill(M), \#count\{Sk : emp(I, Sx, Sk, Sa), inTeam(I)\} \leq M.$

% The sum of the salaries of the team must not exceed the given budget
$\text{:- } budget(B),\ \#sum\{Sa, I : emp(I, Sx, Sk, Sa), inTeam(I)\} > B.$

% The salary of each individual employee is within a specified limit
$\text{:- } maxSal(M),\ \#max\{Sa : emp(I, Sx, Sk, Sa), inTeam(I)\} > M.$

Presentation roadmap

The language of ASP is:

Datalog ← Done!

+ Default negation ← Done!
+ Disjunction ← Done!
+ Integrity Constraints ← Done!
+ Weak Constraints ← Done!
+ Aggregate atoms ← Done!
+ Choice Rules ← Done!

How to solve problems with ASP?

Presentation roadmap

The language of ASP is:

Datalog ← Done!

- \+ Default negation ← Done!
- \+ Disjunction ← Done!
- \+ Integrity Constraints ← Done!
- \+ Weak Constraints ← Done!
- \+ Aggregate atoms ← Done!
- \+ Choice Rules ← Done!

How to solve problems with ASP?

Problem solving in ASP

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

Are you solving a hard combinatorial problem?

- **NO :** → Direct encoding with stratified program
- **YES:** → Guess & Check & Optimize methodology

Problem solving in ASP

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

Are you solving a hard combinatorial problem?

- **NO :** → Direct encoding with stratified program
- **YES:** → Guess & Check & Optimize methodology

Problem solving in ASP

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

Are you solving a hard combinatorial problem?

- **NO :** → Direct encoding with stratified program
- **YES:** → Guess & Check & Optimize methodology

Problem solving in ASP

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

Are you solving a hard combinatorial problem?

- **NO :** → Direct encoding with stratified program
- **YES:** → Guess & Check & Optimize methodology

Problem solving in ASP

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

Are you solving a hard combinatorial problem?

- **NO :** → Direct encoding with stratified program
- **YES:** → Guess & Check & Optimize methodology

Direct Encodings when...

Use a "Direct" Encoding with Datalog rules for

- Polynomial Problems, Deductive Database, 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
$reachable(X, Y) \text{ :- } reachable(X, Z), edge(Z, Y).$

Unfeasible for search problems in NP and beyond:

$\rightarrow$ *Need for a systematic 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

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

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

$$:-\; group(P1, G), group(P2, G), father(P1, P2).$$

...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?

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

$$group(P, 1) \mid group(P, 2) :\!- person(P).$$

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

$$:\!- group(P1, G), group(P2, G), father(P1, P2).$$

...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:

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

i.e., one a.s. for each assignment of 2 pers. to 2 groups!

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:

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

i.e., one a.s. for each assignment of 2 pers. to 2 groups!

Guessing part explained

Consider: $group(P, 1) \mid group(P, 2) :\!- \; person(P).$

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

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

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

i.e., one a.s. for each assignment of 2 pers. to 2 groups!

Guessing part explained

Consider: $group(P, 1) \mid group(P, 2) \coloneq person(P).$

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

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

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

i.e., one a.s. for each assignment of 2 pers. to 2 groups!

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

Now add: $:- group(P1, G), group(P2, G), father(P1, P2).$

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: $person(john).\quad person(joe).\quad father(john, joe).$

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 1)

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 1)

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 $\rightarrow$ only one color per node

Guess and Check (Example 1)

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 1)

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 (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
$inPath(X, Y) \mid outPath(X, Y) \mathrel{:-} edge(X, Y).$ | Guess

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

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

Guess and Check (Example 2)

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
$\text{:- } inPath(X, Y), inPath(X, Y1), Y \neq Y1.$ | Check
$\text{:- } inPath(X, Y), inPath(X1, Y), X \neq X1.$ |
% 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 2)

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 |
$\text{:- } inPath(X, Y), inPath(X, Y1), Y \neq Y1.$
$\text{:- } inPath(X, Y), inPath(X1, Y), X \neq X1.$ | 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 2)

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 |
$\text{:- } inPath(X, Y), inPath(X, Y1), Y \neq Y1.$
$\text{:- } inPath(X, Y), inPath(X1, Y), X \neq X1.$ | 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 2)

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 |
$\text{:- } inPath(X, Y), inPath(X, Y1), Y \neq Y1.$
$\text{:- } inPath(X, Y), inPath(X1, Y), X \neq X1.$ | 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, Check and Optimize (Example 3)

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

% Ensure that it is Hamiltonian
$\text{:- } inPath(X, Y), inPath(X, Y1), Y <> Y1.$ |
$\text{:- } inPath(X, Y), inPath(X1, Y), X <> X1.$ | Check
$\text{:- } node(X), \text{not } reached(X).$ $\text{:- } inPath(X, Y), start(Y).$ |
$reached(X) \text{ :- } reached(Y), inPath(Y, X).$ | Aux. Rules
$reached(X) \text{ :- } start(X).$ |

% Minimize the sum of distances
$:\sim inPath(X, Y), edge(X, Y, C). [C@1, X, Y]$ | Optimize

Guess, Check and Optimize (Example 3)

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 | Check

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

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

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

| Aux. Rules

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

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

% Minimize the sum of distances

$:\sim inPath(X, Y), edge(X, Y, C). [C@1, X, Y]$ | Optimize

Guess, Check and Optimize (Example 3)

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 | 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

$:\sim inPath(X,Y), edge(X,Y,C). [C@1, X, Y]$ | Optimize

Exercises

Rewrite the above encodings:

1. Using aggregates where counting is involved
2. Using choice rules instead of disjunctive rules
3. Extend 3-Col example to n-Col
4. Provide a non-ground encoding for 3SAT

Acknowledgments

Thanks for your attention!

Questions?

References

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. Commun. ACM, 54(12):92–103, 2011.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. ACM Comput. Surv., 33(3):374–425, 2001.

[EGL16] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. AI Magazine, 37(3):53–68, 2016.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. New Generation Comput., 9(3/4):365–386, 1991.

[GLM+18] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. Evaluation techniques and systems for answer set programming: a survey. In IJCAI, pages 5450–5456. ijcai.org, 2018.

References (cont.)

[GMR17] Martin Gebser, Marco Maratea, and Francesco Ricca. The sixth answer set programming competition. J. Artif. Intell. Res., 60:41–95, 2017.

[Lif99] Vladimir Lifschitz. Answer set planning. In ICLP, pages 23–37. MIT Press, 1999.