Probabilistic Answer Set Programming

A Research Draft

Francisco Coelho

NOVA LINCS &
High Performance Computing Chair &
Departamento de Informática, Universidade de Évora

May 26, 2022

In short

- Use **logic programs** to formalize knowledge.
  - logic program = formula = model.
  - **Observations** not always agree with such models — errors may result from sensors or from a wrong or incomplete model.
- We can associate **quantities** to formulas and sub-formulas.
  - And define how observations **update** those quantities.
- Adequate quantities and updates might be used to **interpret or evaluate the model** *e.g.* define a joint distribution or measure the accuracy of a clause.

Problem 1: Probabilities

The stable models of $c_1 \wedge c_2$ where

$$c_1 : b \vee \neg b$$
$$c_2 : h_1 \vee h_2 \quad \leftarrow b$$

are

$$\{\neg b\}, \{b, h_1\} \text{ and } \{b, h_2\}.$$

Associate quantities to clauses and update them with observations.

Then compute:

- The probability of a stable model.
- The probability of an atom.
- The joint distribution of all atoms.

Problem 1: Probabilities

The stable models of $c_1 \wedge c_2$ where

$$\begin{aligned} c_1 &: b \vee \neg b \\ c_2 &: h_1 \vee h_2 \quad \leftarrow b \end{aligned}$$

are

$$\{\neg b\}, \{b, h_1\} \text{ and } \{b, h_2\}.$$

Associate quantities to clauses and update them with observations.

- How to **match** an observation z with a clause case h_i, b?
- How do observations **update** the probabilities?
- Is this enough to compute the **joint distribution of the atoms**?

Matching observations and sub-formulas

- An observation is a subset of the literals[1] from a program.
- A consistent observation has no subset $\{p, \neg p\}$.
- A *consistent* observation z is relevant for the clause $h \leftarrow b$ if $b \subseteq z$.
- A disjunctive clause

$$h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$$

  has n cases: $\{h_i, b_1, \ldots, b_m\}, i = 1 : n$.
- The *consistent* observation z and the case $\{h, b_{1:n}\}$ match if $\{h, b_{1:n}\} \subseteq z$.

The above definitions apply to **facts**, $m = 0$, and **constraints**, $n = 0$.

[1] The set of atoms, a, of the program and their classic negations, $\neg a$.

Counters and updates

A consistent observation **relevant** for a clause $h_1 \vee \cdots \vee h_n \leftarrow b$ should **increase the probability of matched cases**.

Counters and updates

1. Associate **counters**, u, r, n, to clauses $h \leftarrow b$.
2. Associate a **counter**, m_i, to cases h_i, b.
3. **Initial** values result from *prior* knowledge.
4. Each *consistent* observation **increments**:
    - The u counters of relevant unmatched clauses (no matched cases).
    - The r counters of relevant clauses.
    - The n counters of not relevant clauses.
    - The m_i counters of matched cases h_i, b.
    - Clause counters must verify $r \leq u + \sum_i m_i$.

Counters and updates

A consistent observation **relevant** for a clause $h_1 \vee \cdots \vee h_n \leftarrow b$ should **increase the probability of matched cases**.

Counters and updates

- Literals must be explicitly observed: $\neg b \neq \sim b$.
- Counters relate a clause structure with observations.
- So far stable models had no role.

Counters and updates: An example

Given the following clauses with annotated counters,

$$b \lor \neg b \quad \text{counters: } 7, 2; 3, 12, 0$$

$$h_1 \lor h_2 \leftarrow b \quad \text{counters: } 4, 3; 2, 6, 5$$

Counters and updates: An example

Given the following clauses with annotated counters,

$$
\begin{array}{ll}
b \vee \neg b & \text{counters: } 7, 2; 3, 12, 0 \\
h_1 \vee h_2 \leftarrow b & \text{counters: } 4, 3; 2, 6, 5
\end{array}
$$

Counters of $b \vee \neg b$

0 observations where not relevant (because the body is $\top$);
There where 12 relevant observations;
Of those, b was matched by 7, $\neg b$ by 2 and 3 observations matched neither ($\models \sim b, \sim\neg b$).

Counters of $h_1 \vee h_2 \leftarrow b$

There where $11 = 6 + 5$ observations, 6 relevant to this clause;
From these, 4 matched h_1, 3 matched h_2 and 2 matched no case.

Counters and updates: An example

Given the following clauses with annotated counters,

$$b \vee \neg b \qquad \text{counters: } 7, 2; 3, 12, 0$$

$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 2, 6, 5$$

What can be computed?

- $P(\neg b) = \frac{2}{12}$ because $\neg b$ matched 2 of 12 relevant observations.
- $P(h_1|b) = \frac{4}{6}$ because h_1, b matched 4 of 6 relevant observations.
- $P(b)$ needs further information.
  - *E.g.* **assuming independent observations**,

$$P(b) = \frac{7+6}{12+0+6+5}.$$

Counters and updates: An example

Given the following clauses with annotated counters,

$$\begin{array}{ll} b \vee \neg b & \text{counters: } 7, 2; 3, 12, 0 \\ h_1 \vee h_2 \leftarrow b & \text{counters: } 4, 3; 2, 6, 5 \end{array}$$

What can be computed? — assuming independent observations

- $P(b) + P(\neg b) = \frac{13}{23} + \frac{2}{12} \approx 0.73 < 1$ because some observations have neither b nor $\neg b$.
- $P(h_1, b) = P(h_1|b)P(b) = \frac{4}{6}\frac{13}{23}$ from above.
- $P(h_2, b) = P(h_2|b)P(b)$ is analogous.
- But not *e.g.* $P(h_1|\neg b)$ because no clause relates h_1 and $\neg b$.

Counters and updates: An example

Given the following clauses with annotated counters,

$b \vee \neg b$	counters: $7, 2; 3, 12, 0$
$h_1 \vee h_2 \leftarrow b$	counters: $4, 3; 2, 6, 5$

Also. . .

Counters are local to clauses and, for distinct clauses, may result from distinct sources. *E.g. the relevant counter of* $h_1 \vee h_2 \leftarrow b$ *and the match counter of* b *in* $b \vee \neg b$.

Counters and updates: An example

Given the following clauses with annotated counters,

$$b \vee \neg b \qquad \text{counters: } 7, 2; 3, 12, 0$$

$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 2, 6, 5$$

Also...

Some observations may have neither b nor $\neg b$ so:

$$P(b) + P(\neg b) < 1.$$

Counters and updates: An example

Given the following clauses with annotated counters,

$$
\begin{array}{ll}
b \vee \neg b & \text{counters: } 7, 2; 3, 12, 0 \\
h_1 \vee h_2 \leftarrow b & \text{counters: } 4, 3; 2, 6, 5
\end{array}
$$

Also. . .

Assuming independent observations, since h_1 and h_2 are not independent,

$$\sum_m P(m) > 1.$$

Counters and updates: An example

Given the following clauses with annotated counters,

$$\begin{array}{ll} b \vee \neg b & \text{counters: } 7, 2; 3, 12, 0 \\ h_1 \vee h_2 \leftarrow b & \text{counters: } 4, 3; 2, 6, 5 \end{array}$$

Also...

What's missing to define the joint distribution

$$P(H_1, H_2, B)?$$

The joint distribution, according to the clauses

H_1	h_1	$\neg h_1$	$\sim h_1$
b	●	○	○
$\neg b$	○	○	○
$\sim b$	○	○	○

B		
b	$\neg b$	$\sim b$
●	●	·

H_2	h_2	$\neg h_2$	$\sim h_2$
b	●	○	○
$\neg b$	○	○	○
$\sim b$	○	○	○

Shortcomming 2: Default Negation

- How to deal with rules with $\sim a$ parts?
- Should missing elements on observations be replaced with $\sim a$ atoms?

Background Material

Machine Learning

Models are numeric functions: $y \approx f_\theta(x),\ \theta_i, x_j, y \in \mathbf{R}$.

- Amazing achievements.
- Noise tolerant.
- (as of today) Huge enterprise funding .

but

- (essentially) Academically solved.
- Models trained from "large" amounts of samples.
- Hard to add background knowledge.
- Models are hard to interpret.
- Single table, independent rows assumption.

Inductive Logic Programming

Models are logic program: $p_\theta(x, y),\ \theta_i, x_j, y \in \mathcal{A}$.

- Amazing achievements, at scale.
- Models trained from "small" amounts of samples.
- Compact, readable models.
- Background knowledge is easy to incorporate and edit.

but

- as of today, Little enterprise commitment.
- as of today, Mostly academic interest.
- Noise sensitive.

Distribution Semantics

Assigns probability to (marginally independent) facts and derives probability of ground propositions.
Let F be set of facts, $S \subseteq F$, R a set of definite clauses and p a proposition:

$$P_F(S) = \prod_{f \in S} P(f) \prod_{f \notin S} \big(1 - P(f)\big)$$

$$P(W) = \sum_{S \subseteq F:\ W = M(S \cup R)} P_F(S)$$

$$P(p) = \sum_{S:\ S \cup R\ \vdash\ p} P_F(S) = \sum_{W:\ p \in W} P(W)$$

- Amazing achievements, at scale.
- Lots of tools and research.
- The best of both “worlds”?

Answer Set Programming

A program defines stable models.

- Pure declarative language, unlike Prolog.
- Uses *generate & test* methods instead of proofs.
- Uses both default $\sim p$ and classical negation $\neg p$.
- Clauses can be disjunctive $a \vee b \leftarrow c \wedge d$.

ASP definitions

- An **atom** is $r(t_1, \dots t_n)$ where
  - r is a n-ary predicate symbol.
  - each t_i is a constant or a variable.
- A **ground atom** has no variables.
- A **literal** is either an atom a or a negated atom $\neg a$.
- An **ASP Program** is a set of **rules** such as $h_1 \vee \dots \vee h_m \leftarrow b_1 \wedge \dots \wedge b_n$ where
  - Each h_i is a literal, a or $\neg a$.
  - Each b_j is a literal like above or preceded by $\sim$.
  - $m + n > 0$.
- The **head** of such rule is $h_1 \vee \dots \vee h_m$.
- The **body** of such rule is $b_1 \wedge \dots \wedge b_n$.
- Each b_i is a **subgoal**.

ASP definitions (cont.)

- A **non-disjunctive rule** has $m \leq 1$.
- A **normal rule** has $m = 1$.
- A **constraint** has $m = 0$.
- A **fact** is a normal rule with $n = 0$.
- The **dependency graph** of a program is a digraph where:
  - Each grounded atom is a node.
  - For each grounded rule there are edges from the atoms in the body to the atoms in the head.
- A **negative edge** results from an atom with $\sim$; Otherwise it is a **positive edge**.
- An **acyclic program** has an acyclic dependency graph.

ASP definitions (cont.)

- A **normal program** has only normal rules.
- A **definite program** is a normal program that doesn't contains $\neg$ neither $\sim$.
- In the dependency graph of a **stratified program** no cycle contains a negative edge.
  - A stratified program has a single minimal model that assigns either true or false to each atom.
- A **propositional program** has no variables.

ASP definitions (cont.)

- The **Herbrand base** of a program is the set of ground literals that result from combining all the predicates and constants of the program.
- An **interpretation** is a consistent subset (*i.e.* doesn't contain $\{a, \neg a\}$) of the Herbrand base.
- A ground literal is **true**, $I \models a$, if $a \in I$; otherwise the literal is **false**.
- A ground subgoal, $\sim b$, where b is a ground literal, is **true**, $I \models \sim b$, if $b \notin I$; otherwise, if $b \in I$, it is **false**.
- A ground rule $r = h_1 \vee \cdots \vee h_m \leftarrow b_1 \wedge \cdots \wedge b_n$ is **satisfied** by the interpretation I, *i.e.* $I \models r$, iff
  - $I \not\models b_j$ for some j or $I \models h_i$ for some i,
- A **model** of a program is an interpretation that satisfies all the rules.

Stable Semantics

- Every definite program has a unique minimal model; its *semantics*.
- Programs with negation may have no unique minimal model.
- Given a program P and an interpretation I, their **reduct**, P^I is the propositional program that results from
  1. Removing all the rules with $\sim b$ in the body where $b \in I$.
  2. Removing all the $\sim b$ subgoals from the remaining rules.
- A **stable model** of the program P is an interpretation I that is the minimal model of the reduct P^I.
- The **semantics** (the **answer sets**) of a program is the set of stable models of that program.

Stable Semantics

- A program such as $a \leftarrow \sim a$ may have no stable models.
- A stable model is a closed interpretation (under the rules of program).

1. **Development**

2. **Conclusions**