Chapter 3: Answer-Set Programming

Non-Monotonic Logic Programming & Combinatorial Problem Solving

Leopoldo Bertossi

Where We Start From

- Datalog programs with (weak, non-monotonic) negation (*not*) in rule bodies are called "Normal Programs" in the context of Logic Programming
- Datalog programs with stratified negation form a particular class of Normal Programs (NPs)
- In the following, programs may have no negation, stratified negation, or unstratified negation
- We will give a uniform semantics to NPs
- This general semantics will coincide with those we already have for negation-free programs and programs with stratified negation
- As usual, we give a model-based semantics to NPs

  By characterizing the intended models, i.e. a "possible-worlds semantics"

First, a reminder of why we call negation "weak or non-monotonic"

Example:

$$
\begin{aligned}
\mathit{Flies}(x) &\leftarrow \mathit{Bird}(x), \mathit{not}\ \mathit{Abnormal}(x)\\
\mathit{Abnormal}(x) &\leftarrow \mathit{Penguin}(x)\\
\mathit{Abnormal}(x) &\leftarrow \mathit{Ostrich}(x)\\
\mathit{Abnormal}(x) &\leftarrow \mathit{Canary}(x), \mathit{BrokenWing}(x)\\
\mathit{Bird}(x) &\leftarrow \mathit{Canary}(x)\\
\mathit{Canary}(\text{tweety}).&
\end{aligned}
$$

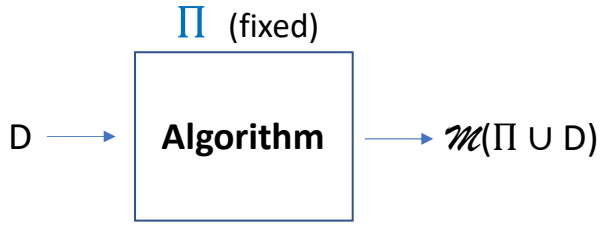
1. A program with stratified negation
2. Its standard model (containing the only true atoms) is $\mathcal{M} = \{\mathit{Canary}(\text{tweety}), \mathit{Bird}(\text{tweety}), \mathit{Flies}(\text{tweety})\}$ ($\Rightarrow$ Tweety flies)
3. No evidence that it is abnormal, so it is assumed it is not (commonsense reasoning!)
4. Negation is weak in that it becomes true when there is not evidence for the affected positive atom to be true
5. Also non-monotonic: adding knowledge to the program may lead to a rejection of a previous conclusion

   With additional atom $\mathit{BrokenWing}(\text{tweety})$, Tweety no longer flies

Data Complexity of Datalog^{s,not}

- The evaluation of a Datalog^{s,not} program can be done in polynomial time in the size of the extensional database
- More precisely, consider the following computational problem: Fix a program Π without an EDB:

  Given an EDB D as input, compute the standard model $\mathcal{M}(\Pi \cup D)$



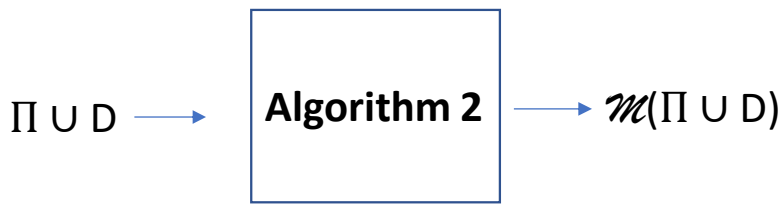
P(x) fixed polynomial depending on Π

Computing $\mathcal{M}(\Pi \cup D)$ takes $P(|D|)$ steps

- This is "data complexity"
- Same for query answering (with fixed program and query)
- Since plain Datalog is contained in Datalog^{s,not}, the same applies to Datalog (and the minimal model)

- The proof is based on the analysis of the "iterative bottom-up algorithm" for computing the intended model
- Polynomial time is considered "short" time, in contrast to exponential time
- When confronted with computational problems related to data, "data complexity" is the relevant measure

  Usually the DB is large, and the other inputs, e.g. a query, a program, etc., tend to be small in comparison with the DB
- A different computational problem:



  Its complexity is "combined complexity", time measured in terms of the sizes of both inputs, Π and D

The Stable Model Semantics

- Example: (revisited) What is the semantics of this program?

  $$P(a) \leftarrow \ not\ P(b)$$

  An unstratified ground program

  What are its intended models?

- We need a semantics for this kind of programs

  Hopefully we will reobtain the good old semantics for negation-free and stratified programs

- It is the stable model semantics (SMS)

  Or more generally, the answer set semantics (Gelfond & Lifschitz, 1988)

  It can be applied to a normal program Π; stratified or unstratified

  Π can be a ground program or have variables

- Consider a normal logic program Π
- Let $S \subseteq HB(\Pi)$, a subset of its Herbrand Base

  S is a set of assumptions, and a candidate to be a (stable) model of Π

  A "guess" that will be accepted if properly supported by Π
- For S to be an intended model, its atoms have to be properly justified by Π

  S is stable model of Π if it is a model and is properly justified by Π

  In other terms, if assuming S, we can recover S via Π
- We make the model candidate S pass through a test

1. Pass from Π to Π_H, the ground instantiation of Π
2. Construct a new ground program Π_H^S, depending on S as follows:

   (a) Delete from Π_H every rule that has a subgoal *not* A in the body, with $A \in S$

   Intuitively: We are assuming A to be true, then *not* A is false, then the whole body is false, and nothing can be concluded with that rule, it is useless

   (b) From the remaining rules, delete the negative subgoals

   Intuitively: Those rules are left because the negative subgoals are true, and since they are true, we can eliminate them as conditions in bodies (because they hold)
3. We are left with a ground, negation-free program Π_H^S, a residual program determined by S
4. Compute $\underline{M}(\Pi_H^S)$, the minimal model of this positive program
5. If $\underline{M}(\Pi_H^S) = S$, S is a stable model of Π

- Intuitively, we started with S (as an assumption) and we recovered it

  It was stable wrt. to the Π-guided process described above; it is self-justified

- Example: Program Π $\quad P(a) \leftarrow \mathit{not}\ P(b)$ (already ground)

  Consider $S = \{P(a)\}$

  $P(b) \notin S$, then $\mathit{not}\ P(b)$ is satisfied in S and can be eliminated from the body

  We obtain Π^S: $P(a) \leftarrow$ a ground, negation-free program

  Its minimal model is $\{P(a)\}$, that is equal to S

  S is a stable model of the original program

  This is the only stable model (check other subsets of the HB!)

- Notice that Π is unstratified (there is recursion via negation), but has a stable model

Exercise: For the program above, verify that:

(a) The empty set $\{\}$ (as a subset of the HB) is not a model

(b) $\{P(a)\}$ is a model

That is, it makes all the implications of the program true

For $S \subseteq HB(\Pi)$, by definition: *not* $P(a)$ is true in S iff $P(a) \notin S$

(c) $\{P(a)\}$ is a minimal model, that is, no proper subset is a model

(d) $\{P(b)\}$ is a model

(e) $\{P(b)\}$ is a minimal model

(f) $\{P(b)\}$ is not a stable model

So, there are minimal models that are not stable

(g) What about $\{P(a), P(b)\}$?

Example: Program Π (unstratified)

$P(x) \leftarrow Q(x, y),\ not\ P(y).$ $\qquad$ $Q(a, b).$

Π_H: (ground instantiation) $\qquad$ Candidate $S = \{P(b)\}$

$P(a) \leftarrow Q(a, a),\ not\ P(a)$
$P(a) \leftarrow Q(a, b),\ not\ P(b)$ $\quad \times$
$P(b) \leftarrow Q(b, a),\ not\ P(a)$
$P(b) \leftarrow Q(b, b),\ not\ P(b)$ $\quad \times$ $\qquad$ $Q(a, b).$

Π_H^S : $\qquad$ $P(a) \leftarrow Q(a, a)$ (residual program)
$P(b) \leftarrow Q(b, a)$ $\qquad$ $Q(a, b).$

Minimal model of Π_H^S is $\{Q(a, b)\} \neq S$

Then, S is not a stable model

Now consider: $S = \{Q(a,b), P(a)\}$

$P(a) \leftarrow Q(a,a),\ \underline{not\ P(a)}$ $\times$
$P(a) \leftarrow Q(a,b),\ \underline{not\ P(b)}$
$P(b) \leftarrow Q(b,a),\ \underline{not\ P(a)}$ $\times$
$P(b) \leftarrow Q(b,b),\ \underline{not\ P(b)}$ $Q(a,b).$

Π_H^S : $P(a) \leftarrow Q(a,b)$
$P(b) \leftarrow Q(b,b)$ $Q(a,b).$

Minimal model of Π_H^S is $\{Q(a,b), P(a)\} = S$

S is a stable model of Π

Example: Program Π:

$$\begin{aligned} Male(x) &\leftarrow Person(x),\ not\ Female(x) \\ Female(x) &\leftarrow Person(x),\ not\ Male(x) \\ Person(a). & \end{aligned}$$

If $S_1 = \{Person(a), Male(a)\}$, then Π^{S_1} is

$$\begin{aligned} Male(a) &\leftarrow Person(a) \\ Person(a). & \end{aligned}$$

S_1 is a stable model

$S_2 = \{Person(a),\ Female(a)\}$ is also a stable model of Π

There may be more than one stable model for a program!

(Check them!)

Again, Π is unstratified

The semantics for Datalog or Datalog^{s,not} cannot be applied

Exercise: For the program $P(a) \leftarrow \ not\ P(a)$, verify that:

(a) $\{\}$ is not a model

(b) The program has no stable models

Hint: Consider two cases for an $S \subseteq HB(\Pi)$: (a) $P(a) \notin S$; (b) $P(a) \in S$

We say that the program is inconsistent (under the stable model semantics)

So, normal programs may have no stable models

(c) Actually it has no (Herbrand) models

Exercise: Find and verify the stable models of the program

$$P(a) \leftarrow \ not\ P(b)$$
$$P(b) \leftarrow \ not\ P(a)$$

Example: Program Π with a function symbol

$$even(0).$$
$$even(x) \leftarrow not\ even(s(x))$$

$H = \{0, s(0), s(s(0)), s(s(s(0))), \ldots\}$

Π_H:

$$even(0).$$
$$even(0) \leftarrow not\ even(s(0))$$
$$even(s(0)) \leftarrow not\ even(s(s(0)))$$
$$\ldots \qquad \ldots$$

$S = \{even(0), even(s(s(0))), even(s(s(s(s(0))))), \ldots\}$ is the only stable model

Π_H^S:

$$even(0).$$
$$even(0).$$
$$even(s(s(0))). \qquad \text{Etc.}$$

$\underline{M}(\Pi_H^S) = \{even(0), even(s(s(0))), even(s(s(s(s(0))))), \ldots\}$

Exercise: Find the stable models for the unstratified program:

$P(x) \leftarrow R(x),\ not\ Q(x)$
$Q(x) \leftarrow P(x)$
$Q(a),\ R(b)$

Exercise: Show using any of the example programs above that the use of negation in the stable model semantics is indeed non-monotonic

That is, show that by adding a new fact to the EDB, you may lose a previous certain consequence, i.e. something true in all stable models

Some Results and Notions

- Every stable model of Π is a Herbrand model in the usual sense

  In them, *not* is interpreted as "not belonging to the model"

- A stable model is always a minimal model (i.e. no proper subset of it is a model of the program)

- A normal program may have several stable models

- Several stable models may determine the semantics of a program:

  What is true of (wrt.) the program is what is true *in all* its stable models

- If there are several stable models for a program, some atoms are left *undetermined*

  Those that are true in some models, but false in others

- Example: The program on page 13 leaves every *Male*-atom and *Female*-atom undetermined, uncertain

  However, it is certain that $Person(a)$

  However, it is still certain that $Male(a) \vee Female(a)$

- If we apply the SMS to a Datalog program, its only stable model is its minimal model
- If we apply the SMS to a Datalog program with stratified negation, its only stable model is its standard model
- The SMS is an extension of the semantics for the previous, simpler classes of programs!

  Then, the stable model semantics extends the ones we had for the "good" classes before!

- For those classes, the unique stable model can be computed by means of an bottom-up iterative process

  In polynomial time in the size of the EDB

- Exercise: Check that the standard model for the stratified program on page 3 is a stable model

- We also obtain right away that the stable model semantics is non-monotonic (already shown for stratified programs)

  So, it is useful for representation of commonsense knowledge, and commonsense reasoning

- We have given a declarative, model-based semantics to a wider class of programs (with or without negation), even non-stratified

- This is a "possible-world semantics" (this kind of semantics is common in data management and AI)

- Notions of entailment from a program under the stable model semantics of a normal program?
  - Skeptical, certain or cautious semantics: What is true of a program is what is true in all stable models of the program

    In the previous example, $Person(a)$ is skeptically true, but not $Male(a)$
  - Brave or possible semantics: What is true of a program is what is true in some stable model of the program

    In the previous example, $Male(a)$ is possibly (or bravely) true
- Both are useful depending on the use of the program

Example: Extension of previous program Π':

$$
\begin{aligned}
\mathit{HumanBeing}(x) &\leftarrow \mathit{Male}(x) \\
\mathit{HumanBeing}(x) &\leftarrow \mathit{Female}(x) \\
\mathit{Male}(x) &\leftarrow \mathit{Person}(x),\ \mathit{not}\ \mathit{Female}(x) \\
\mathit{Female}(x) &\leftarrow \mathit{Person}(x),\ \mathit{not}\ \mathit{Male}(x) \\
\mathit{Person}(a).&
\end{aligned}
$$

Queries $\mathcal{Q}(x)$:

- $\Pi \models_{\mathsf{skep}} \mathit{HumanBeing}(x)?$ $\quad x = a$
- $\Pi \models_{\mathsf{brave}} \mathit{HumanBeing}(x)?$ $\quad x = a$
- $\Pi \models_{\mathsf{skep}} \mathit{Person}(x)?$ $\quad x = a$
- $\Pi \models_{\mathsf{brave}} \mathit{Person}(x)?$ $\quad x = a$
- $\Pi \models_{\mathsf{skep}} \mathit{Male}(x)?$ $\quad \emptyset$
- $\Pi \models_{\mathsf{brave}} \mathit{Male}(x)?$ $\quad x = a$
- $\Pi \models_{\mathsf{skep}} \mathit{Female}(x)?$ $\quad \emptyset$
- $\Pi \models_{\mathsf{brave}} \mathit{Female}(x)?$ $\quad x = a$

Always: $\mathit{Certain}(\Pi, \mathcal{Q}) \subseteq \mathit{Possible}(\Pi, \mathcal{Q})$

Program Constraints

- Program constraints (PCs) can be added to a NP Π:

  $$\leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_k,$$

  with A_i atoms (a rule with empty head)

- Intuitively, it says: It is not possible for the body to be true (for any values for the variables) ... A prohibition!

- It has the effect of filtering or eliminating the stable models of Π that make the body true

- Example:

  $$\begin{aligned} ug(x) &\leftarrow stud(x),\ \mathit{not}\ grad(x) \\ grad(x) &\leftarrow stud(x),\ \mathit{not}\ ug(x) \\ stud(mary) &\leftarrow \\ &\leftarrow ug(x) \end{aligned}$$

  "no UG students!"

- Without the PC, two stable models: $\{stud(mary), ug(mary)\}$ and $\{stud(mary), grad(mary)\}$

- With the PC, only the second one

Example: (3-GC) Want a normal program that gives us coloring of a graph with 3 colors, without adjacent nodes sharing the same color

$color(x, green) \leftarrow country(x),\ not\ color(x, blue),\ not\ color(x, red)$
$color(x, blue) \leftarrow country(x),\ not\ color(x, green),\ not\ color(x, red)$
$color(x, red) \leftarrow country(x),\ not\ color(x, blue),\ not\ color(x, green)$

plus atoms of the form $edge(a, c)$ representing the graph at hand

We need a PC:

$$\leftarrow edge(x, y), color(x, z), color(y, z)$$

"No neighboring countries can share the same color"

- Each SM will represent a possible coloring of the whole graph
- No models if the problem is not solvable

  In which case the program is inconsistent

- The same program can be used with any instance of the problem, i.e. concrete graph, represented trough the facts
- Can a country be painted with more that one color?
- The minimality of stable models will ensure that no country is painted with more than one color
- Keep this minimality in mind when you create your program

  It is a useful implicit tool ...
- This is a normal program that can be used to solve a hard (NP-complete) combinatorial problem

- PCs look like a new element in NPs
- Are they really needed?
- Actually, not, but it is useful to have them as above
- They could be eliminated if wanted, as follows
- The PC $\leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_k$

  can be replaced by the usual rule:

  $$q \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_k, \mathit{not}\ q$$

  q is a fresh propositional (or any) atom
- If in a stable model $A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_k$ becomes true, then also $q \leftarrow \mathit{not}\ q$ becomes true, which is not possible (cf. page 14)

  So, in any SM of the original program,
  $A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_k$ cannot be true
- Notice that with PCs, the original NP becomes unstratified (if it was not already)

- Example: (cont.) $q \leftarrow ug(x),\ not\ q$

  If in a candidate S to be a SM $ug(x)$ becomes true (for any value of x), also $q \leftarrow not\ q$ has to be true

- A PC has the effect of filtering the SMs of Π (without the PC) where the body of the PC becomes true
- PCs are an extra source of unstratified negation, and then, of extra computational cost (as we will see)
- In data management applications, PCs are useful to capture common ICs
- E.g. denial constraints:

  $$\leftarrow Employee(x, \text{janitor}), InBoard(x, \text{yes})$$

  "No janitors in the company board"

- A functional dependency $R: A \rightarrow B$:

  $$\leftarrow R(x, y), R(x, z), y \neq z$$

- The rules of the program may specify (and perform) different data management tasks
- We want to make sure the possible alternative states of the DB, represented by SMs, are consistent wrt. ICs
  ~~Se~~
- PCs can be used for that
- Example: Consistency of DB wrt. a referential constraint:

  "the values in the second argument of table $R(A, B)$ must appear in the first column of table $S(B, C)$"

- An attempt: $\leftarrow R(x, y), not\ S(y, z)$ (not safe)
- Better: (c.f. Chap. 2)

$$\leftarrow \quad R(x, y), not\ S'(y)$$
$$S'(y) \quad \leftarrow \quad S(y, z)$$