Logic-based Learning in ASP

Mark Law
mark.law09@imperial.ac.uk

Structure

- Stable Model Semantics
  - History and motivations
  - Fundamental definitions for the Stable Model Semantics
  - Brave and Cautious entailment
- Answer Set Programming (ASP)
  - Extended syntax of ASP including constraints, choice rules and optimisation
  - Modelling in ASP
- Non-monotonic learning in ASP
  - Brave and Cautious Induction in ASP
  - The algorithm of ASPAL
- Learning from Answer Sets (LAS)
  - Motivation: the need for a brave and cautious learning approach in ASP
  - Relationship to other learning approaches
  - Algorithm for computing the optimal solutions of any LAS task

In this part of the course, we will consider non-monotonic learning under the Answer Set/Stable Model Semantics. In todays lecture, we will introduce the concepts needed for the next 4 lectures.
In Lecture 2 we look at early approaches to learning under the Answer Set Semantics and study the algorithm ASPAL which maps its ILP task to an ASP program which can then be solved for optimal hypotheses.
In the final two lectures we will study a recent more general approach to learning ASP programs developed within the department. This consists of a new learning task - Learning from Answer Sets - and its corresponding algorithm - ILASP.

Stable Model Semantics

Mark Law
mark.law09@imperial.ac.uk

4	3	1	2
2	1	3	4
3	2	4	1
1	4	2	3

Prolog and negation

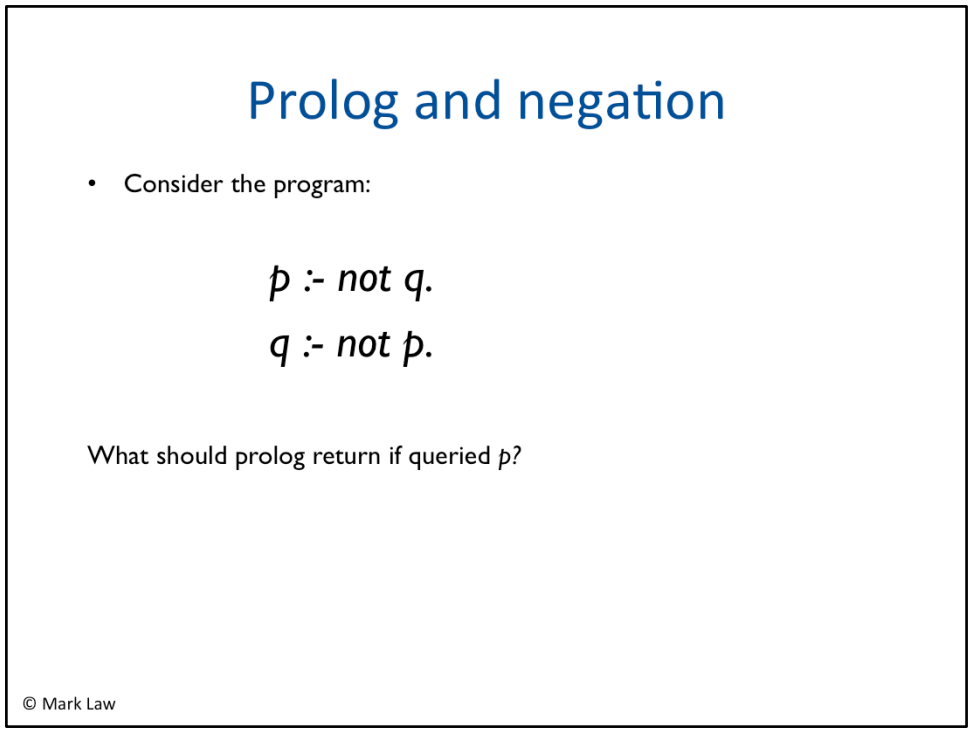
- Consider the program:

```
p :- not q.
q :- a.
q :- b.
```

What should prolog return if queried *p?*

Prolog programs can be queried as to whether a particular formula is true; for example, in the program *{p :- not q. q :- a. q :- b.}* the query *p?* causes the search shown on the slide.

Searching for *(not q)* leads to a search for *q*. If each branch of the search tree return false (in finite time) then *(not q)?* returns true. As the search for *q* returns false, *(not q)?* returns true. Therefore the query *p?* returns true.



Given the program { p :- not q. q :- not p. }, the Prolog query p? leads to the infinite search:

p?
(not q)?
q?
(not p)?
…

There are, in fact, multiple answers to the query. The value of p depends on the value of q: if q is true then p is false, but if q is false then p is true. q depends similarly on p and thus cannot be determined either.

The stable model semantics (Gelfond and Lifschitz 1988) is a different approach to solving normal logic programs. Rather than providing solutions to specific queries, they define the set of "stable" models for the program. This particular program (as we will see) has the stable models {p} and {q}.

Stable Model Semantics

Syntax

- The syntax of normal logic programs is a subset of the syntax of Prolog.
- Atoms are of the form: $p(f(X),c)$
- p is a predicate, X is a *variable*, f is a function symbol, c is a constant.
- Ground atoms are those which contain no variables.

- Normal logic programs are sets of rules of the form R:

$$h \text{ :- } b_1, \ldots, b_m, \text{ not } c_1, \ldots, \text{ not } c_n.$$

- h, b's, c's are all atoms.
- $head(R) = h$; $body^+(R) = \{ b_1, \ldots, b_m \}$; $body^-(R) = \{ c_1, \ldots, c_n \}$
- Note: for facts (rules with no body) we will omit the :-

When the stable model semantics were first defined, these definitions were applied to normal logic programs. These are collections of rules of the form of the rule R below:

h :- $b_1, \ldots, b_m$, not $c_1, \ldots,$ not c_n.

where the h, b's and c's are all atoms.

We refer to h as the head of the rule, and the b's and c's (collectively) as the body of the rule. We write $head(R)$ to denote h, $body^+(R)$ to denote $\{b_1, \ldots, b_m\}$ and $body^-(R)$ to denote $\{c_1, \ldots, c_n\}$.

Note: as is conventional, for rules with empty bodies (facts) we omit the :-

Grounding

- The first stage of solving any normal logic program, is to ***ground*** it.
- For any program *P* with function symbols, there are infinitely many ground instances of each rule in *P*.

P:

p(f(X)) :- q(X).
q(a).

ground(P):

- We can generate the set of relevant ground instances of rules in *P* (written *ground(P)*) incrementally.
- At each step we generate all rules which are ground instances R_g of rules *R* in *P*, such that for each atom *A* in $body^+(R_g)$ there is at least one rule already in *ground(P)* with *A* as the head.

The grounding of a program *P* is the program constructed by replacing each rule *R* in *P* by every ground instance of *R*. Most of the time, this grounding will be infinite as there are infinitely many redundant rules whose bodies could never be satisfied. ASP solvers do not generate redundant rules. They generate the grounding incrementally, only generating rules R such that each atom a in $body^+(R)$ is already the head of another ground rule. We write *ground(P)* to refer to this grounding.

Safety

- In Prolog, rules will flounder:

p(X) :- not q(X, Y), r(X, Y).
r(a, b).

- In ASP, this rule is fine, but there is another condition: safety
- A rule is unsafe if it contains a variable which doesn't occur in $body^+(R)$.

p(Z) :- not q(X,Y), r(X,Y).
p(X) :- not q(X,Y), r(X,Y).
p(Y) :- not q(X,Y), r(Y,Y).

In Prolog, floundering is a problem because negative literals may contain a variable which is only ground by a positive literal occurring later in the body of the rule. As ASP does not consider the order of the literals in a rule, this is less of a problem in ASP; however, due to ASP's need to ground the entire program, another condition (safety) is needed instead. ASP solvers are restricted to only "safe" rules. A rule *R* is safe, if every variable in *R* occurs in at least one atom in $body^+(R)$. For example *p(X) :- q(Y)* is not safe, but *p(X) :- p(X)* is. *p(X) :- q(X), not r(Y)* is not safe either. Note that this means that every variable in the head of the rule must occur positively in the body; the fact *p(X)* is unsafe. This is very different to Prolog.

Even restricting ourselves to safe rules, *ground(P)* can still be infinite; for example:
p(f(X)) :- p(X).
p(1).
has an infinite grounding.

Herbrand Model

- For any normal logic program *P*, the Herbrand Base of *P* (written *HB(P)*) is the set of all ground atoms made from the constants, function symbols and predicate symbols in *P*.

p(f(X)) :- q(X). q(a).

{ p(a), p(f(a)), ..., q(a), q(f(a)), ... }

- An Herbrand interpretation of *P* assigns each atom in *HB(P)* to either *true* or *false*. We write it as the set of all atoms it assigns to *true*.
- An Herbrand model *M* of P, is an Herbrand interpretation such that for every ground instance of a rule in *P* whose body is satisfied by *M*, the head is also satisfied by *M*.

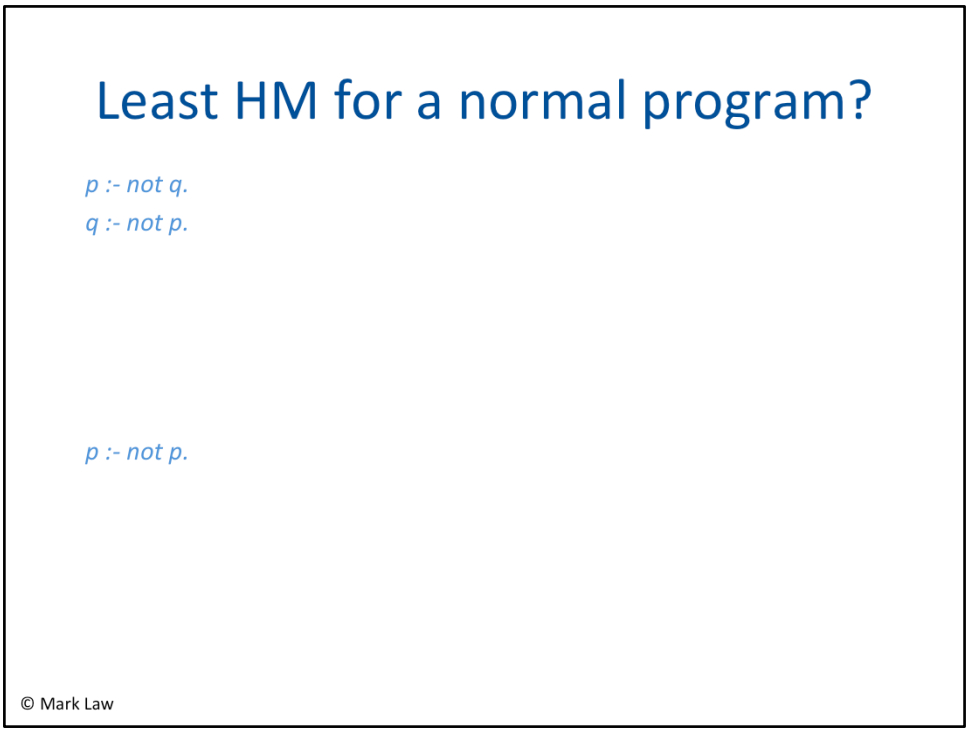
For any logic program *P* the Herbrand Base *HB(P)* is the (possibly infinite) set of all ground atoms constructed from predicates, constants and function symbols in *P*.
An Herbrand interpretation of P assigns a truth value (*true* or *false*) to each atom in *HB(P)*. We will write an Herbrand interpretation *I* as the set of all atoms in *HB(P)* which *I* assigns to true.
For definite logic programs an Herbrand interpretation *I* of *P* is an Herbrand model if for every rule *R* such that each atom in $body^+(R)$ is true in *I* and each atom in $body^-(R)$ is false in *I*, the head of *R* is true in *I*.

Least Herbrand Model

- An Herbrand model M is a minimal Herbrand model of a program *P* if no subset of M is also a model.
- For definite logic programs, there is a unique minimal Herbrand model called the least Hebrand model (denoted *M(P)*)
- We can construct it, starting with *M* = {}, by iteratively adding the heads of (the ground instances of) those rules whose bodies are already satisfied by *M*.

p(f(X)) :- q(X).
p(X) :- r(X), q(Y). *M(P) = ?*
q(b). *r(a).*

An Herbrand model *M* of *P* is a minimal Herbrand model of *P* if there is no smaller Herbrand model of *P*. For definite logic programs, there is a unique minimal Herbrand model, called the least Herbrand model, which we will denote with *M(P)*.
For a definite logic program *P*, *M(P)* can be constructed by starting with the empty set *M = {}* and repeatedly adding any atom *h* to *M* such that *h* is the head of a rule whose body is a subset of *M*. For definite programs with no loops this is the same as what is provable using Prolog.



When it comes to normal logic programs, in general, there is not a least Herbrand model. The first program depicted on the slide has two minimal models; in fact, we shall see that these coincide with the stable models of the same program.

The second program does have a least Herbrand model. But *p* is "unsupported". That is: there is no rule in the program with *p* as its head whose body is true given the model *{p}*. In fact, there are no stable models of this program.

Reduct

- The reduct of any ground normal logic program *P* with respect to any set of atoms *X* is constructed in two steps:
- 1) Remove any rule from *P* whose body contains the negation as failure of an atom in *X*.
- 2) Remove any negation as failure atoms from the remaining rules in P
- The remaining logic program is the ***reduct*** of *P* with respect to *X*, written P^X

X = {p} *P:* *p :- not q.*
q :- not p.

For a normal logic program, in general there may not be a unique least Herbrand model. In 1988 Gelfond and Lifschitz defined the stable model semantics for normal logic programs. To determine whether an interpretation *X* is a stable model of *P*, we must first construct the reduct of *P* with respect to *X* (written P^X).
The first step in constructing P^X is to remove any rule *R* from *P* such that $body^-(R)$ contains an atom which is not in *X*.
The second (and final) step is, for all remaining rules *R*, to remove $body^-(R)$.

This process can be thought of as making a guess at an interpretation *X* which might be an Answer Set and then assuming the Answer Set to be true when interpreting the negation as failure in the program. For example, on the slide when we assume *{p}* to be the Answer Set, we know that *not q* is true as *q* isn't in *{p}* so we can remove the literal from the first rule; we also know that *not p* is false as *p* is in *{p}* and can therefore remove the rule with this literal in the body.

Stable Model

- An interpretation *X* is a stable model of a normal logic program *P* iff *X* is the least Herbrand model of ground(*P*)^X. (X = M(ground(P)^X))

P *p :- not q.* *X = {p}*
q :- not p.

P^X *p.*

$M(P^X) = \{p\} = X$

X is a stable model of P. *Similarly {q} is also a stable model of P*

An interpretation *X* is a stable model of *P* if and only if *X* is the unique least Herbrand Model of *ground(P)*^{X}. For propositional programs, we will often omit the *ground* as *ground(P) = P* and write instead P^X.
For the logic programs we have considered so far, stable models are equivalent to the more general concept of Answer Sets. Answer Set Programming (ASP) allows for many more kinds of rule such as classical (strong) negation, constraints, aggregates, optimisation statements and weak constraints. We will explore some, but not all, of these in this course.
From this point onwards, even when referring to stable models of normal logic programs, we will use the term Answer Set.

Stable Model

- An interpretation *X* is a stable model of a normal logic program *P* iff *X* is the least Herbrand model of ground(P)^X. ($X = M(\text{ground}(P)^X)$)

P *p :- not q.* *X = {p, q}*
q :- not p.

P^X *empty!*

$M(P^X) = \{\} \neq X$

X is **not** *a stable model of P.*

Stable Model

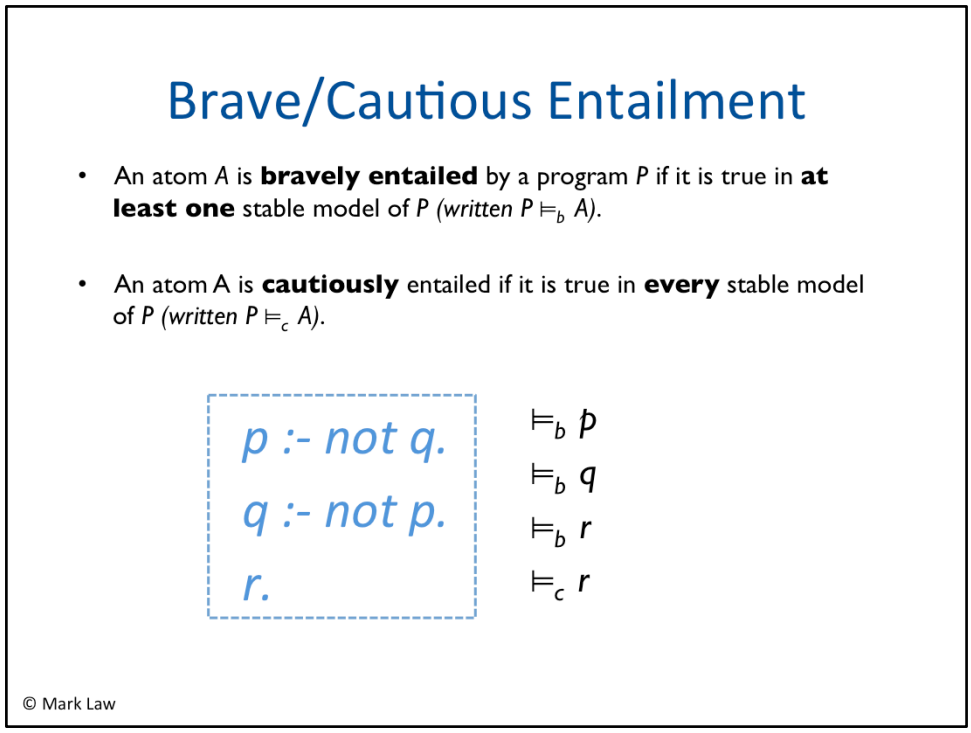
- An interpretation *X* is a stable model of a normal logic program *P* iff *X* is the least Herbrand model of ground(*P*)^X. (X = M(ground(P)^X)

P *p :- not q.* *X = { }*
q :- not p.

P^X *p.* *q.*

$M(P^X) = \{p, q\} \neq X$

X is **not** *a stable model of P.*



For definite logic programs *P*, the notion of entailment, is just whether a formula is true in the least Herbrand model of a program *P*. For normal logic programs, there can be one, many or even no stable models. This leads to two different notions of entailment: **brave** and **cautious**.
We say that a formula is bravely entailed by a program *P* if it is true in at least one stable model of *P*. Conversely, it is cautiously entailed if it is true in all stable models of *P*.
For definite logic programs, the least Herbrand model coincides with the programs unique stable model and therefore the notions of brave and cautious both correspond with the least Herbrand entailment.

Summary

- Saw some examples of normal logic programs in which some queries cannot be evaluated in Prolog.
- Defined
  - Safety and grounding
  - Least Herbrand Model (Recap)
  - The reduct of a program
  - Stable models of a normal logic program
  - Brave and cautious entailment
- We saw a general way to test whether an interpretation is a stable model of a particular program