491 Knowledge Representation

Stable models ('Answer sets')

Marek Sergot
Department of Computing
Imperial College, London

February 2006 v1.2, November 2016 v1.2c

A *normal logic program* (sometimes a 'general logic program') is a set of clauses of the form:

$$A \;\leftarrow\; L_1, \ldots, L_n \qquad (n \geq 0)$$

where A is an *atom* and each L_i is either an atom or a *nbf-literal* of the form not A where A is an atom. not denotes negation by failure.

A *model* of a normal logic program P is a set of atoms X such that $\mathrm{T}_P(X) \subseteq X$, or equivalently, such that X is an (ordinary, classical) model of the clauses $P^\neg$ obtained by replacing every occurrence of not in P by ordinary, truth-functional negation $\neg$.

Stable models (Gelfond & Lifschitz) – also called 'answer sets' – provide a simple semantics for normal logic programs that works even for programs like this:

$$\begin{array}{l} p(1,2) \leftarrow \\ q(X) \leftarrow\; p(X,Y),\ \text{not}\ q(Y) \end{array}$$

This program is not stratified, and not even locally stratified. Yet it seems to have a clear intuitive meaning (such programs arise very naturally in many forms of default reasoning). Even in Prolog the query ?-$q(X)$ does not loop and returns the answer you would intuitively expect.

The ideas of stable models can be generalised very naturally to extended logic programs, which combine negation-by-failure not with 'classical' ('strong', 'explicit') negation $\neg$. For a normal logic program, a stable model is a set of *atoms*. For an extended logic program, a stable model (*alias* an 'answer set') is a set of *literals*. We will look at extended logic programs later.

There is a strong connection between stable models and *autoepistemic logic*, which is an attempt to formalise how a rational agent can reason about its own beliefs. We won't pursue the connection to autoepistemic logic.

Stable models (answer sets) can also be seen as a special case of 'extensions' in Reiter *Default Logic*. What normal logic programs do to atoms, and extended logic programs do to literals, Default Logic does to *formulas*. We will look at Default Logic later in the course.

Stable models

For a normal logic program, a stable model is a set of *atoms*.

Let P be a *ground* normal logic program, i.e., one without variables. If P is not ground (contains variables) then replace it by all the ground instances of its clauses. (The resulting set of clauses may not be finite.)

Notation When r is a (ground) clause of the form:

$$A \;\leftarrow\; B_1,\ \ldots,\ B_m,\ \text{not}\ C_1,\ \ldots,\ \text{not}\ C_n$$

$\mathit{head}(r) = A$, $\mathit{body}^+(r) = \{B_1, \ldots, B_m\}$, $\mathit{body}^-(r) = \{C_1, \ldots, C_n\}$. When P is a set of clauses, $\mathit{heads}(P) = \{\, \mathit{head}(r) \mid r \in P \,\}$.

Suppose we have a set X of atoms from the language of P. The idea is that we use X to simplify P by 'partially evaluating' all clauses with nbf-literals against X, and then we see whether the simplified program P^X we are left with (the 'reduct') has a least Herbrand model that coincides with X.

Definition

Let P be a ground normal logic program. Let X be a set of atoms. The *reduct* P^X is the set of clauses obtained from P as follows:

- delete any clause in P that has a condition not A in its body where $A \in X$;
- delete every condition of the form not A in the bodies of the remaining clauses.

That is:

$$P^X \stackrel{\text{def}}{=} \{\, \mathit{head}(r) \leftarrow \mathit{body}^+(r) \mid r \in P,\ \mathit{body}^-(r) \cap X = \emptyset \,\}$$

The reduct P^X is obviously a logic program with no occurrences of negation-by-failure not, i.e., it is a set of *definite* clauses.

X is a *stable model* of P iff it satisfies the following 'stability' equation:

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

where $\mathrm{M}(P^X)$ denotes the *least Herbrand model* of the definite clause progam P^X.

Now what?
It still remains to show that every stable model of P as defined above is indeed a model of P (thus justifying the term 'stable model'). It turns out that this is so, and moreover that every stable model of P is a *minimal* model of P, and a *supported* minimal model of P.
Then we establish some key results for stable models, relate this to other semantics for negation-by-failure, and look at ways of computing stable models.
Later we will do the same for extended logic programs, and after that generalise further when we look at Reiter Default Logic.

Example

$p(1,2) \leftarrow$
$q(X) \leftarrow\ p(X,Y),\ \text{not}\ q(Y)$

Replace these clauses by all their ground instances P:

$p(1,2) \leftarrow$
$q(1) \leftarrow\ p(1,1),\ \text{not}\ q(1)$
$q(1) \leftarrow\ p(1,2),\ \text{not}\ q(2)$
$q(2) \leftarrow\ p(2,1),\ \text{not}\ q(1)$
$q(2) \leftarrow\ p(2,2),\ \text{not}\ q(2)$

Let's try $X_1 = \{q(2)\}$. The reduct P^{X_1} is:

$p(1,2) \leftarrow$
$q(1) \leftarrow\ p(1,1)$
$q(2) \leftarrow\ p(2,1)$

The least Herbrand model $\mathrm{M}(P^{X_1}) = \{p(1,2)\}$. X_1 is not stable.

Let's try $X_2 = \{p(1,2), q(1)\}$. The reduct P^{X_2} is:

$p(1,2) \leftarrow$
$q(1) \leftarrow\ p(1,2)$
$q(2) \leftarrow\ p(2,2)$

The least Herbrand model $\mathrm{M}(P^{X_2}) = \{p(1,2), q(1)\}$. X_2 is stable.

Theorem

Let P be a normal logic program and P^* the set of all its ground instances. Any stable model of P^* is a minimal Herbrand model of P^* (and therefore of P). And it is a supported minimal model of P^*.

Proof Let X be a stable model of P^*.

1) It is straightforward to show that X is a model of P^*. (Consider any clause of P^*. It will be of the form $H \leftarrow A_1, \ldots, A_m, \text{not}\ B_1, \ldots, \text{not}\ B_n$ $(m \geq 0, n \geq 0)$. We need to show that this clause is true in X: if $\{A_1, \ldots, A_m\} \subseteq X$ and $\{B_1, \ldots, B_n\} \cap X = \emptyset$ then $H \in X$ (in other words, $\mathrm{T}_{P^*}(X) \subseteq X$). If $\{B_1, \ldots, B_n\} \cap X \neq \emptyset$ there is nothing to show. Suppose $\{B_1, \ldots, B_n\} \cap X = \emptyset$. In that case $H \leftarrow A_1, \ldots, A_m$ is a clause in the reduct P^{*X}. We know $X = \mathrm{M}(P^{*X})$. So if $\{A_1, \ldots, A_m\} \subseteq X$ then $H \in X$, as required.)

2) Now suppose that some subset X' of X is a model of P^*. It is easy to check that X' is also a model of the reduct P^X. Since by definition X is the least Herbrand model of P^X it follows that $X' = X$.

3) Suppose X is not supported. Then there is an atom $A \in X$ such that A is not the head of a clause in P^* whose body is true in X. But then A could not be the head of a clause in the reduct P^X whose body is true in X, and so X could not be the least Herbrand model of P^X.

Note: Every stable model is minimal and supported. But not the converse: there could be a minimal and supported model that is not stable. (Example later. And see comments on 'Tight programs' at the end.)

Example

$p \leftarrow\ q,\ \text{not}\ r$
$p \leftarrow\ \text{not}\ q, s$
$q \leftarrow$

Try $X = \{p, q\}$. The reduct is

$p \leftarrow\ q$
$q \leftarrow$

Its least Herbrand model is $\{p, q\}$. Stable.

Example

$p \leftarrow\ q, r$
$q \leftarrow$
$r \leftarrow$

(Note no nbf-literals in this program.) Clearly only $\{p, q, r\}$ is a stable model, because this is the least Herbrand model of this program, and in this example the reduct and the original program are identical whatever set of atoms we try.

Example

$p \leftarrow\ p,\ \text{not}\ p$ (What a database !)

Consider $\emptyset$. The reduct is

$p \leftarrow\ p$

Its least Herbrand model is $\emptyset$. So $\emptyset$ is stable.
Consider now $\{p\}$. Now the reduct is $\emptyset$. Clearly the least Herbrand model is $\emptyset$.
$\{p\}$ is not stable.

Example

$p \leftarrow$ not q
$q \leftarrow$ not p

– Is $\{p, q\}$ stable? The reduct is $\emptyset$. Its least H-model is $\emptyset$. No.

– Is $\{p\}$ stable? The reduct is $\{p \leftarrow\}$. Its least H-model is $\{p\}$. Yes.

– Is $\{q\}$ stable? The reduct is $\{q \leftarrow\}$. Its least H-model is $\{q\}$. Yes.

– Is $\emptyset$ stable? The reduct is $\{p \leftarrow, q \leftarrow\}$. Its least H-model is $\{p, q\}$. No.

Note that this program has *two* stable models and that (by symmetry) there is no way to choose between them.

Example

$p \leftarrow q,$ not r
$q \leftarrow r,$ not p
$r \leftarrow p,$ not q

Let's try all the possibilities: $\{p, q, r\}$, $\{p, q\}$, $\{p\}$, $\emptyset$. (The other combinations are covered by symmetry.)

– Is $\{p, q, r\}$ stable? The reduct is $\emptyset$. Its least H-model is $\emptyset$. No.

– Is $\{p, q\}$ stable? The reduct is $\{p \leftarrow q\}$. Its least H-model is $\emptyset$. No.

– Is $\{p\}$ stable? The reduct is $\{p \leftarrow q;\ r \leftarrow p\}$. Its least H-model is $\emptyset$. No.

– Is $\emptyset$ stable? The reduct is $\{p \leftarrow q;\ q \leftarrow r;\ r \leftarrow p\}$. Its least H-model is $\emptyset$. Yes.

(We could have saved ourselves work, of course, by first trying $\emptyset$. Because if $\emptyset$ is a stable model, no other set of atoms can be – stable models are minimal.)

Example

Every stable model is minimal and supported. But not the other way round.

$q \leftarrow$ not p
$p \leftarrow p$

There are two minimal models: $\{q\}$ and $\{p\}$. Both are supported.
But only $\{q\}$ is stable. The reduct with $\{p\}$ is $\{p \leftarrow p\}$, whose least Herbrand model is $\emptyset$.
(Note that this program is stratified.)

Stable models: Relationship to other semantics

Even a program which is not locally stratified can have a unique stable model. See the first example in these notes, for instance. But here is a comforting result to make us feel good.

Theorem

If a normal logic program is *locally stratified* then its unique *stable* model coincides with its unique *'perfect'* model.
Note (2006): The 'perfect' model of a (locally) stratified normal logic program is another kind of semantics, related to a general approach to nonmonotonic default reasoning called 'circumscription'. I have omitted details of 'perfect' models this year.
The point is that when a normal logic program is stratified (and not just locally stratified) then its 'perfect' model is the same as its ABW 'iterated fixpoint' model. So we have . . .

Corollary

If a normal logic program is *stratified* then its unique *stable*, *'perfect'* and *iterated fixpoint (ABW)* models all coincide. If a program is *definite*, its unique *minimal (least) Herbrand*, *stable*, *'perfect'* and *iterated fixpoint (ABW)* models all coincide.

We also know (earlier)

- every stable model of P is a supported model of P
- and a minimal model of P

and for a finite logic program P:

- the supported models of P are the models of comp(P)

So we also know

- every stable model of P is a minimal model of comp(P)

(but not the converse)

Concluding remarks

- Stable models provide a very simple and intuitive semantics for negation-by-failure in normal logic programs. (There are other reasons to think its a natural semantics, because of the connection to 'autoepistemic logic' referred to above. Here, one reads not A as 'it is not the case that it can be proved that A'. Further details omitted this year.)

- Stable model semantics has been extended to the wider class of *disjunctive logic programs* which are sets of clauses of the form

  $$A_1 \vee \cdots \vee A_m \leftarrow\ L_1 \wedge \cdots \wedge L_n \qquad (m \geq 1; n \geq 0)$$

  where the A_i are *atoms* and where the L_j are *nbf-literals* (atoms or atoms preceded by not). We won't look at disjunctive logic programs.

- Stable model semantics has also been extended to *extended logic programs* which are sets of clauses of the form

  $$L \leftarrow\ L_1, \ldots, L_n \qquad (n \geq 0)$$

  where L is a *literal* (an atom or an atom preceded by $\neg$) and where every L_i is of the form A, not A, $\neg A$, or not $\neg A$, where A is an atom. We will look at extended logic programs next.

  For extended logic programs, 'stable model semantics' is now usually referred to as 'answer set semantics'. Since the underlying ideas are exactly the same for 'normal programs', I tend to use 'stable models' and 'answer sets' as synonyms. Even though the ideas are exactly the same, the terms 'stable model' is not appropriate when we look at extended logic programs (classical negation mixed with negation-as-failure). More about this later.

- As presented so far, the practical value of stable models is somewhat limited. First there is the problem of generating all the ground instances, especially when there are function symbols in the program (and so infinitely many ground instances). Moreover, the 'stability equation' only tells us how to *check* whether a candidate set of atoms is stable or not. We need some systematic way of generating the candidates without having to guess. In the worst case we could try all the subsets of the Herbrand base (the set of atoms in the program, remember) in turn. This is clearly not practical. But a bit of thought and a bit of experience with some simple examples quickly suggests a number of short cuts that can be turned into a systematic procedure.

  In the next section we will look at a general method called *splitting sets*.

Stable models: Summary

- To test whether a set of atoms X is a stable model of a normal logic program P:
  - construct the reduct P^X: generate all ground instances of P if necessary, then

    $$P^X \ = \{\, head(r) \leftarrow body^+(r) \mid r \in P,\ body^-(r) \cap X = \emptyset \,\}$$

  - test whether $\quad X = \mathrm{M}(P^X)$

- Every stable model of P is a minimal supported Herbrand model of P.

  The converse does not hold. A minimal and supported model is not necessarily stable. (There was an example above.)

  And remember: the supported models of P are the models of comp(P) (when P is finite).

- Stable models need not be unique, and may not exist.

- For normal logic programs where other semantics for negation-by-failure are defined (stratified, in particular) stable model semantics and the other semantics agree.

  In particular, every stratified program has a *unique* stable model (which is the same as its 'ABW iterated fixpoint model').

Generating rules

Let P be a normal logic program and X a set of atoms. The *generating rules* $gen(P, X)$ are the clauses of P whose bodies are satisfied in X:

$$gen(P, X) \ \stackrel{\text{def}}{=}\ \{\, r \in P \mid body^+(r) \subseteq X,\ body^-(r) \cap X = \emptyset \,\}$$

Note, straight from the definition: $\mathrm{T}_P(X) \ = \ heads(\, gen(P, X)\,)$.
Also, easy to check: $(gen(P, X))^X = gen(P^X, X)$.

Answer sets and their generating rules always come in pairs:

Theorem Let P be a normal logic program and X a set of atoms.

- If X is an answer set of P then X is the unique answer set of $gen(P, X)$.
- X is an answer set of P iff X is an answer set of $gen(P, X)$.
- If X is an answer set of P then $X = heads(\, gen(P, X)\,)$.

These are really just restatements of results we already saw, or bits of their proofs. For instance the last item says just that if X is an answer set of P then $\mathrm{T}_P(X) = X$, i.e., X is a model of P and is supported. We already knew that.

Tight programs (For interest—no need to memorise!!)

We know that every stable model (answer set) of P (if P is finite)

- is a minimal supported model of P, or equivalently
- is a minimal model of comp(P)

But not necessarily the other way round.

The mismatch between supported models and stable models is caused by cyclic derivations. The following observation might be helpful. There is no need to *memorise* it. It is taken from notes *Answer Set Programming* by Torsten Schaub, Martin Gebser, Marius Schneider (University of Potsdam).

Positive atom dependency graph

Just like the dependency graph, except only the *refers*$^+$ *to* edges.

- *atoms*(P) at nodes;
- an edge from p to q when there is a clause r in P s.t. *head*(r) $= p$ and $q \in$ *body*$^+$(r):

$$p \leftarrow \; \cdots, q, \cdots \qquad (\text{‘}p \textit{ refers}^+ \textit{ to } q\text{’})$$

(Some people draw the edges the other way round. I read them as ‘p depends on q’. It doesn’t matter, as long as you are consistent, obviously.)

Examples

$$P_1 = \left\{ \begin{array}{ll} a \leftarrow \text{not } b & b \leftarrow \text{not } a \\ c \leftarrow a,\ \text{not } d & d \leftarrow a,\ \text{not } c \\ e \leftarrow c,\ \text{not } a & e \leftarrow d,\ \text{not } a \end{array} \right\}$$

$$\begin{tikzcd}
& e \arrow[dl] \arrow[dr] & \\
c \arrow[dr] & & d \arrow[dl] \\
& a & b
\end{tikzcd}$$

Answer sets (Stable models): $\{\, \{a, c\}, \{a, d, e\}, \{b\} \,\}$
Supported models: $\{\, \{a, c\}, \{a, d, e\}, \{b\} \,\}$

$$P_2 = \left\{ \begin{array}{ll} a \leftarrow \text{not } b & b \leftarrow \text{not } a \\ c \leftarrow \text{not } a & c \leftarrow d \\ d \leftarrow a,\ b & d \leftarrow c \end{array} \right\}$$

$$\begin{tikzcd}
c \arrow[r, bend left] & d \arrow[l, bend left] \arrow[dl] \arrow[d] \\
a & b
\end{tikzcd}$$

Answer sets (Stable models): $\{\, \{a\}, \{b, c, d\} \,\}$
Supported models: $\{\, \{a\}, \{b, c, d\}, \{a, c, d\} \,\}$

Tight programs

- A normal logic program P is *tight* iff its positive atom dependency graph is *acyclic*.
- If a normal logic program P is tight, then

$$\begin{array}{ll} X \text{ is a stable model (answer set) of } P & \text{iff } X \text{ is a supported model of } P \\ & \text{iff } X \text{ is a model of comp}(P) \end{array}$$

That is, for tight programs, stable models and (minimal) supported models coincide.