Relationships Among Semantics

If a program + EDB has a stratified or perfect (locally stratified) model, then that is the unique stable model.

- A program + EDB can have a unique stable model even if there is no perfect model.

Example

```
p :- NOT q
q :- NOT p
p :- NOT p
```

- Only $\{p\}$ is a stable model.
- Note that without the 3rd rule, both $\{p\}$ and $\{q\}$ are stable.

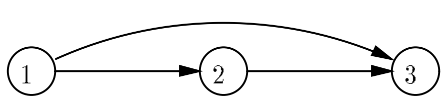
Why Stratified Models are Stable

Intuition to prove stratified model M is stable:

- Divide M into strata $M_0, M_1, \ldots$ (M_0 includes the EDB).
- LFP of stratum 0 does not involve negation, so its instantiated rules survive to the inference step of the GL transform, and exactly M_0 is inferred.
- Thus, M_0, and nothing else for stratum 0, is in $GL(M)$.
- Consider what happens computing the stratified model at stratum 1.
    - Negated subgoals are resolved according to M_0.
    - Instantiated rules with a negated member of M_0 effectively disappear, and those with a negated nonmember of M_0 effectively lose that subgoal.
- Thus, the LFP for stratum 1 looks just like the GL transform for the relevant instantiated rules.

Example

Let's revisit the locally stratified "Win" example:



whose relevant instantiated rules were:

```
r1: win(1) :- move(1,2) & NOT win(2)
r2: win(1) :- move(1,3) & NOT win(3)
r3: win(2) :- move(2,3) & NOT win(3)
```

- Recall stratum $0 = win(3)$ + EDB facts; stratum $1 = win(2)$; stratum $2 = win(1)$.
- Also recall: the stratified model is $\{win(1), win(2)\}$ + the EDB; we must show this model is also stable.

- Stratum 0: No relevant rules, so $win(3)$ is false. Likewise, $win(3)$ is not inferred in the GL procedure.
- Stratum 1: Since $win(3)$ is false, the rules for stratum 1 become:

  ```
  r3: win(2) :-
  ```

  $win(2)$ is inferred, both in the stratified and GL procedures.
- Stratum 2: Since $win(2)$ is true and $win(3)$ is false, the rule for stratum 2 becomes:

  ```
  r2: win(1) :-
  ```

Again, the same thing happens to this rule in the GL procedure, so we infer $win(1)$ in both stratified and stable approaches.

Well-Founded Model

- 3-valued model: true, false, unknown.
- WF model has positive facts like $p(1)$ and negative facts like $\neg p(1)$.
- IDB ground atoms not mentioned are assumed to be "unknown."
- EDB ground atoms not mentioned are assumed false.

Two Modes of Inference

1. If body is true, infer head.
2. Look for *unfounded sets*: If, after instantiating rules in all possible ways and eliminating those with a known false subgoal, there is a set U of positive ground atoms such that every rule with a member of U in the head has a member of U as one of its subgoals, then U is unfounded.

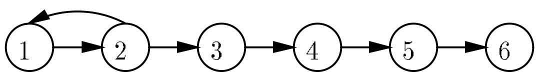
   - We cannot prove any member of U, because we would have to prove another member first.

   - ❖ In WF semantics, we infer the negation of all members of U.

- Repeat inference modes until no new inferences of either type are possible.

Example

"Win" rule with EDB $1 \to 2$, $2 \to 1$, $2 \to 3 \to 4 \to 5 \to 6$.



- Start by instantiating the Win rule

```
win(X) :- move(X,Y) & NOT win(Y)
```

  in all possible ways.

   - ❖ Eliminate rules with false bodies.
   - ❖ Also eliminate true subgoals from remaining bodies.
   - ❖ For convenience, eliminate rules whose head has already been inferred.

```
win(1) :- NOT win(2)
win(2) :- NOT win(1)
win(2) :- NOT win(3)
win(3) :- NOT win(4)
win(4) :- NOT win(5)
win(5) :- NOT win(6)
```

Round 1: No positive inferences. Largest unfounded set = $\{win(6)\}$. Infer $\neg win(6)$.

Round 2: Infer $win(5)$. Delete last two rules. One now has a false subgoal, the other an already-inferred head.

```
win(1) :- NOT win(2)
win(2) :- NOT win(1)
win(2) :- NOT win(3)
win(3) :- NOT win(4)
```

Largest unfounded set = $\{win(4)\}$. Infer $\neg win(4)$.

Round 3: Infer $win(3)$, delete last two rules.

```
win(1) :- NOT win(2)
win(2) :- NOT win(1)
```

Now, no unfounded sets, so done.

- WF model is

$$\{win(3), win(5), \neg win(4), \neg win(6)\}$$

* Truth value of $win(1)$ and $win(2)$ is "unknown."

Example

```
p :- q
q :- p
r :- p & q
s :- NOT p & NOT q
```

* $\{p, q\}$ is an unfounded set.
* But $\{p, q, r\}$ is the *largest* unfounded set.
    * Note: union of unfounded sets is unfounded, so there is always a largest.
* WF model: $\{\neg p, \neg q, \neg r, s\}$.

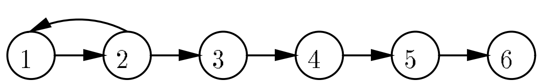
Alternating Fixed Point

1. Instantiate rules in all possible ways.
2. Eliminate rules with false EDB or arithmetic subgoal; eliminate true EDB and arithmetic subgoals from remaining rules for convenience.
3. Initialize all IDB ground atoms to false.
4. Repeatedly evaluate IDB subgoals by applying the GL transform to the model consisting of the EDB + an IDB based on the previous round's true/false decisions.

* In the limit, IDB ground atoms that converge to true are true. Those that converge to false are false. Those that oscillate are unknown.

Example

Repeating above "Win" example:



Rules processed by (1) and (2):

```
win(1) :- NOT win(2)
win(2) :- NOT win(1)
win(2) :- NOT win(3)
win(3) :- NOT win(4)
win(4) :- NOT win(5)
win(5) :- NOT win(6)
```

Using alternating fixed point:

Round	0	1	2	3	4	5
$win(1)$	0	1	0	1	0	1
$win(2)$	0	1	0	1	0	1
$win(3)$	0	1	0	1	1	1
$win(4)$	0	1	0	0	0	0
$win(5)$	0	1	1	1	1	1
$win(6)$	0	0	0	0	0	0

Another Example

`p :- q; q :- NOT p`

Round	0	1	2	3	4	5
p	0	1	0	1	0	1
q	0	1	0	1	0	1

- Round 0: Both are 0 as always.
- Round 1: Rules simplify to `p :- q; q :-`. Infer both q and p.
- Round 2: Rules simplify to `p :- q`. No inference possible!
- Round 3 and later: Repeats.
- Conclude both p and q are "unknown."

Relationships Among Semantics

- If there is a 2-valued WF model, it is the unique stable model.
- If there is a perfect model (i.e., program + EDB is locally stratified), then this model is also the stable and WF model, and obviously is 2-valued.
- There can be a 3-valued WF model when there is no stable semantics (i.e., no unique stable model).

Example

Win program with EDB `move(1,2)`, `move(2,1)`.

- Two stable models, $\{win(1)\}$ and $\{win(2)\}$. Thus, a stable semantics does not exist.
- However, the WF model exists and makes both $win(1)$ and $win(2)$ "unknown."

- There can be a unique stable model $\neq$ WF model.

Example

```
p :- NOT q
q :- NOT p
p :- NOT p
```

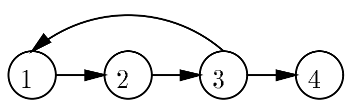
- WF model makes p, q "unknown."
- $\{p\}$ is the only stable model.

- There can be a 2-valued WF model when there is no locally stratified model.

Example

```
win(X) :- move(X,Y) & NOT win(Y)
```

with *move* defined by



- There is a cycle among 1, 2, and 3, so this program and EDB is not locally stratified.

- However, the WF model is 2-valued, intuitively because the cycle is not followed from board 3 on best play.

```
win(1) :- NOT win(2)
win(2) :- NOT win(3)
win(3) :- NOT win(1)
win(3) :- NOT win(4)
```

Round	0	1	2	3	4
$win(1)$	0	1	0	1	1
$win(2)$	0	1	0	0	0
$win(3)$	0	1	1	1	1
$win(4)$	0	0	0	0	0

Comparisons Between Stable and WF Approaches

- In "win" program, true = forced win; false = forced loss; unknown = draw with best play.
- However, in "cafeteria" example to follow, the rules are the same but the intuitive semantics favors the stable approach.

Example

Consider a collection of buildings:

- Buildings have either lounges or cafeterias, not both.
- No adjacent buildings both have cafeterias.
- If a building does not have a cafeteria, then an adjacent building must have one.

```
lounge(X) :- adj(X,Y) & cafeteria(Y)
cafeteria(X) :- NOT lounge(X)
```

If we get rid of "cafeteria":

```
lounge(X) :- adj(X,Y) & NOT lounge(Y)
```

Looks just like "win."

- Problem is really: find a maximal independent set of buildings in which to put cafeterias.
- The stable models *are* the maximal independent sets.
- But the WF model makes `cafeteria(X)` true iff X is in *every* maximal independent set.

Comparison of Complexity for Stable Versus WF

- It is NP-hard to tell whether a propositional logic program has a stable semantics, i.e., a unique stable model.
- It is polynomial to construct the WF model.
- Same comments hold for first-order logic, but with complexity measured in terms of the EDB size rather than the number of propositions.

Modularly Stratified Semantics

- Motivation: largest known class of Datalog-with-negation programs for which magic-sets (query optimization technique to be discussed) works.
- Must be able to partition the rules into "modules," such that

  1. All recursion is within a module.

2. All modules have locally stratified semantics with respect to the EDB and the previously computed models for any lower modules.

   ❖ I.e., treat all true facts belonging to lower modules as if they were EDB facts.

- Modularly stratified semantics = what we get by computing locally stratified semantics for modules, bottom up.
- Note modules are partially ordered by dependence among their predicates, because all recursion must take place within a single module.

Example

Consider:

```
win(X) :- move(X,Y) & NOT win(Y)
```

with *move* relation $\{move(1,2)\}$.

- We might appear to have a cycle in the instantiated rules

  r_1: `win(1) :- move(1,2) & NOT win(2)`
  r_2: `win(2) :- move(2,1) & NOT win(1)`

  But the fact that $move(2,1)$ is false removes r_2.

- The only dependence is $win(1) \rightarrow win(2)$, and this program + EDB is locally stratified.

Example (Continued)

- Next, suppose we add an IDB predicate *move*1 to be identical to *move*:

```
win(X) :- move1(X,Y) & NOT win(Y)
move1(X,Y) :- move(X,Y)
```

Now, with the same EDB, we instantiate the *win* rule as:

r_1: `win(1) :- move1(1,2) & NOT win(2)`
r_2: `win(2) :- move1(2,1) & NOT win(1)`

- It is not apparent that $win(2)$ does not depend on $win(1)$, so it looks like we have a cycle in the dependency graph.

  ❖ The difference is that *move*1 is IDB, while *move* is EDB.

- This program + EDB is not locally stratified.

- However, the program + EDB is modularly stratified, because we can group *move* and *move*1 into a module and *win* into a higher module.
- Module for *move* and *move*1: We first compute the locally stratified model for *move*1, which is $\{move1(1,2)\}$.
- Module for *win*: We discover that r_2 can be removed, because $move1(2,1)$ is known to be false.
- Thus, there is no cycle, and the *win* module is also locally stratified.
- Thus, the whole program + EDB is modularly stratified.

Summary of Semantics

- Definition A above definition B means that every program + EDB that has a semantics in B
  1. Has a semantics according to A, and
  2. The meaning (chosen model) is the same for both A and B.