## Local Stratification

- *Instantiate* rules; i.e., substitute all possible constants for variables, but reject instantiations that cause some EDB subgoal to be false.

    ❖ *Ground atom* = atom with no variables.

- Build dependency graph at the level of ground atoms by instantiating the rules.

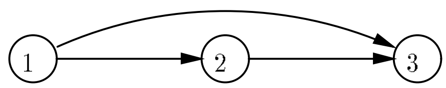
- Whether program + EDB is locally stratified depends not only on program, but on EDB.

- Program + EDB is *locally stratified* iff no negative cycles in dependency graph.

---

## Example

Win program with boards $\{1, 2, 3\}$ and moves $1 \rightarrow 2$, $2 \rightarrow 3$, and $1 \rightarrow 3$.

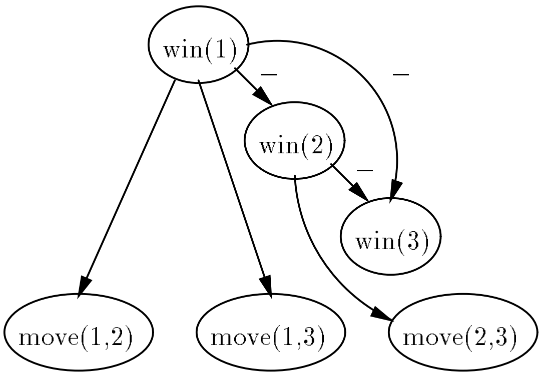



- The following three instantiations are the only ones that cannot be ruled out immediately by a false *move* subgoal:

    $r_1$: `win(1) :- move(1,2) & NOT win(2)`
    $r_2$: `win(1) :- move(1,3) & NOT win(3)`
    $r_3$: `win(2) :- move(2,3) & NOT win(3)`

---

## The Dependency Graph




- The three *move* ground atoms and $win(3)$ are in stratum 0; $win(2)$ is in stratum 1, and $win(1)$ is in stratum 2.

---

### Computing the Locally Stratified Model

Compute *locally stratified* (*"perfect"*) model bottom-up, deciding on the truth or falsehood of atoms by computing the LFP of each stratum in turn.

### Example

Stratum 0: We find $win(3)$ is false.

Stratum 1: That lets us use $r_3$ to infer $win(2)$ is true.

Stratum2: We then use $r_2$ to infer $win(1)$ is true.

---

### Stable Models

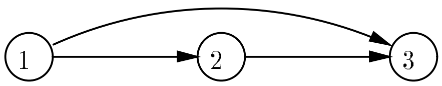
Intuitively, model $M$ is "stable" if when you apply the rules to $M$ you get exactly $M$ back.

### Example

"Win" rule:

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

with EDB $1 \rightarrow 2$, $2 \rightarrow 3$, $1 \rightarrow 3$.




- $M$ = EDB + $\{win(1), win(2)\}$ is stable.
- The three useful instantiations are
  - $r_1$: `win(1) :- move(1,2) & NOT win(2)`
  - $r_2$: `win(1) :- move(1,3) & NOT win(3)`
  - $r_3$: `win(2) :- move(2,3) & NOT win(3)`
- $M$ makes only the bodies of $r_2$ and $r_3$ true, letting us infer exactly $M$.
- Note you get the EDB facts "for free" in this process.

---

### Gelfond-Lifschitz Transform

Formal notion of applying rules to a model $M$.

1. Instantiate rules in all possible ways.
2. Delete instantiated rules with a (nonnegated) EDB subgoal that is not in $M$ or with a false arithmetic subgoal.
   - Remember, EDB is part of $M$.

3. Delete instantiated rules with a subgoal `NOT p(x)`, where $p(x)$ is in $M$.

   ✥ In (3) and (4), $p$ can be either EDB or IDB.

4. Delete any subgoal `NOT p(x)` if $p(x)$ is not in $M$.

5. Delete any EDB subgoal in $M$ and any true arithmetic subgoal.

---

- What's left? Rules with zero or more nonnegated, relational subgoals with IDB predicates.

  ✥ Note that a rule with empty body is an assertion that the head is true.

- $GL(M)$ = EDB + result of inferring IDB with the remaining rules.

**Bottom Line on GL Transform**

You can use negative EDB or IDB facts in $M$ (i.e., atoms missing from $M$) to help infer facts, and you use positive EDB facts, but you *don't* use the positive IDB facts in $M$ unless you *derive* them from other facts.

---

**Formal Definition of Stable Models**

- If $GL(M) = M$, then $M$ is *stable*.
- The "stable semantics" for a program + EDB is the unique stable model with that EDB, if there is one.

  ✥ Sometimes it is interesting to look at the *set* of stable models, as well.

---

**Example**

$M = \{move(1,2),\ move(1,3),\ move(2,3),\ win(1),\ win(2)\}$ (formal version of previous example).

- After step (2):

  $r_1$: `win(1) :- move(1,2) & NOT win(2)`
  $r_2$: `win(1) :- move(1,3) & NOT win(3)`
  $r_3$: `win(2) :- move(2,3) & NOT win(3)`

- After step (3):

  $r_2$: `win(1) :- move(1,3) & NOT win(3)`
  $r_3$: `win(2) :- move(2,3) & NOT win(3)`

- After step (4):

```
r2: win(1) :- move(1,3)
r3: win(2) :- move(2,3)
```

- After step (5):

  ```
  r2: win(1) :-
  r3: win(2) :-
  ```

- Thus, $GL(M) = \{win(1), win(2)\}$ + EDB = $M$.
- $M$ is a stable model.

---

**Example**

Consider the "program":

```
p(X) :- p(X)
```

- $\emptyset$ is the only stable model.
- Why? The only instantiated rules are of the form `p(a) :- p(a)`.
  - ✥ The GL transform doesn't affect these, no matter what $M$ is.
  - ✥ Thus, there is no way to infer any $p(a)$.

---

**Example**

For any Datalog program *without* negation, the unique LFP is the only stable model.

- Why? To test whether $M$ is stable, we compute $GL(M)$.
  - ✥ Since there is no negation in bodies, the surviving instantiated rules are exactly the ones with true EDB subgoals.
  - ✥ Thus, GL infers exactly the LFP for the EDB portion of $M$, regardless of what $M$ is.
  - ✥ If we start with the LFP, we infer it, so that model is stable; if we start with another model, we still infer the LFP, so that model is *not* stable.

---

**Propositional Stable Models**

It is often useful to find propositional examples.

- No EDB in propositional logic.
- Thus, only steps (3) and (4), plus the final inference, are relevant for the GL transform.

---

**Example**

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

- $M = \{p, q\}$.
- After step (3):

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

- After step (4):

```
p :- q; q :- ; r :- s
```

- Inference: $GL(M) = \{p, q\} = M$.

**Multiple Stable Models Possible**

Notice that $\{r, s\}$ is also a stable model of the above rules.