Semantics of Datalog With Negation

Local Stratification

Stable Models

Well-Founded Models

The Story So Far --- 1

- When there is no (IDB) negation, there is a unique minimal model (least fixedpoint), which is the accepted meaning of the Datalog program.
- With negation, we often have several minimal models, and we need to decide which one is meant by the program.

The Story So Far --- 2

- When the program is stratified, one minimal model is the stratified model.
  - This model appears in all cases to be the one we intuitively want.
  - Important technical point: if the program actually has no negation, then the stratified model is the unique minimal model.
  - Thus, stratified semantics extends least-fixedpoint semantics.

What About Unstratified Datalog?

- There are some more general conditions under which an "accepted" choice among models exists.
- From least to most general: Locally stratified models, modularly stratified models, stable/well-founded models.

Why Should We Care?

1. Solidify our understanding of when declarative assertions, like logical rules, lead to a meaningful description of something.
2. SQL recursion really deals with ambiguities of the same kind, especially regarding aggregations, as well as negation.

Ground Atoms

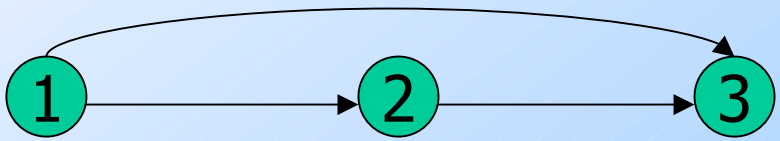
- All these approaches start by *instantiating* the rules: replace variables by constants in all possible ways, and throw away instances of the rules with a known false EDB subgoal.
- An atom with no variables is a *ground atom*.
  - Like propositions in propositional calculus.

Example: Ground Atoms

- Consider the Win program:

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

with the following moves:




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

Example --- Continued

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

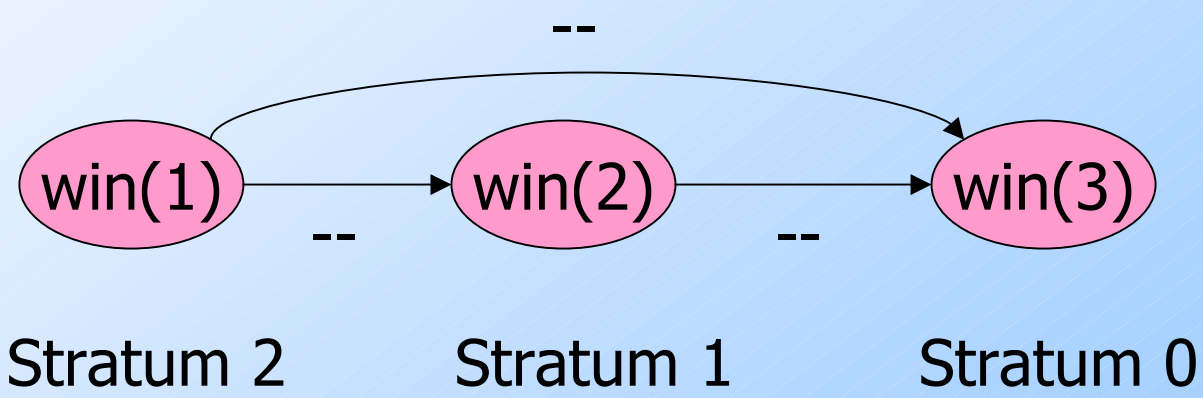
- Other instantiations of the rule have a false `move` subgoal and therefore cannot infer anything.
- win(1), win(2), and win(3) are the only relevant IDB ground atoms for this game.

Locally Stratified Models

1. Build *dependency graph* with:
   - Nodes = relevant IDB ground atoms.
   - Arc *p* -> *q* iff *q* appears in an instantiated body with head *p*.
   - Label – on arc if *q* is negated.
2. Stratum of each node defined as before.
3. *Locally stratified* = finite strata only.

Example

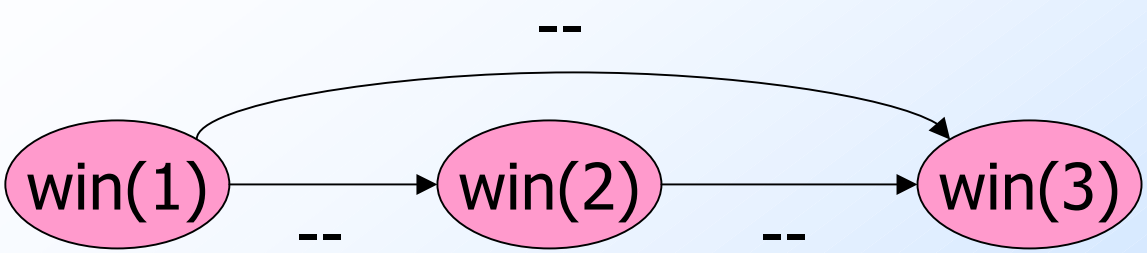
```
win(1) :- move(1,2) & NOT win(2)
win(1) :- move(1,3) & NOT win(3)
win(2) :- move(2,3) & NOT win(3)
```



Locally Stratified Model

```
Include all EDB ground atoms;
FOR (stratum i = 0, 1, …) DO
  WHILE (changes occur) DO
     IF (some rule for some p at
     stratum i has a true body)
     THEN add p to model;
```

Example




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

1. No rules for win(3); therefore false.
2. win(2) rule satisfied; therefore true.
3. Second rule for win(1) satisfied, therefore win(2) also true.

Stable Models --- Intuition

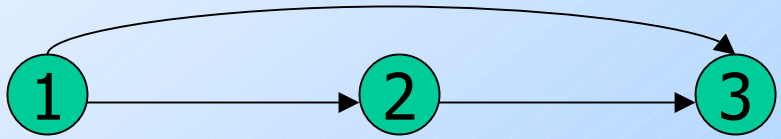
- A model *M* is "stable" if, when you apply the rules to *M* and add in the EDB, you infer exactly the IDB portion of *M*.
- But … when applying the rules to *M*, you can only use non-membership in *M*.

Example

- The Win program again:

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

with moves:



- M = EDB + {win(1), win(2)} is stable.
  - $Y = 3$, $X = 1$ yields win(1).
  - $Y = 3$, $X = 2$ yields win(2).
  - Cannot yield win(3).

Gelfond-Lifschitz Transform (Formal Notion of Stability)

1. Instantiate rules in all possible ways.
2. Delete instantiated rules with any false EDB or arithmetic subgoal (incl. NOT).
3. Delete instantiated rules with an IDB subgoal NOT p(…), where p(…) is in *M*.
4. Delete subgoal NOT p(…) if p(…) is not in *M*.
5. Delete true EDB and arithmetic subgoals.

GL Transform --- Continued

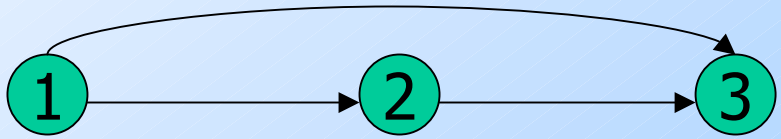
- Use the remaining instantiated rules plus EDB to infer all possible IDB ground atoms.
- Then add in the EDB.
- Result is GL(*M*).
- *M* is *stable* iff GL(*M*) = *M*.

Example

- The Win program yet again:

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

with moves:

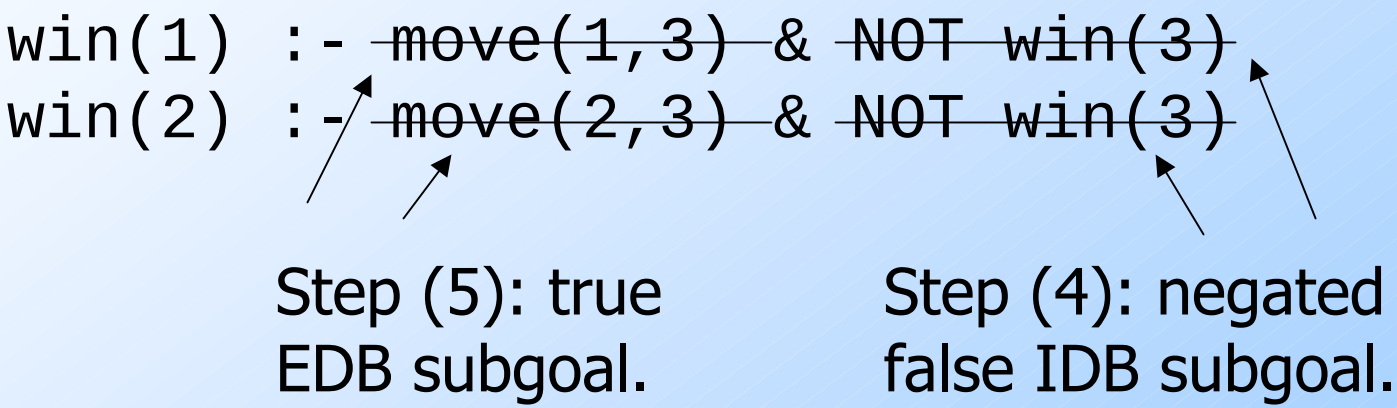


- M = EDB + {win(1), win(2)}.
- After steps (1) and (2):

Step (3): negated true IDB.

~~`win(1) :- move(1,2) & NOT win(2)`~~
`win(1) :- move(1,3) & NOT win(3)`
`win(2) :- move(2,3) & NOT win(3)`

Example --- Continued



Remaining rules have true (empty) bodies.
Infer win(1), win(2).
GL(*M*) = EDB + {win(1), win(2)} = *M*.

Bottom Line on the GL Transform

- You can use (positive or negative) IDB and EDB facts from *M* to satisfy or falsify a negated subgoal.
- You can use a positive EDB fact from *M* to satisfy a positive subgoal.
- But you can only use a positive IDB fact to satisfy a positive IDB subgoal if that fact has been derived in the final step.

To Make the Point Clear…

- If all I have is the instantiated rule p(1) :- p(1), then M = {p(1)} is not stable.
- We cannot use the membership of positive IDB subgoal p(1) in M to make the body of the rule true.

The Stable Model

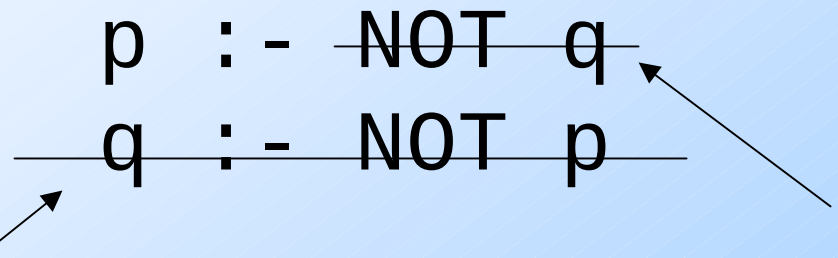
- If a program and EDB has exactly one model *M* with that EDB that is stable, then *M* is *the stable model* for the program and EDB.
- Otherwise, there is no stable model for this program and EDB.

Propositional Datalog

- Many useful examples use propostional variables (0-ary predicates).
- All propositional variables are IDB.
- For GL transform, just:
  1. Eliminate rules with a negated variable that is in *M*.
  2. Eliminate subgoal NOT *p* if *p* in *M*.
  3. Run the deduction step.

Example: p:- NOT q, q :- NOT p

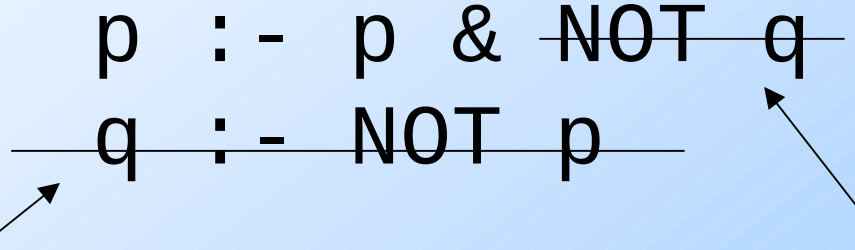
{*p*} is stable.



Infer only *p*.
Thus, {*p*} is stable.
Unfortunately, so is {*q*}.
Thus, this program has no stable model.

p:- p & NOT q, q :- NOT p

{*p*} is <u>not</u> stable.



Cannot infer *p* !
GL({p}) = ∅.
{*p*} is not stable.

Example --- Continued

{*q*} is stable.

~~p :- p & NOT q~~

q :- ~~NOT p~~

Eliminate rule with a false negated subgoal.

Eliminate true negated subgoal.

Infer only *q*.

Thus, {*q*} is stable.

GL({*p*,*q*}) = ∅; GL(∅) = {*q*}.

Thus, {*q*} is the (unique) stable model.

3-Valued Models

- Needed for "well-founded semantics."
- Model consists of:
  1. A set of true EDB facts (all other EDB facts are assumed false).
  2. A set of true IDB facts.
  3. A set of false IDB facts (remaining IDB facts have truth value "unknown").

Well-Founded Models

- Start with instantiated rules.
- *Clean* the rules = eliminate rules with a known false subgoal, and drop known true subgoals.
- Two modes of inference:
  1. "Ordinary": if the body is true, infer head.
  2. "Unfounded sets": assume all members of an unfounded set are false.

Unfounded Sets

- *U* is an unfounded set (of positive, ground, IDB atoms) if every remaining instantiated rule with a member of *U* in the head also has a member of *U* in the body.
- Note we could never infer any member of *U* to be true.
- But assuming them false is still "metalogic."

Example

```
p :- q, q :- p
```

- {p,q} is an unfounded set.
- So is ∅.
- Note the property of being an unfounded set is closed under union, so there is always a unique, maximal unfounded set.

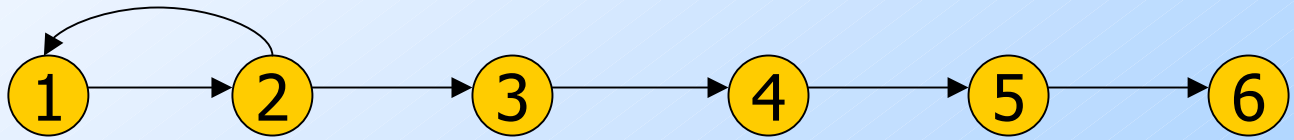
Constructing the Well-Founded Model

```
REPEAT
  “clean” instantiated rules;
  make all ordinary inferences;
  “clean” instantiated rules;
  find the largest unfounded set
          and make its atoms false;
UNTIL no changes;
make all remaining IDB atoms
     “unknown”;
```

Example

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

with these moves:



Instantiated, Cleaned Rules

```
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)
```

No ordinary inferences.

{win(6)} is the largest unfounded set. Infer NOT win(6).

Second Round

```
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) :-
```

Infer win(5).

{win(4)} is the largest unfounded set. Infer NOT win(4).

Third Round

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

Infer win(3).

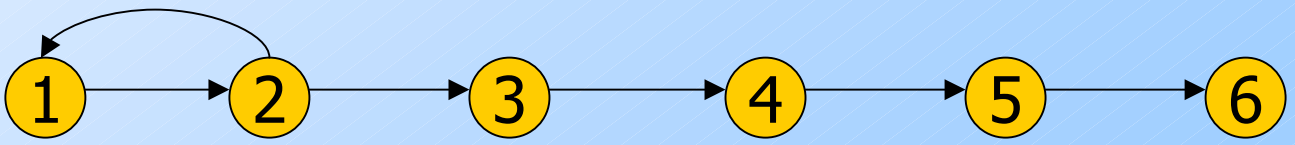
No nonempty unfounded set, so done.

Example --- Concluded

- Well founded model is:

{win(3), win(5), NOT win(4), NOT win(6)}.

- The remaining IDB ground atoms --- win(1) and win(2) --- have truth value "unknown."
- Notice that if both sides play best, 3 and 5 are a win for the mover, 4 and 6 are a loss, and 1 and 2 are a draw.



Another Example

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

- First round: no ordinary inferences.
- {p, q} is an unfounded set, but {p, q, r} is the largest unfounded set.
- Second round: infer s from NOT p and NOT q.
- Model: {NOT p, NOT q, NOT r, s}.

Alternating Fixedpoint

1. Instantiate and "clean" the rules.
2. In "round 0," assume all IDB ground atoms are false.
3. In each round, apply the GL transform to the EDB plus true IDB ground atoms from the previous round.

Alternating Fixedpoint --- 2

- Process converges to an alternation of two sets of true IDB facts.
- Even rounds only increase; odd rounds only decrease the sets of true facts.
- In the limit, true facts are true in both sets; false facts are false in both sets, and "unknown" facts alternate.

Previous Win Example


```
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)
```

Computing the AFP

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

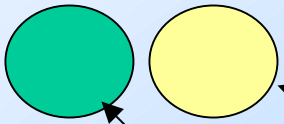
Another Example

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

Round	0	1	2
p	0	0	0
q	0	0	0
r	0	0	0
s	0	1	1

Yet Another Example

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

Round	0	1	2
p	0		
q	0	1	0

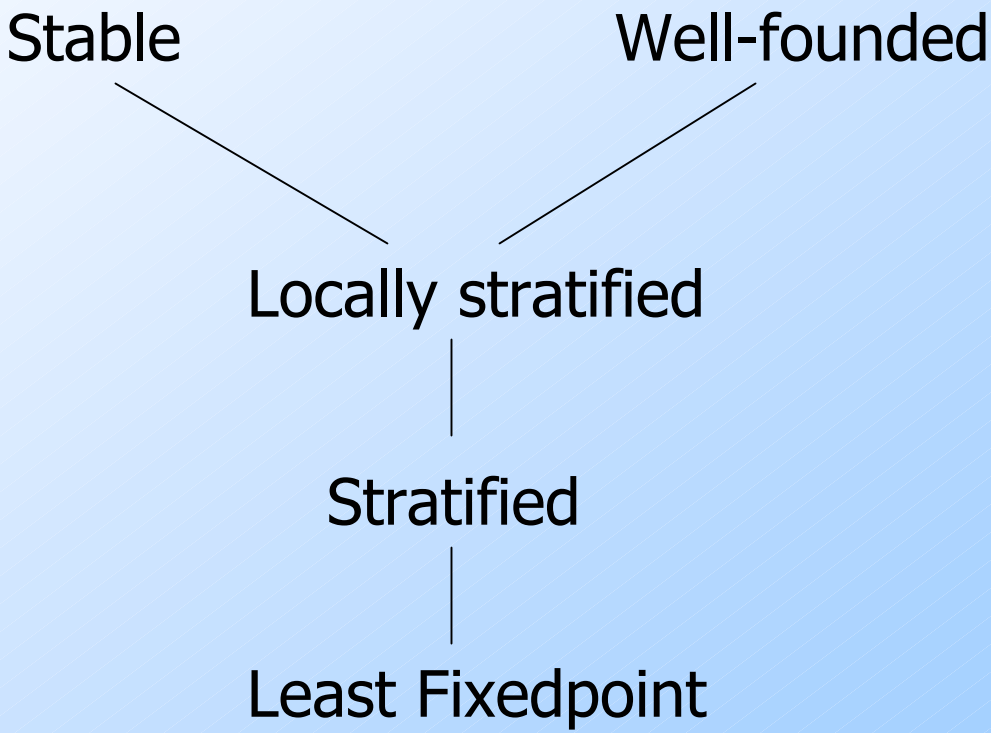
Notice how p is inferred only after we infer q on this round

Notice that we may not use positive IDB fact q from previous round to infer p.

Containment of Semantics

- Say method *A* < method *B* if:
  1. Whenever a program has a model according to method *A*, it has the same model under method *B*.
  2. There is at least one program that has a model under *B* but not under *A*.
- Draw *B* above *A* in diagrams.

Comparison of Semantics



LFP < Stratified

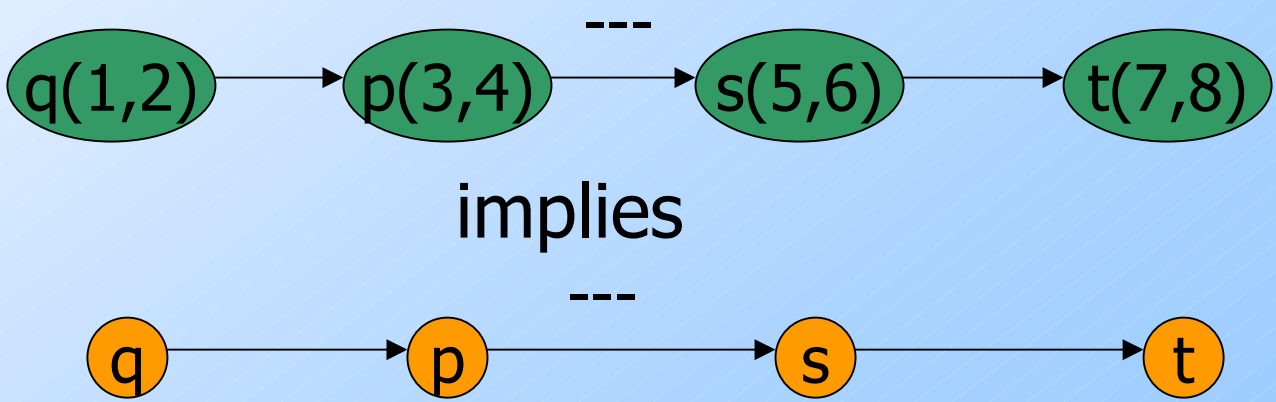
- LFP only applies to Datalog without negation.
- What does Stratified do when there is no negation?
- Everything is in stratum 0, and the whole IDB is computed by the LFP algorithm.
- So Stratified model = LFP model.

Stratified < Locally Stratified

- Consider a Datalog program *P* with a stratified model, *S*. Need to show:
  1. *P* is locally stratified.
  2. Fact q(…) in *S* is in the locally stratified model *L*.
  3. Fact q(…) in *L* is also in *S*.

P Is Locally Stratified

- Key idea: path with *n* ---'s starting at q(…) in dependency graph of ground atoms implies a path with *n* ---'s starting at *q* in the dependency graph of predicates.



P Is Locally Stratified --- 2

- Thus, stratum of ground atom q(…) is no greater than the stratum of predicate *q*.
- If the strata of all predicates in *P* are finite, then the strata of all ground atoms are finite.
  - I.e., if *P* is stratified, it is locally stratified for any EDB.

q(…) in *S* iff q(…) in *L*

- Proof = induction on stratum of *q*.
- Basis: stratum 0.
- Then q(…) is inferred for *S* using naïve evaluation, with no negated IDB subgoals ever used.
- The same sequence of inferences, using instantiated rules, lets us infer q(…) as we compute *L*.

S = *L*, Continued

- Conversely, if q(…) is in *L*, then it is inferred using no negated IDB subgoals.
- Thus, the same sequence of inferences will be carried out using naïve evaluation for *S*.

Inductive Step

- Suppose *q* is at stratum *i*.
- Suppose q(…) is inferred for *S* at stratum *i*, using naïve evaluation with all IDB at stratum $< i$ treated as EDB.
- By inductive hypothesis, *L* and *S* agree below stratum *i*.
- Thus, same sequence of inferences, starting with what is known about *L*, infers q(…) for *L*; i.e. $S \subseteq L$.

Inductive Step: $L \subseteq S$

- Suppose q(…) is inferred for L, and q is at stratum i.
- Then the inference uses an instantiated rule, with any negated IDB subgoals having predicates at strata $< i$.
- By the inductive hypothesis, L and S agree on all those instantiated subgoals.

Inductive Step --- Concluded

- Thus, naïve evaluation at stratum *i* puts q(…) in *S*.
  - Requires another induction about predicates at stratum *i*.
- That is, $L \subseteq S$.
- Therefore $L = S$.

More Proofs

- We’re not going to prove that the locally stratified model, if it exists, is both the unique stable model and the well-founded model.

Comparison of Stable and Well-Founded

- If there is a 2-valued (no UNKNOWN's) well-founded model, then that is also the stable model.
- Yet, computing the well-founded model by alternating fixedpoint is polynomial in the size of the database.
- But it is NP-hard to tell whether a program has a unique stable model.