Datalog with negation

Adapted from slides by Jeff Ullman
Problem: the meaning of negation

Example 1 (propositional)

\[ p : - \) NOT \ q \]
\[ q : - \) NOT \ p \]

Several minimal models/fixpoints:

\{p\}, \{q\}

Bottom-up fixpoint evaluation: \{p,q\}
Example 2 (non-propositional)

EDB

\text{red}(X, Y) = \text{Red bus line runs from } X \text{ to } Y
\text{green}(X, Y) = \text{Green bus line runs from } X \text{ to } Y
IDB

greenPath(X, Y) = you can get from X to Y using only Green buses.

monopoly(X, Y) = Red has a bus from X to Y, but you can’t get there on Green, even changing buses.
Rules

greenPath(X,Y) :- green(X,Y)
greenPath(X,Y) :- greenPath(X,Z) & greenPath(Z,Y)
monopoly(X,Y) :- red(X,Y) & NOT greenPath(X,Y)
EDB Data

red(1,2), red(2,3), green(1,2)
Two Minimal Models

1. EDB + greenPath(1,2) + monopoly(2,3)
2. EDB + greenPath(1,2) + greenPath(2,3)
   + greenPath(1,3)

\[
\text{greenPath}(X,Y) :\text{green}(X,Y)
\]
\[
\text{greenPath}(X,Y) :\text{greenPath}(X,Z) \& \text{greenPath}(Z,Y)
\]
\[
\text{monopoly}(X,Y) :\text{red}(X,Y) \& \neg \text{greenPath}(X,Y)
\]
Syntactic restriction: safety

- all variables appear in some positive relational subgoal of the body.
Examples: Nonsafety

\[ p(X) :\neg q(Y) \]  
X is the problem

\[ \text{bachelor}(X) :\neg \text{married}(X,Y) \]  
Both X and Y are problems

\[ \text{bachelor}(X) : \text{person}(X) \land \neg \text{married}(X,Y) \]  
Y is still a problem
Semi-positive programs

- Simplest case: negate EDB predicates

\[ \text{Redonly}(X,Y) :\neg \text{red}(X,Y), \neg \text{green}(X,Y) \]

Natural meaning: \( \neg \text{green}(a,b) \) is true iff \((a,b)\) is not in \(\text{green}\)
(closed world assumption)
Next step: stratified negation

- Idea: if an IDB $R$ has already been defined, then we know what NOT $R$ is

- “Stratified program”: can be parsed so that each IDB relation is defined by previous rules before its negation is needed
Example: Monopoly

\[
greenPath(X,Y) :\neg green(X,Y) \\
greenPath(X,Y) :\neg greenPath(X,Z) \& \\
\quad greenPath(Z,Y) \\
monopoly(X,Y) :\neg red(X,Y) \& \\
\quad \neg greenPath(X,Y)
\]
Stratified programs

1. *Dependency graph* describes how IDB predicates depend negatively on each other.

2. *Stratified Datalog* = no recursion involving negation.

3. *Stratified model* is a particular model that “makes sense” for stratified Datalog programs.
Dependency Graph

- Nodes = IDB predicates.
- Arc $p \rightarrow q$ iff there is a rule for $p$ that has a subgoal with predicate $q$.
- Arc $p \rightarrow q$ labeled – iff there is a subgoal with predicate $q$ that is negated.
Monopoly Example

- monopoly
- greenPath

\[ \text{--} \]
Another Example: “Win”

\[
\text{win}(X) : - \text{move}(X,Y) \; \& \; \text{NOT} \; \text{win}(Y)
\]

- Represents games like Nim where you win by forcing your opponent to a position where they have no move.
Dependency Graph for “Win”
Strata

- The stratum of an IDB predicate is the largest number of –’s on a path from that predicate, in the dependency graph.

Examples:

- Stratum = ∞
- Stratum = 0

- Stratum = 1
Stratified Programs

- If all IDB predicates have finite strata, then the Datalog program is *stratified*.
- If any IDB predicate has the infinite stratum, then the program is *unstratified*, and no stratified model exists.
Stratified Model

- Evaluate strata 0, 1, … in order.
- If the program is stratified, then any negated IDB subgoal has already been evaluated.
  - Safety assures that we can “subtract it from something.”
  - Treat it as EDB.
- Result is the stratified model.
Examples

- For “Monopoly,” `greenPath` is in stratum 0: compute it (the transitive closure of `green`).
- Then, `monopoly` is in stratum 1: compute it by taking the difference of `red` and `greenPath`.
- Result is first model proposed.
- “Win” is not stratified, thus no stratified model.
Technical points

- The stratified model is one of the minimal models for the program.

- If there is no negation, the stratified model is the minimum model.

nice agreement with Datalog semantics!
What About Unstratified Datalog?

- There are extensions of the stratified semantics (more or less convincing).

well-founded semantics
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 false EDB subgoal.

- An atom with no variables is a *ground atom*.
  - Like propositions in propositional calculus.
Example: Ground Atoms

Consider the Win program:

\[
\text{win}(X) :- \, \text{move}(X,Y) \, \& \, \text{NOT win}(Y)
\]

with the following moves:

\[
\begin{align*}
\text{win}(1) & :- \, \text{move}(1,2) \, \& \, \text{NOT win}(2) \\
\text{win}(1) & :- \, \text{move}(1,3) \, \& \, \text{NOT win}(3) \\
\text{win}(2) & :- \, \text{move}(2,3) \, \& \, \text{NOT win}(3)
\end{align*}
\]
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.
Well-founded semantics

- provides model-based semantics for all programs!
- has corresponding fixpoint semantics ("alternating fixpoint")
- BUT: gives up settling all IDB facts; some facts remain unknown
Example: well founded semantics of

\[ p :- p \]

is that \( p \) is unknown

Intuition: program gives no information on \( p \)
3-Valued Models

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”
3-Stable Models

Intuition: M is a 3-stable model if
--its positive IDB facts follow from its negative ones
--all negative IDB facts that can be inferred from M are already in M
- Inferring positive facts:
  body of rule must be true

- Inferring negative facts:
  bodies of rules with fact in head are false (contain at least one false fact) or no body has fact in head
In general, programs may have several 3-stable models.

Well-founded semantics: take the “certain” positive and negative facts from all 3-stable models; other facts remain unknown.
Example

\[\text{win}(X) : - \text{move}(X, Y) \land \neg \text{win}(Y)\]

with these moves:

Well-founded model:

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

\text{win}(1), \text{win}(2) \text{ remain unknown}

Note: 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
Alternating fixpoint semantics

- Idea: alternate underestimates and overestimates of positive facts until convergence
1. Underestimate of positive facts: $\emptyset$
1. Underestimate of positive facts: \( \emptyset \)
2. Overestimate of negative facts: everything
3. Infer overestimate of positive facts
4. Obtain underestimate of negative facts
4. Infer new underestimate of positive facts (larger than before)
4. Obtain new overestimate of negative facts (smaller than before)
4. Infer new overestimate of positive facts (smaller than before)
4. Obtain new underestimate of negative facts (larger than before)
4. Infer new underestimate of positive facts (larger than before)
We get increasing sequences of positive and negative facts.
Limit: well-founded model
Previous Win Example

\[
\begin{align*}
\text{win}(1) & : - \text{NOT} \quad \text{win}(2) \\
\text{win}(2) & : - \text{NOT} \quad \text{win}(1) \\
\text{win}(2) & : - \text{NOT} \quad \text{win}(3) \\
\text{win}(3) & : - \text{NOT} \quad \text{win}(4) \\
\text{win}(4) & : - \text{NOT} \quad \text{win}(5) \\
\text{win}(5) & : - \text{NOT} \quad \text{win}(6)
\end{align*}
\]
Computing the AFP

<table>
<thead>
<tr>
<th>Round</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>win(1)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>win(2)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>win(3)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>win(4)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>win(5)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>win(6)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Even round: underestimate of positive facts
Odd round: overestimate of positive facts
Same result as earlier:

\[
\text{win}(X) :- \text{move}(X,Y) \& \text{NOT win}(Y)
\]

with these moves:

Well-founded model:

\[
\{\text{win}(3), \text{win}(5), \text{NOT win}(4), \text{NOT win}(6)\}
\]

\[
\text{win}(1), \text{win}(2) \text{ \text{remain unknown}}
\]
Another Example

\[
\begin{align*}
p & :- q \\
r & :- p \land q \\
q & :- p \\
s & :- \neg p \land \neg q
\end{align*}
\]

<table>
<thead>
<tr>
<th>Round</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Well-founded model:
\{s, \neg p, \neg q, \neg r\}
Yet Another Example

\[ p :- q \quad q :- \text{NOT} \; p \]

<table>
<thead>
<tr>
<th>Round</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>q</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Well-founded model: \(\emptyset\)  (everything is unknown)
Comparison of Semantics

- Well-founded
  - Stratified
    - Datalog (no negation)
Drawbacks of “non-monotonic reasoning” approach to negation:

- Beyond Datalog, no coincidence of least fixpoint, minimum model, and proof-theoretic semantics
- Beyond stratified Datalog, semantics are somewhat arbitrary – “guessing what the programmer has in mind” is risky.
Alternative: bottom-up inflationary fixpoint evaluation

- Semantics: fire rules up to a fixpoint
- At each firing, assume facts not yet inferred to be negative
- Computational approach: no consistency between negative and positive facts
  
  NOT A can be used to infer A
Propositional example

\[ p :\neg q \]
\[ q :\neg p \]

Inflationary fixpoint semantics: \{p,q\}
More interesting example

Closer(x, y, x', y')\colon d(x,y) < d(x',y') \text{ in } G

T(x,y) \colon\! -\! G(x,y)
T(x,y) \colon\! -\! T(x,z), \ G(z,y)
Closer(x, y, x', y') \colon\! -\! T(x,y), \ \text{NOT} \ T(x',y')
Surprising fact: Inflationary semantics has the same expressiveness as Well-founded semantics (reduced to 2-valued semantics)

Example: complement of transitive closure

Note: this doesn’t work with inflationary semantics

\[
\begin{align*}
T(x,y) & :\ - G(x,y) \\
T(x,y) & :\ - T(x,z), \ G(z,y) \\
CT(x,y) & :\ - NOT \ T(x,y)
\end{align*}
\]

Need to “delay” computation of CT until after computation of T is fully completed.
This can be done as follows:

\[
\begin{align*}
T(x,y) & : - G(x,y) \\
T(x,y) & : - T(x,z), G(z,y) \\
old-T(x,y) & : - T(x,y) \\
old-T\text{-except-final}(x,y) & : - T(x,y), T(x',z'), T(z',y'), \text{ NOT } T(x',y') \\
CT(x,y) & : - \text{ NOT } T(x,y), \text{ old-T}(x',y'), \text{ NOT } \text{ old-T\text{-except-final}}(x',y')
\end{align*}
\]

Theorem: Inflationary and well-founded Datalog (coerced to 2-valued answers) are equivalent.
Datalog expressive power summary

Well-founded $\equiv$ Inflationary

Stratified

Datalog (no negation)