Logic Programming Negation

CSE 595 – Semantic Web
Instructor: Dr. Paul Fodor
Stony Brook University

http://www3.cs.stonybrook.edu/~pfodor/courses/cse595.html
Negation in Logic Programs

\begin{verbatim}
above(X, Y) :- on(X, Y).
above(X, Y) :- on(X, Z), above(Z, Y).

on(c, b).
on(b, a).

?- above(c, a).
  • Yes, since \texttt{above(c, a)} is in the least Herbrand model of the program.

?- above(b, c).
  • There are models which contain \texttt{above(b, c)}, but it is not in the least Herbrand model of the program.
  • Not a logical consequence of the program.

?- \neg above(b, c).
  • Yes, since \texttt{above(b, c)} is not a logical consequence of the program.
\end{verbatim}
Closed World Assumption

“... the truth, the whole truth, and nothing but the truth ...”

- the truth: anything that is the logical consequence of the program is true.
- “the whole truth, and nothing but the truth”: anything that is not a logical consequence of the program is false.
- Closed World Assumption (CWA):

\[
\frac{P \not\models A}{\neg A} \quad \frac{P \models A}{\neg A}
\]

- Negation as (finite) failure:

\[
\leftarrow A \text{ has a finitely failed SLD tree} \quad \neg A
\]
Finite Failure

- Every SLD derivation that fails in a finite number of resolution steps

```
:- above(b, c)

:- on(b, c)      :- on(b, Z0), above(Z0, c)
    fail

    Z0 = a

    :- above(a, c)

    :- on(a, c)      :- on(a, Z1), above(Z1, c)
    fail      fail
```
A problem with CWA

above(X, Y) :- on(X, Y).
above(X, Y) :- on(X, Z), above(Z, Y).
on(c, b).
on(b, a).

?- ¬above(b, c).

above(b, c) is not a logical consequence of the program so
¬above(b, c) must be true.

- But ¬above(b, c) is not a logical consequence of the program either!
  - (There are models with ¬above(b, c))
- Must strengthen what we mean by a program (NORMAL INTUITION.)
Completion

\[ \text{above}(X, Y) :\neg \text{on}(X, Y). \]
\[ \text{above}(X, Y) :\neg \text{on}(X, Z), \text{above}(Z, Y). \]

- Logical meaning of the program:

\[ \text{above}(X, Y) \iff \text{on}(X, Y) \lor (\text{on}(X, Z) \land \text{above}(Z, Y)) \]

- \text{above}(X,Y) cannot be true in any other way (by CWA)!
- Hence the above program is equivalent to:

\[ \text{above}(X, Y) \iff \text{on}(X, Y) \lor (\text{on}(X, Z) \land \text{above}(Z, Y)) \]

Called the “completion” (also "Clark’s completion") of the program.
How to complete a program

1. Rewrite each rule of the form
   \[ p(t_1, \ldots, t_m) \leftarrow L_1, \ldots, L_n. \]
   to
   \[ p(x_1, \ldots, x_m) \leftarrow x_1=t_1, \ldots, x_m=t_m, L_1, \ldots, L_n. \]

2. For each predicate symbol \( p \) which is defined by rules:
   \[ p(x_1, \ldots, x_m) \leftarrow B_1. \]
   \[ \ldots \]
   \[ p(x_1, \ldots, x_m) \leftarrow B_n. \]
   replace the rules by:
   - If \( n > 0 \):
     \[ \forall x_1, \ldots, x_m \ p(x_1, \ldots, x_m) \leftrightarrow B_1 \lor B_2 \lor B_3 \lor \ldots \lor B_n. \]
   - If \( n = 0 \):
     \[ \forall x_1, \ldots, x_m \ \neg p(x_1, \ldots, x_m). \]
Negation in Logic Programs

- The negation-as-failure 'not' predicate could be defined in Prolog as follows:

  \[
  \text{not}(P) :- \text{call}(P), !, \text{fail}.
  \]

  \[
  \text{not}(P).
  \]

- Quintus, SWI, and many other prologs use '\+1' rather than 'not'.

- Another way one can write the 'not' definition is using the Prolog implication operator '\rightarrow' (if-then-else):

  \[
  \text{not}(P) :- (\text{call}(P) \rightarrow \text{fail} ; \text{true})
  \]
Negation in Logic Programs

\[
bachelor(P) :- \text{male}(P), \text{not(married}(P)).
\]

\[
\text{male}(henry).
\]

\[
\text{male}(tom).
\]

\[
\text{married}(tom).
\]

?- bachelor(henry).

\[
yes
\]

?- bachelor(tom).

\[
\text{no}
\]

?- bachelor(Who).

\[
\text{Who= henry ;}
\]

\[
\text{no}
\]

?- \text{not(married}(Who)).

\[
\text{no}.
\]

This might not be intuitive!

\[
\text{not(married}(Who)) \text{ fails because for the variable binding } Who=\text{tom, married}(Who) \text{ succeeds, and so the negative goal fails.}
\]
Negation in Logic Programs

\[
p(X) : - q(X), \text{not}(r(X)).
\]
\[
r(X) : - w(X), \text{not}(s(X)).
\]
\[
q(a).
\]
\[
q(b).
\]
\[
q(c).
\]
\[
s(a) : - p(a).
\]
\[
s(c).
\]
\[
w(a).
\]
\[
w(b).
\]
\[
?-p(a).
\]
Negation in Logic Programs

\[ u(X) :- \text{not}(s(X)). \]
\[ s(X) :- s(f(X)). \]
\[ \text{?-} u(1). \]