Logic Programming Negation

CSE 505 – Computing with Logic
Stony Brook University

http://www.cs.stonybrook.edu/~cse505
Negation in Logic Programs

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

?- \text{above}(c, a).

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

?- \text{above}(b, c).

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

?- \neg \text{above}(b, c).

- Yes, since \text{above}(b, c) is not a logical consequence of the program.
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):

\[
\frac{P \nvdash A}{\neg A} \quad \frac{P \nvdash A}{\neg A}
\]

- 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

\[
\begin{align*}
\text{:- } & \text{above}(b, c) \\
\text{:- } & \text{on}(b, c) \\
\text{fail} & \\
\text{:- } & \text{on}(b, Z0), \text{above}(Z0, c) \\
Z0 & = a \\
\text{:- } & \text{above}(a, c) \\
\text{:- } & \text{on}(a, c) \\
\text{fail} & \\
\text{:- } & \text{on}(a, Z1), \text{above}(Z1, c) \\
\text{fail} & 
\end{align*}
\]
A problem with CWA

\[
\text{above}(X, Y) :\text{-} \text{on}(X, Y).
\]
\[
\text{above}(X, Y) :\text{-} \text{on}(X, Z), \text{above}(Z, Y).
\]
\[
\text{on}(c, b).
\]
\[
\text{on}(b, a).
\]

?- \neg \text{above}(b, c).

\text{above}(b, c) \text{ is not a logical consequence of the program so } \neg \text{above}(b, c) \text{ must be true.}

\bullet \text{ But } \neg \text{above}(b, c) \text{ is not a logical consequence of the program either!}

\bullet \text{ (There are models with } \neg \text{above}(b, c))

\bullet \text{ 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:

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

- \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 "\text{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:

   \[ p(x_1, \ldots, x_m) \leftarrow B_1. \]

   ... 

   \[ p(x_1, \ldots, x_m) \leftarrow B_n. \]

   replace the rules by:

   • If \( n > 0 \):

     \[ \forall x_1, \ldots, x_m \ p(x_1, \ldots, x_m) \iff B_1 \lor B_2 \lor B_3 \lor \ldots \lor B_n. \]

   • 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:

  ```prolog
  not(P) :- call(P), !, fail.
  not(P).
  ```

- Quintus, SWI, and many other prologs use '⊥' rather than 'not'.

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

  ```prolog
  not(P) :- (call(P) -> fail ; true)
  ```
Negation in Logic Programs

\[ \text{bachelor}(P) : \neg \text{male}(P), \neg \text{married}(P). \]
\[ \text{male}(\text{henry}). \]
\[ \text{male}(\text{tom}). \]
\[ \text{married}(\text{tom}). \]
\[ ?- \text{bachelor}(\text{henry}). \]
\[ \text{yes} \]
\[ ?- \text{bachelor}(\text{tom}). \]
\[ \text{no} \]
\[ ?- \text{bachelor}(\text{Who}). \]
\[ \text{Who} = \text{henry}; \]
\[ \text{no} \]
\[ ?- \neg \text{married}(\text{Who}). \]
\[ \text{no}. \]

This might not be intuitive!

\[ \neg \text{married}(\text{Who}) \] fails because for the variable binding \( \text{Who} = \text{tom}, \text{married}(\text{Who}) \) succeeds, and so the negative goal fails.
Negation in Logic Programs

\[ p(X) :- q(X), \neg (r(X)). \]
\[ r(X) :- w(X), \neg (s(X)). \]

\[ q(a). \]
\[ q(b). \]
\[ q(c). \]
\[ s(a) :- p(a). \]
\[ s(c). \]
\[ w(a). \]
\[ w(b). \]
\[ ?- p(a). \]
Negation in Logic Programs

\[ u(X) :- \neg s(X). \]
\[ s(X) :- s(f(X)). \]
\[ ?- u(1). \]