Negation in Logic Programs

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

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

\[
\text{on}(c, b).
\]

\[
\text{on}(b, a).
\]

• \(-\) \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):

\[
P \not\models A \quad \Rightarrow \quad \neg A
\]

\[
P \models A \quad \Rightarrow \quad \neg A
\]

- Negation as (finite) failure:

\[
\neg A \quad \iff \quad A \text{ has a finitely failed SLD tree}
\]

(c) Paul Fodor (CS Stony Brook) and Elsevier
Finite Failure

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

```
:- 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

\begin{align*}
\text{above}(X,Y) & : - \text{on}(X,Y). \\
\text{above}(X,Y) & : - \text{on}(X,Z), \text{above}(Z,Y).
\end{align*}

\begin{align*}
\text{on}(c,b).
\text{on}(b,a).
\end{align*}

\begin{itemize}
\item \texttt{?- \neg above(b,c)}.
\end{itemize}

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

\begin{itemize}
\item But \texttt{\neg above(b,c)} is not a logical consequence of the program either!
  \begin{itemize}
  \item (There are models with \texttt{\neg above(b,c)})
  \end{itemize}
\item Must strengthen what we mean by a program (NORMAL INTUITION.)
\end{itemize}
Completion

above(X, Y) :- on(X, Y).
above(X, Y) :- on(X, Z), above(Z, Y).

• Logical meaning of the program:

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

• 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:
   \[ p(X_1, \ldots, X_m) \leftarrow B_1. \]
   ...
   \[ p(X_1, \ldots, X_m) \leftarrow B_n. \]

   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:

  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 '->' :

  not(P) :- (call(P) -> fail ; true)
bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
mari**ed**(tom).
?- bachelor(henry).
yes
?- bachelor(tom).
no
?- bachelor(Who).
Who = henry ;
no
?- not(married(Who)).
no.

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

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

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