Logic Programming Negation

CSE 505 – Computing with Logic
Stony Brook University

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

• In real life the negative information is seldom stated explicitly
  • Example: the train table states there is a daily train from Stony Brook to New York at 9:10am, but **it does not explicitly state that there is no train departing at 9:11am or 9:12am** or …

• Thus, in many real-life situations the **lack of information is taken as evidence to the contrary**
  • Example: since the timetable does not indicate a departure from Stony Brook to New York at 9:14am, one does not plan to take such a train
    • This is because we assume that timetable lists all trains from Stony Brook to New York

• This idea is the intuition behind the so-called closed world assumption
  • The **closed world assumption** is a mechanism that allows us to draw **negative conclusions based on the lack of positive information**
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

?– \text{not 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

- Semantics: 

  \[
  \frac{P \not
  \vdash A}{\neg A}
  \]

  Closed World Assumption (CWA):

  \[
  \frac{P \vdash A}{\neg A}
  \]

  **Negation as (finite) failure:**

  \[
  \leftarrow A \text{ has a finitely failed SLD tree}
  \]

  \[
  \neg A
  \]
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
```

(c) Paul Fodor (CS Stony Brook) and Elsevier
A problem with CWA

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

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

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

- But \textbf{\(\neg \text{above}(b, c)\)} is not a logical consequence of the program
  - Because there are models with \textbf{above}(b, c)
- So we must strengthen what we mean by a program
Completion

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

• Logical meaning of the program:

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

But we want that \textbf{above}(X, Y) cannot be true in any other way (by CWA)!

• Hence the above program is equivalent to:

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

Called the “\textit{completion}” (also "\textit{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 \( n > 0 \):

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

   • 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:

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

• Quintus, SWI, and many other prologs use '\(+\)' or 'naf' (for negation as failure) in the syntax rather than 'not'.

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

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

bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
mari**

了解一下

This might not be intuitive!

?- bachelor(henry).
yes
?- bachelor(tom).
no
?- bachelor(Who).
Who= henry ;
no
?- not(married(Who)).
no.

This might not be intuitive!
Negation in Logic Programs

\[
\begin{align*}
u(X) & : \neg s(X). \\
s(X) & : s(f(X)).
\end{align*}
\]

\[?-u(1).\]
Negation in Logic Programs

\[ p(X) \quad \text{:-} \quad q(X), \neg r(X). \]
\[ r(X) \quad \text{:-} \quad w(X), \neg s(X). \]
\[ q(a). \]
\[ q(b). \]
\[ q(c). \]
\[ s(a) \quad \text{:-} \quad p(a). \]
\[ s(c). \]
\[ w(a). \]
\[ w(b). \]

\[ \neg \quad p(a). \]

\[ \infty \]