Logic Programming Negation

CSE 505 – Computing with Logic

Stony Brook University

http://www.cs.stonybrook.edu/~cse505

Negation in Logic Programs

```
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 above(c,a) is in the least Herbrand model of the program.
- ?- above(b,c).
  - There are models which contain above(b, c), but it is not in the least Herbrand model of the program.
  - Not a logical consequence of the program.
- ?- ¬ above(b,c).
  - Yes, since 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):

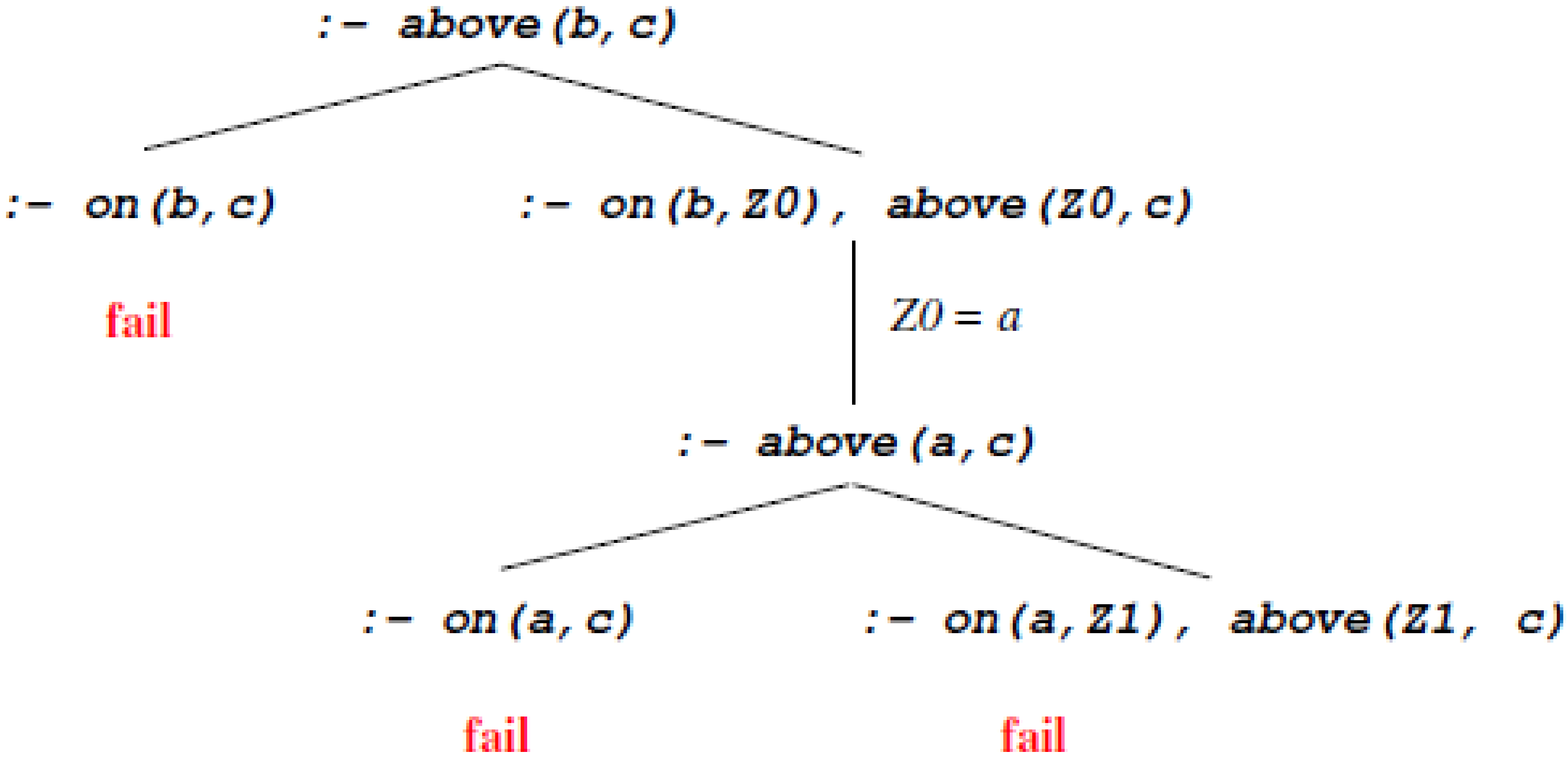
$$\frac{P \not\models A}{\neg A} \qquad\qquad \frac{P \not\vdash A}{\neg A}$$

- Negation as (finite) failure:

$$\frac{\leftarrow A \text{ has a finitely failed SLD tree}}{\neg A}$$

Finite Failure

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



:- above(b, c)

:- on(b, c)

fail

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

$Z0 = a$

:- above(a, c)

:- on(a, c)

fail

:- on(a, Z1), above(Z1, c)

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

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

- Logical meaning of the program:

  above(X,Y) ← on(X,Y) ∨ (on(X,Z) ∧ above(Z,Y))

- above(X,Y) cannot be true in any other way (by CWA).
- Hence the above program is equivalent to:

  above(X,Y) ↔ on(X,Y) ∨ (on(X,Z) ∧ 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

$p(t1, \ldots, tm) \leftarrow L1, \ldots, Ln.$

to

$p(X1, \ldots, Xm) \leftarrow X1 = t1, \ldots, Xm = tm, L1, \ldots, Ln.$

2. For each predicate symbol p which is defined by rules:

$p(X1, \ldots, Xm) \leftarrow B1.$

…

$p(X1, \ldots, Xm) \leftarrow Bn.$

replace the rules by:

- If $n > 0$:

  $\forall X1, \ldots, Xm.\ p(X1, \ldots, Xm) \leftrightarrow B1 \vee B2 \vee B3 \vee \ldots \vee Bn.$

- If $n = 0$:

  $\forall X1, \ldots, Xm.\ \neg p(X1, \ldots, Xm)$

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)
```

Negation in Logic Programs

```
bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
married(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) :- q(X), not(r(X)).
r(X) :- w(x), 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) :- not(s(X)).
s(X) :- s(f(X)).
?-u(1).
```