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

```
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**

`?- not 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**
- Semantics:

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

Closed World Assumption (CWA):

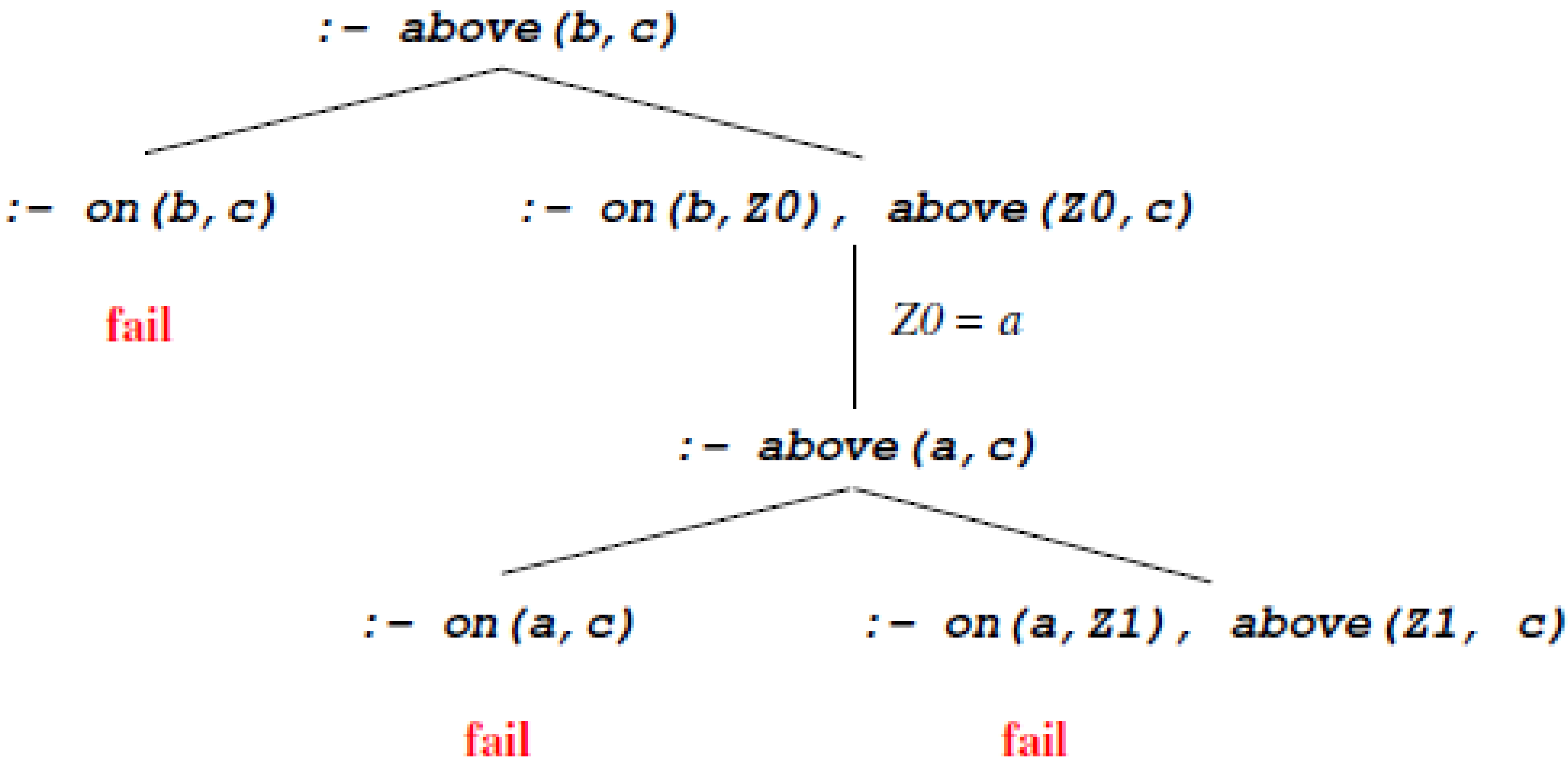
$$\frac{P \not\vdash A}{\neg A}$$

Negation as (finite) failure:

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

Finite Failure

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



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).
```

```
?- not 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
  - Because there are models with **`above(b,c)`**
- So we must strengthen what we mean by a program

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) ).
```

But we want that **`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

$$\mathtt{p(t_1,\ \ldots,\ t_m)} \leftarrow \mathtt{L_1,\ \ldots,L_n.}$$

to

$$\mathtt{p(X_1,\ \ldots,X_m)} \leftarrow \mathtt{X_1=t_1,\ldots,X_m=t_m,\ L_1,\ldots,L_n.}$$

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

$$\mathtt{p(X_1,\ \ldots,X_m)} \leftarrow \mathtt{B_1.}$$

...

$$\mathtt{p(X_1,\ \ldots,X_m)} \leftarrow \mathtt{B_n.}$$

replace the rules by:

- If $\mathtt{n > 0}$:

$$\forall \mathtt{X_1,\ \ldots,X_m}\ \mathtt{p(X_1,\ldots,X_m)} \leftrightarrow \mathtt{B_1} \lor \mathtt{B_2} \lor \mathtt{B_3} \lor \ldots \lor \mathtt{B_n.}$$

- If $\mathtt{n = 0}$:

$$\forall \mathtt{X_1,\ \ldots,X_m}\ \neg \mathtt{p(X_1,\ \ldots,X_m)}$$

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 '**\+**' 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):

```
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.
```

This might not be intuitive!

`?- not(married(Who)).` fails because for the variable binding **`Who=tom`**, **`married(Who)`** succeeds, and so the negative goal fails.

Negation in Logic Programs

```
u(X) :- not(s(X)).
s(X) :- s(f(X)).
```

```
?-u(1).
```

∞

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).
```

∞