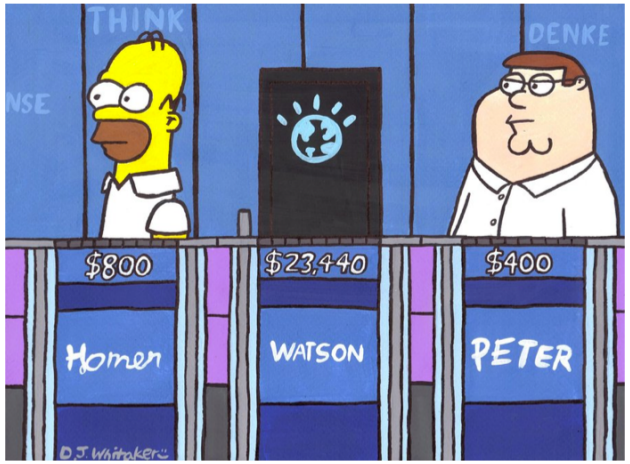
Prolog and declarative programming, pt 4



The cut operator

- Backtracking search is pretty awesome, but right now we only have two ways to control it:
  - Reorder the rules
  - Reorder the goals

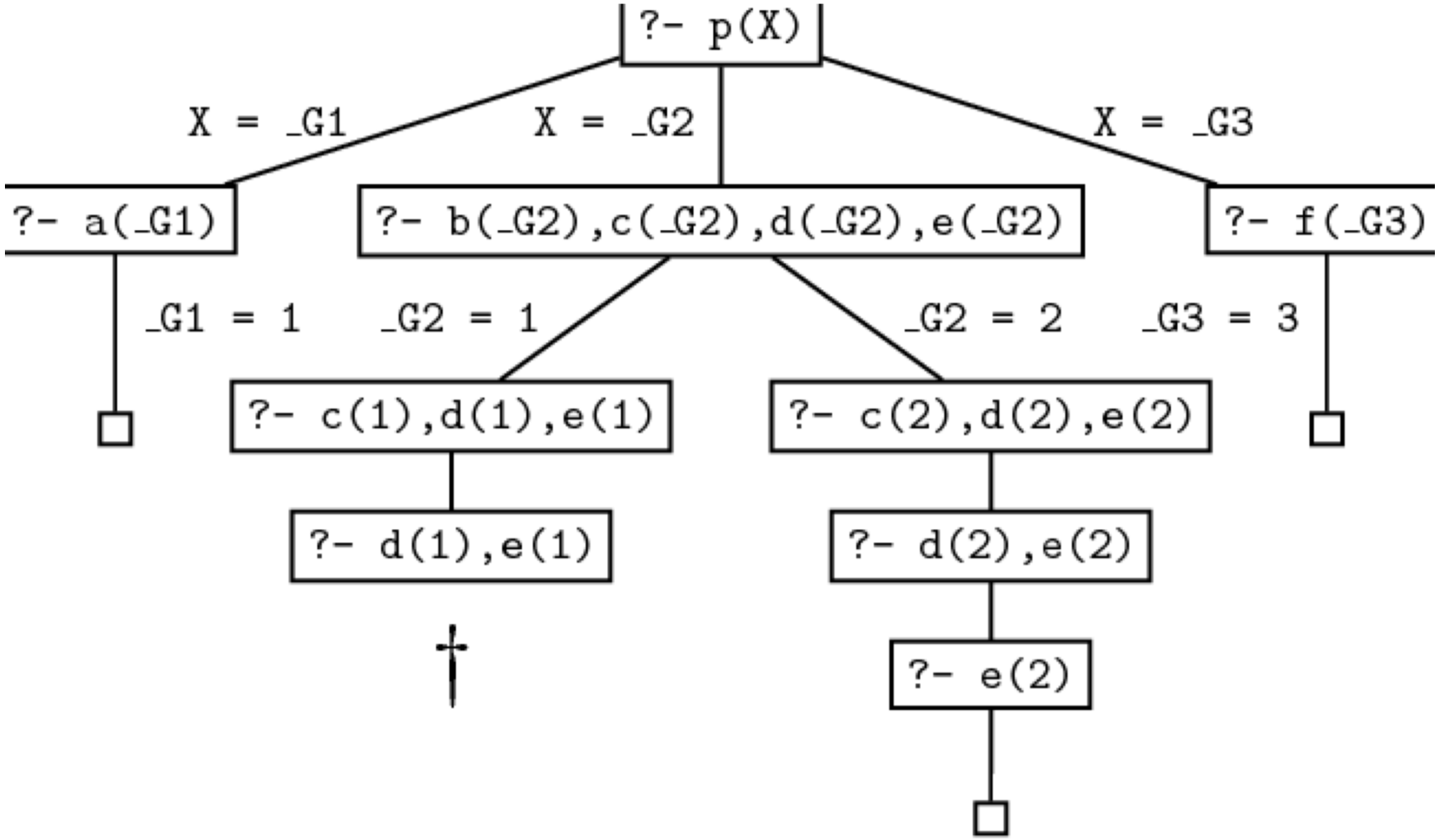
- The *cut* operator allows us to control the way the search unfolds

The cut operator

```
p(X):-  a(X).
p(X):-  b(X),  c(X),  d(X),  e(X).
p(X):-  f(X).
a(1). b(1). c(1). d(2). e(2). f(3). b(2). c(2).

?- p(X)
X  =  1  ;
X  =  2  ;
X  =  3  ;
no
```

The cut operator



The cut operator

- Suppose we change the second rule, and then issue the same query.

```
...
p(X):-  b(X),  c(X),  !,  d(X),  e(X).
...

?- p(X)
X  =  1  ;
no
```

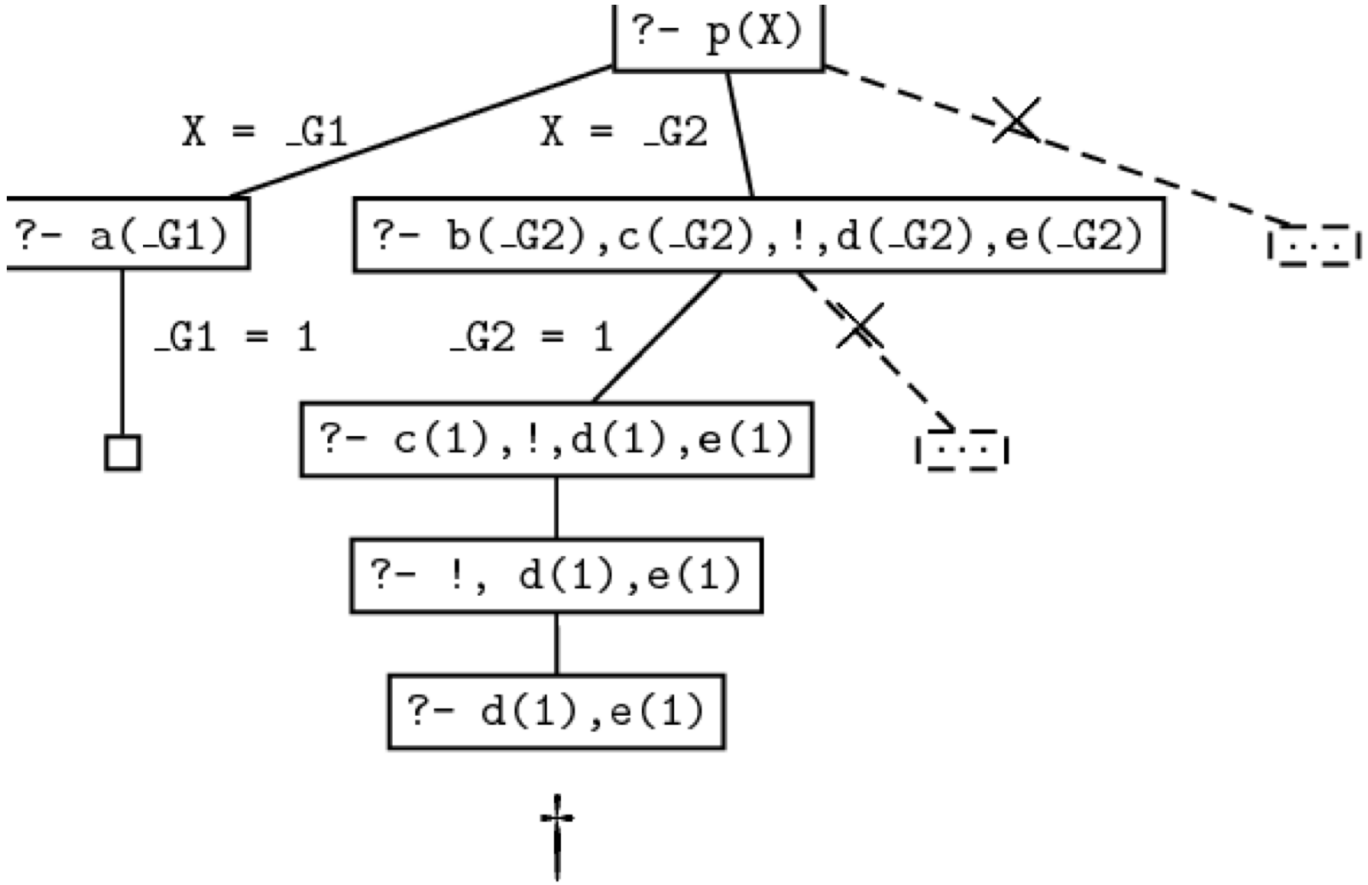
- What happened???

The cut operator

- Cut is a goal that always succeeds.
- Second, and more importantly, it has a side effect.
- Suppose that some goal makes use of this clause (we call this goal the parent goal).
- Then the cut commits Prolog to any choices that were made since the parent goal was unified with the left hand side of the rule (including, importantly, the choice of using that particular clause).

The cut operator

```
p(X):-  a(X).
p(X):-  b(X),  c(X), !, d(X),  e(X).
p(X):-  f(X).
a(1). b(1). c(1). d(2). e(2). f(3). b(2). c(2).
```



- p(X) is first unified with the first rule, so we get a new goal a(X) . By instantiating X to 1 , Prolog unifies a(X) with the fact a(1) and we have found a solution. So far, this is exactly what happened in the first version of the program.
- We then go on and look for a second solution. p(X) is unified with the second rule, so we get the new goals b(X),c(X),!,d(X),e(X) . By instantiating X to 1 , Prolog unifies b(X)with the fact b(1) , so we now have the goals c(1),!,d(1),e(1) . But c(1) is in the database so this simplifies to !,d(1),e(1) .
- Now for the big change. The ! goal succeeds (as it always does) and commits us to the choices made so far. In particular, we are committed to having X = 1 , and we are also committed to using the second rule.
- But d(1) fails. And there's no way we can re-satisfy the goal p(X) . Sure, if we were allowed to try the value X=2 we could use the second rule to generate a solution (that's what happened in the original version of the program). But we can't do this: the cut has removed this possibility from the search tree. And sure, if we were allowed to try the third rule, we could generate the solution X=3 . But we can't do this: once again, the cut has removed this possibility from the search tree.

An example – the max function

```
?-  max(2,3,3).
?-  max(3,2,3).
?-  max(3,3,3).
```

Should succeed

```
?-  max(2,3,2).
?-  max(2,3,5).
```

Should fail

Write the max predicate!

```
?-  max(2,3,Max).
Max  =  3
yes

?-  max(2,1,Max).
Max  =  2
yes
```

The max function

```
max(X,Y,Y)  :-  X  =<  Y.
max(X,Y,X)  :-  X>Y.
```

```
max(X,Y,Y)  :-  X  =<  Y,!.
max(X,Y,X)  :-  X>Y.
```

A *green cut*

The max function – can we do better?

```
max(X,Y,Y)  :-  X  =<  Y,!.
max(X,Y,X).

?-  max(100,101,X).
X  =  101
yes

?-  max(3,2,X).
X  =  3
yes


?-  max(2,3,2).
```

```
max(X,Y,Z)  :-  X  =<  Y,!,  Y  =  Z.
max(X,Y,X).
```

A *red cut*

Cut-fail combination: negation as failure

```
enjoys(vincent,X)  :-  big_kahuna_burger(X),!,fail.
enjoys(vincent,X)  :-  burger(X).

burger(X)  :-  big_mac(X).
burger(X)  :-  big_kahuna_burger(X).
burger(X)  :-  whopper(X).

big_mac(a).
big_kahuna_burger(b).
big_mac(c).
whopper(d).
```

```
?-  enjoys(vincent,a).
yes

?-  enjoys(vincent,b).
no

?-  enjoys(vincent,c).
yes

?-  enjoys(vincent,d).
yes
```

```
enjoys(vincent,X)  :-  burger(X), \+  big_kahuna_burger(X).
```

Negation as failure: not logical negation!

These two statements are not equivalent!

```
enjoys(vincent,X)  :-  burger(X), \+  big_kahuna_burger(X).
```

```
enjoys(vincent,X)  :-  \+  big_kahuna_burger(X),  burger(X).
```

```
?-  enjoys(vincent,X).
no
```

var/1 and nonvar/1

```
?- var(X).
yes

?- var(mia).
no

?- var(8).
no

?- var(3.25).
no

?- var(loves(vincent,mia)).
no
```

```
?- X = a, var(X).
no

?- X = a, nonvar(X).
X = a
yes

?- var(X), X = a.
X = a
yes

?- nonvar(X), X = a.
no
```