Concepts of programming languages

Prolog

Winand, Roald, Sjoerd, Alexey and Anvar

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Cut 1

Take this simple knowledge base:

```
bar(a).
bar(b).
baz(a).
baz(b).

foo(X, Y) :- bar(X), baz(Y).
```

which translates to the tree:

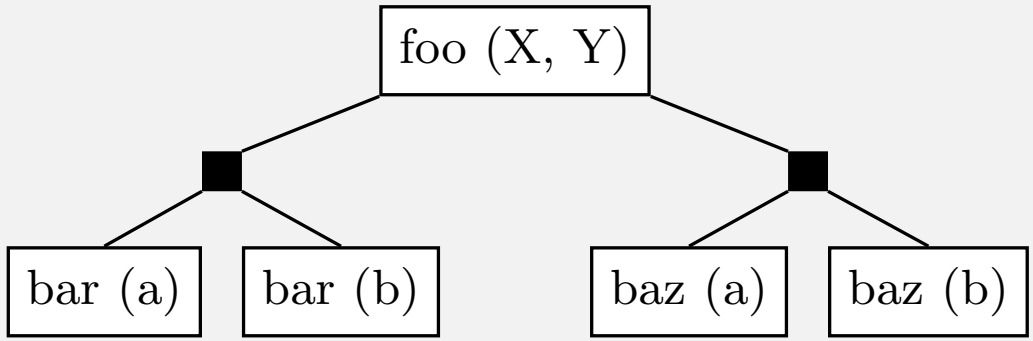

Figure 1: Branches of foo(X,Y)

Cut 2

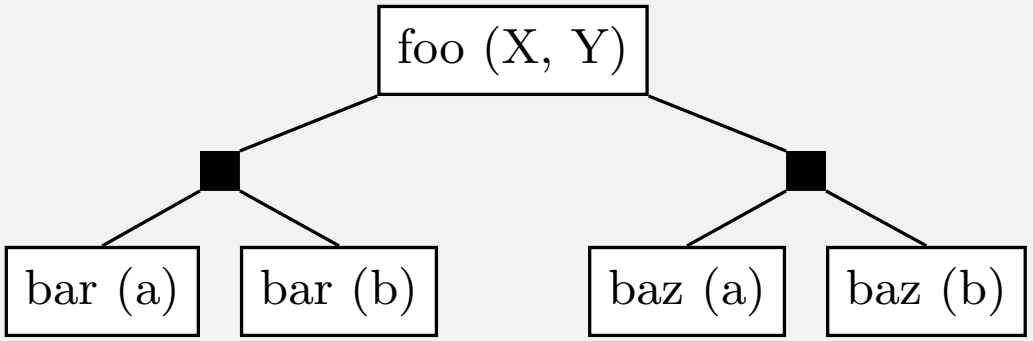

Figure 2: Branches of foo(X, Y)

Querying `prolog foo(X, Y)` will give the results:

```
(X, Y) = (a, a); (a, b); (b, a); (b, b).
```

But what if you would only like to get the first n results?

Cut 3

A *cut* is an atom that can be used to influence the backtracking in Prolog. It tells Prolog to ignore backtracking up to the point it encounters a cut:

```
foo(X, Y) :- bar(X), !, baz(Y).
```

The query `foo(X, Y).` now gives the results:

```
(X, Y) = (a, a); (a, b).
```

Cut 4

Lets see what actually happens:

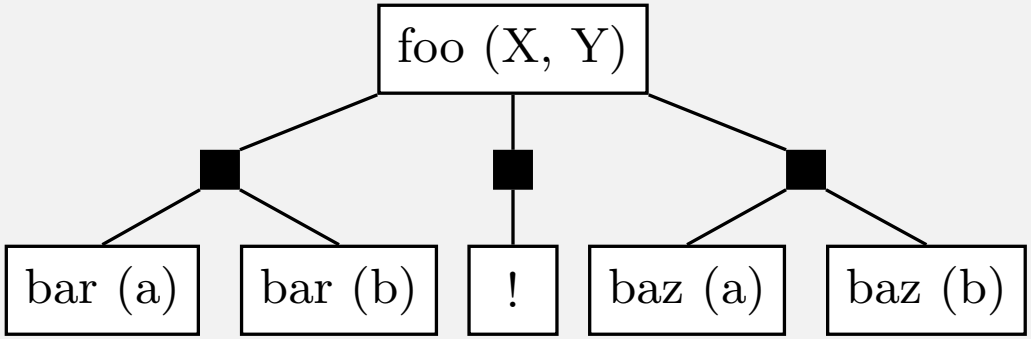

Figure 3: Branches of foo(X, Y) with cut

Cut 5

Moving the cut from left to right,

```
foo(X, Y) :- 1, bar(X), 2, baz(Y), 3.
```

will yield the results:

```
0: (X, Y) = (a, a); (a, b); (b, a); (b, b).
1: (X, Y) = (a, a); (a, b); (b, a); (b, b).
2: (X, Y) = (a, a); (a, b).
3: (X, Y) = (a, a).
```

These results show how cut actually works!

Cut 6

There are two *types* of cuts: green and red cuts.

Cuts are considered green in case they only make a program more efficient, without changing its output:

```
drunk(X) :- beer(X), !.
drunk(X) :- money(X), \+ beer(X).
```

Without the cut, it will do backtracking. This example will keep functioning when the first line is removed.

Cut 7

When cuts are not green they are red, hence they influence the output of a program:

```
drunk(X) :- beer(X), !.
drunk(X) :- money(X).
```

In this case the output will be different when the first line is removed. Without the cut this example will yield two results, while the previous example will yield only one.

In the second example you aren't checking whether you still have beer… (or you are really really drunk).

Cut 8

Some advices about the usage of *cut*:

- It can make a program much more efficient and prevent it from going down branches that are not interesting.
- It commits the goal to being proven, hence is must be used when the alternative must not be tried.
- There are alternatives for the usage of cuts, like *once*.
- Using cut correct requires an intrinsic understanding of logic and Prolog, so when there is a way around it, try to avoid the use it will make life a much more pleasant.