POPL class 2 (2020-04-27)

left-most

determines how to select a literal to resolve upon

and which clause is used when multiple are applicable

top-down

definite clauses

selection rule

SLD

linear resolution

the resolution step obtained from a clause in the next (and not not with a program pe of the feso/vent) is

pe of the

refers to the shape of the resulting proof trees

```
sibling(a,b).
sibling(b,c).
sibling(X,Y) := sibling(X,Z), sibling(Z,Y).
        :-sibling(a, X)
                                                       resolvents
               :-sibling(a, Z), sibling(Z, Y)
                                                         grow
          :-sibling(b,Y) :-sibling(a,U), sibling(U,Z),
                                               sibling(Z,Y)
           :-sibling(a, Z), sibling(Z, Y)
             infinite
              tree
```

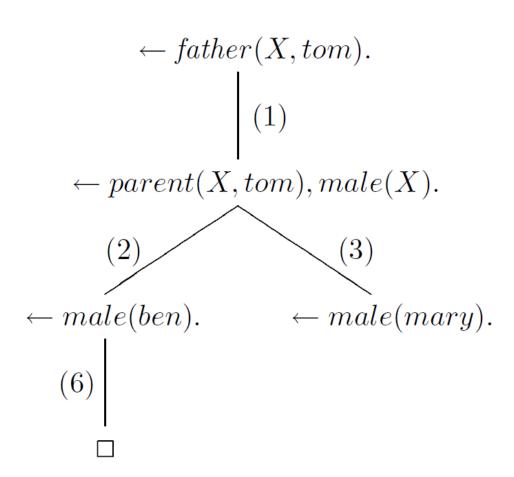
Cut: Pruning the SLD tree

- SLD-tree of a goal may have many failed branches
- programmer may want to prevent the interpreter from constructing failed branches by adding control information
- an infinite branch in the SLD-tree may prevent the interpreter from finding an existing correct answer.
- 'Cut' will control the search and avoid construction of some subtrees of the SLD-tree.

Cut

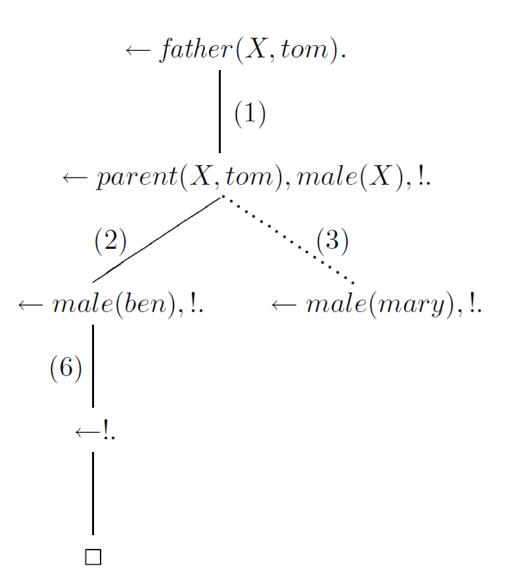
- (1) $father(X,Y) \leftarrow parent(X,Y), male(X).$
- (2) parent(ben, tom).
- $(3) \quad parent(mary, tom).$
- $(4) \quad parent(sam, ben).$
- (5) parent(alice, ben).
- (6) male(ben).
- (7) male(sam).

no person has more than one father, When a solution is found the search can be stopped



Cut

 $father(X; Y) \leftarrow parent(X; Y), male(X), !$



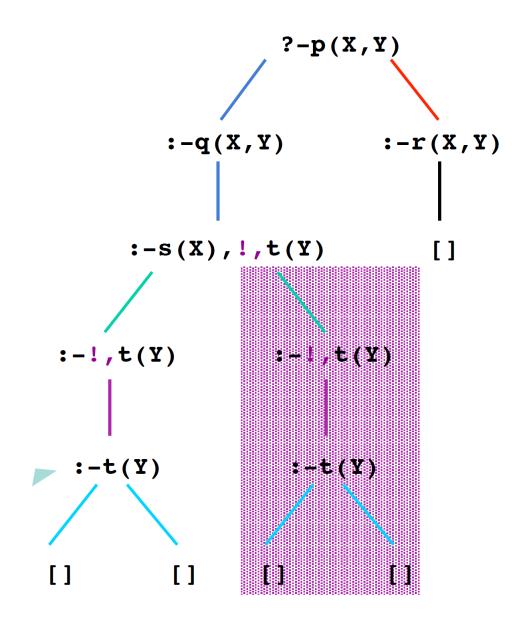
```
parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
father(john,paul).
mother(mary,paul).
```

```
?-parent(john,C)
:-father(john,C) :-mother(john,C)

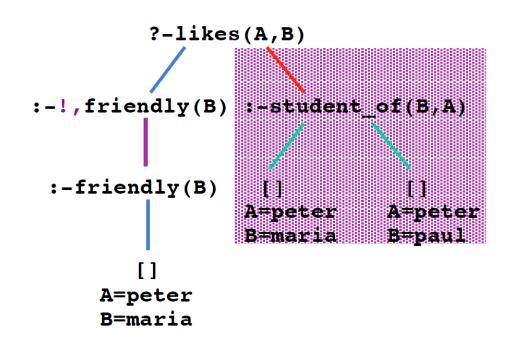
[] at this point, we know that exploring the alternative clause for parent/2 will fail
```

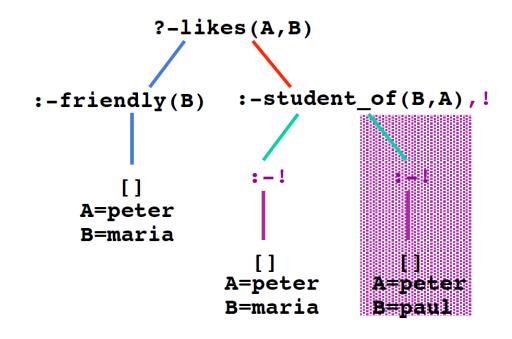
```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father (john, paul).
mother (mary, paul).
                              choice points on the
                               stack below and
                             including ?-parent
                            (john, C) are pruned
       ?-parent(john,C)
:-father(john,C),!
                     :-mother(john,C)
              tells Prolog that this is the
                only success branch
        []
```

```
p(X,Y):-q(X,Y).
p(X,Y):-r(X,Y).
q(X,Y):-s(X),!,t(Y).
r(c,d).
s(a).
s(b).
t(a).
t(b).
```



```
likes(peter,Y):-friendly(Y).
likes(T,S):-student_of(S,T).
student_of(maria,peter).
student_of(paul,peter).
friendly(maria).
```





likes(peter,Y):-!, friendly(Y).

likes(T,S):-student_of(S,T),!.

Cut problems

Cannot find more than one element

 $father(X; Y) \leftarrow parent(X; Y), male(X), !$

```
father(X,Y) \leftarrow parent(X,Y), male(X).
     parent(ben, tom).
                                                                  \leftarrow father(X,Y).
     parent(mary, tom).
(4)
     parent(sam, ben).
                                                            \leftarrow parent(X, Y), male(X), !.
     parent(alice, ben).
(5)
      male(ben).
(6)
      male(sam).
                                       \leftarrow male(ben),!. \leftarrow male(mary),!. \leftarrow male(sam),!. \leftarrow male(alice),!.
```

Cut problems

- (1) $proud(X) \leftarrow father(X, Y), newborn(Y).$
- (2) $father(X,Y) \leftarrow parent(X,Y), male(X).$
- $(3) \quad parent(john, mary).$
- $(4) \quad parent(john, chris).$
- (5) male(john).
- (6) newborn(chris).
- (2') $father(X,Y) \leftarrow parent(X,Y), male(X),!.$

proud(john) is "yes" since, as described, John is the father of Chris who is newborn.

answer to the goal *proud(john)* is "no".

Cuts

- two principal uses of cut: cut failing branches and to prune succeeding branches.
- Green cuts: Cutting of failing branches is harmless, asit does not alter the answers
- Red cuts: cutting succeeding branches is considered harmful



its left are deterministic and therefore do not have alternative solutions

some logical consequences of the program are not returned

Pruning the search by means of cut: red cuts ?-parent(john,c

```
?-parent(john,C)
parent (X,Y):-father (X,Y),!.
                                       :-father(john,C),!
                                                             :-mother(john,C)
parent(X,Y):-mother(X,Y).
father(john,paul).
father (john, peter).
                             same query,
                             but John has
                                            :-!
mother (mary, paul).
                                                              the cut is now red as a
                           multiple children
mother (mary, peter).
                           in this program
                                                             success branch is pruned
                                             []
      {C/beter
                                               ?-parent(P,paul)
parent (X,Y):-father (X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
                                     :-father(P, paul),!
                                                            :-mother(P,paul)
mother (mary, paul).
                          same program,
                            but query
                           quantifies over
                                               :-!
       P/maru
                                                                 parents rather
                           than children
                                                         the cut is only green when the
                                                         literal to its left is deterministic
```

16

Green cut

```
proud(X) \leftarrow father(X, Y), newborn(Y).
\vdots
father(john, sue).
father(john, mary).
\vdots
newborn(sue).
newborn(mary).
```

```
\leftarrow proud(X)
```

How to avoid getting the same answer twice or more.

Red cuts

$$min(X, Y, X) \leftarrow X < Y, !.$$

 $min(X, Y, Y).$

$$min(X, Y, X) \leftarrow X < Y, !.$$

 $min(X, Y, Y) \leftarrow X \ge Y.$

Negation as failure: specific usage pattern of cut

cut is often used to ensure clauses are mutually exclusive

cf. previous example

```
- q,!,r.
   only tried when q fails
                                                    []
```

such uses are equivalent to the higher-level

```
not_q:-q,!,fail.
                 where
:- q,r.
:- not_q,5.
                         not_q.
```

built-in predicate always false

Prolog's not/1 meta-predicate captures such uses:

in modern Prologs: use \+ instead of not

```
not(Goal) :- Goal, ! fail.
not(Goal).
                slight abuse of syntax
               equivalent to call(Goal)
```

not(Goal) is proved by failing to prove Goal

Negation as failure: SLD-tree where not(q) succeeds because q fails

?-p

```
p:-q,r.
p:-not(q),s.
                                                 :-not(q),s
                                     :-q,r
not(Goal):-Goal,!,fail.
not(Goal).
                                       :-q,!,fail,s
                                         q evaluated
                                            twice
                   version with! was more
                                                            []
                   efficient, but uses of not/1
                   are easier to understand
```