Loogilise programmi täitmise juhtimine

ITI0021

Sügis 2016

Loeng 7

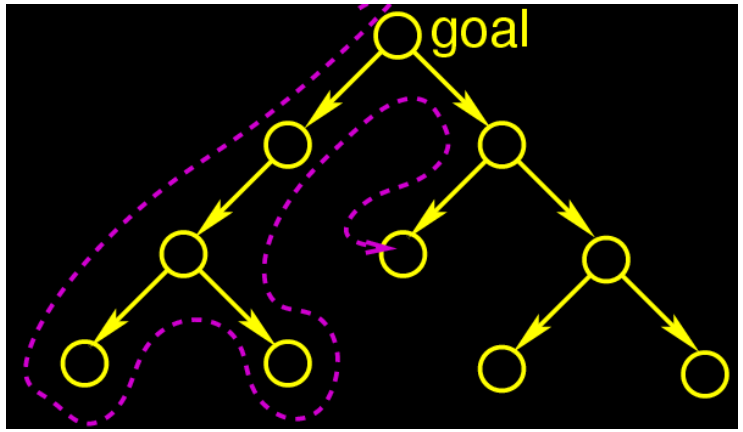
Tagasivõtuga otsing

Täitmise raja esitamine puuna:

· Juurtipp on Prologi päring

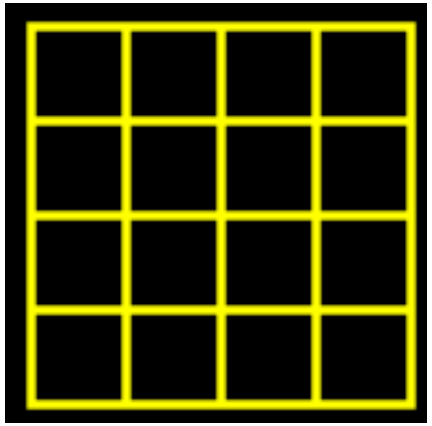
· Iga haru esitab ühe või mitme termi unifitseerimist pöördumisel reegli kehast.

· Puu lehed on muutujateta faktid



Näide: 4 lipu ülesanne

- Olgu 4 × 4 ruuduga malelaud
- Paigutada lauale 4 lippu nii, et samale reale, veerule ja diagonaalile ei satuks korraga 2 lippu.
- Otsingu kodeerib reegel `solution(Q1,Q2,Q3,Q4)`, kus `Q1,Q2,Q3,Q4` on lippude positsioonid.

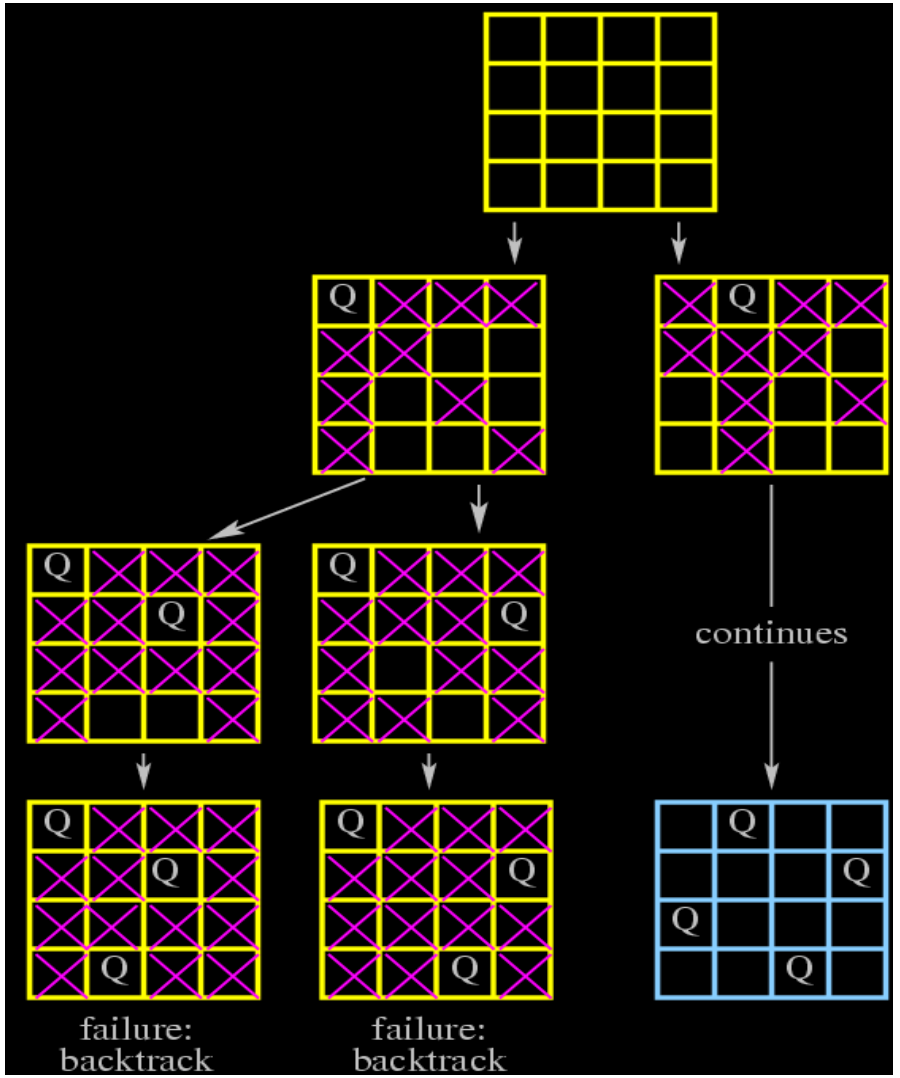


4 lipu paigutamine: otsingu puu

- Otsing **peatub** kui päringu kõik muutujad on väärtustatud.
- **Tagasivõtt**: kui jõutakse tupikusse, siis võetakse

muutujate jooksev väärtustus tagasi ja otsitakse

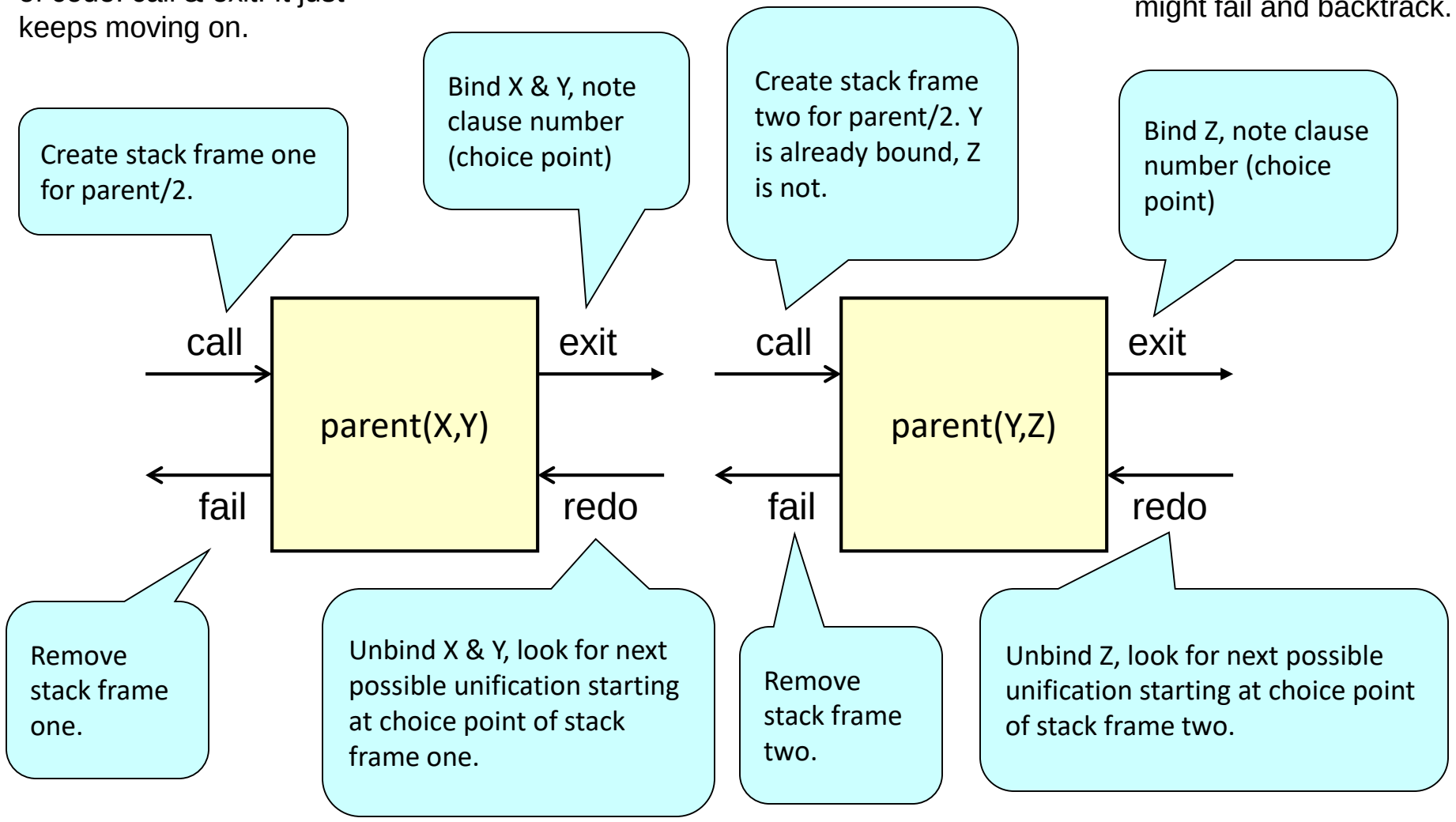
muutujatele uut väärtustust reegli järgmises alternatiivis.



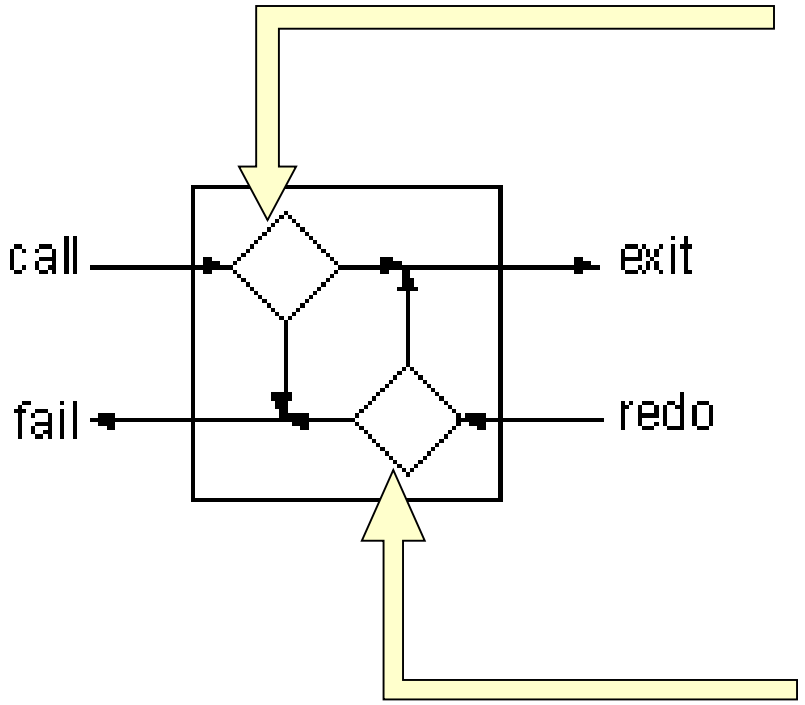
Täitimise mudel

A conventional language has two ports at each line of code: call & exit. It just keeps moving on.

At any line of code, Prolog might exit and move on; or might fail and backtrack.



Internal Flow of Control



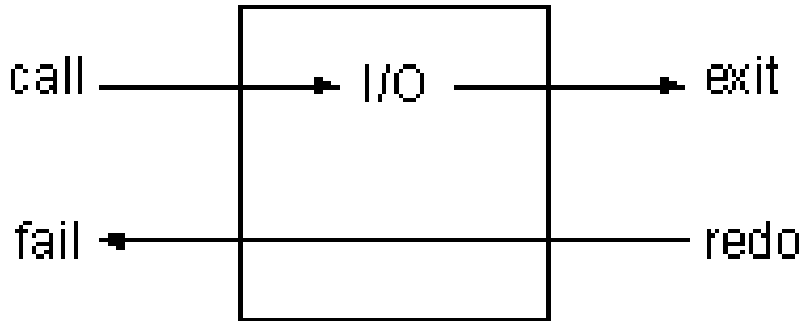
call -- Look for first clause of a predicate that unifies with goal.

If found, bind variables and **exit**, else **fail**.

redo -- From last clause tried, unbind variables bound with this goal and look for next clause that unfies with goal.

If found, bind variables and **exit**, else **fail**.

I/O Predicate Flow of Control



Do not change direction of execution.

in CALL, do I/O, out EXIT.

in REDO, out FAIL.

Control Built-Ins

fail – reverses forward control flow,
in CALL out FAIL
(*no way past fail/0*)

repeat – reverses backward control flow.
in CALL out EXIT
in REDO out EXIT
(*no way back after repeat/0*)

X, Y (and) – succeeds if X and Y succeed, otherwise fails.
REDO backtracks into Y, and then X.

X; Y (or) – succeeds if X or Y succeed.
REDO backtracks into X, and then Y.

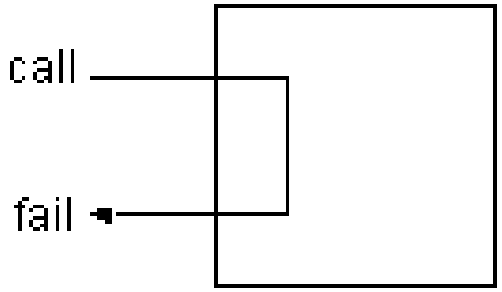
call(X) – treats X as a goal.

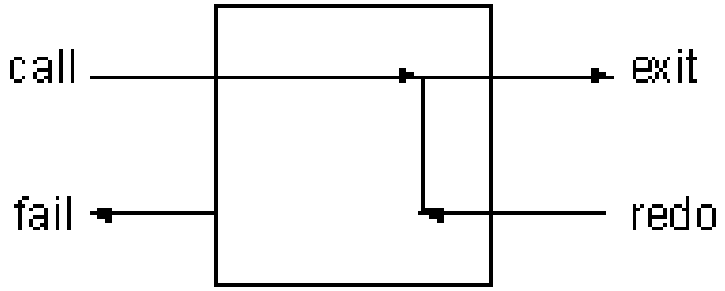
not(X) – calls X and succeeds if X fails, otherwise fails.
FAIL on REDO.

```
parent(michael, diego).
parent(ana, diego).
parent(pilar, ana).

male(michael). male(diego).

female(ana). female(pilar).
```

```
?- parent(X, diego), write(X), nl, fail.
michael
ana
false

?- repeat, write('> '), read(quit).
> hi.
> quit.
true

?- (male(X) ; female(X)).
X = michael ;
X = diego ;
X = ana ;
X = pilar ;
false

?- not(parent(pilar,diego)).
true
```

Flow of Control Predicates





fail -- always fails, reverses flow of control from right to left, to left to right.

repeat -- always succeeds, reverses flow of control from left to right, to right to left.

Cut- ja fail- operaatorid.

Küsimus:

Kuidas kirjutada reeglit: „Peeter armastab kõiki loomi välja arvatud madusid“?

- Lahendus 1:

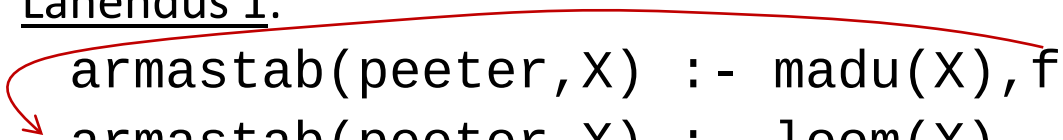
```
armastab(peeter,X) :- madu(X),fail.
armastab(peeter,X) :- loom(X).
```

Cut- ja fail- operaatorid.

Küsimus:

Kuidas kirjutada reeglit: „Peeter armastab kõiki loomi välja arvatud madusid“?

- Lahendus 1:



```
armastab(peeter,X) :- madu(X),fail.
armastab(peeter,X) :- loom(X).
```

See ei ole korrektne, sest fail sunnib tagasivõtuga valima järgmise alternatiivi ja tagastab `true` ka päringule `?- armastab(peeter, madu).`

Cut- ja fail- operaatorid.

Küsimus:

Kuidas kirjutada reeglit: „Peeter armastab kõiki loomi välja arvatud madusid“?

- Lahendus 1:

```
armastab(peeter,X) :- madu(X),fail.
armastab(peeter,X) :- loom(X).
```

- See ei ole korrektne, sest fail sunnib tagasivõtuga valima järgmise alternatiivi ja true.
- Lahendus 2:

```
armastab(peeter,X) :- madu(X),!,fail.
armastab(peeter,X) :- loom(X).
```

CUT-operaator !/0

- Tagasivõtt on Prologi otsingumootori oluline omadus,
- kuid võib anda ebaefektiivse programmi täitmise:
  - Prolog võib kulutada aega ja mälu otsingupuu nende harude läbimiseks, mis ei anna vajalikku tulemust.
  - Otsingu juhtimiseks on vaja keeles vahendeid
- Tagasivõttu saab juhtida cut-operaatoriga (!)
- cut-operaatoril ei ole argumente

cut-operaatori kasutamise näide

- Saame lisada selle reegli kehasse nagu iga teise predikaadi:
- Näide:

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

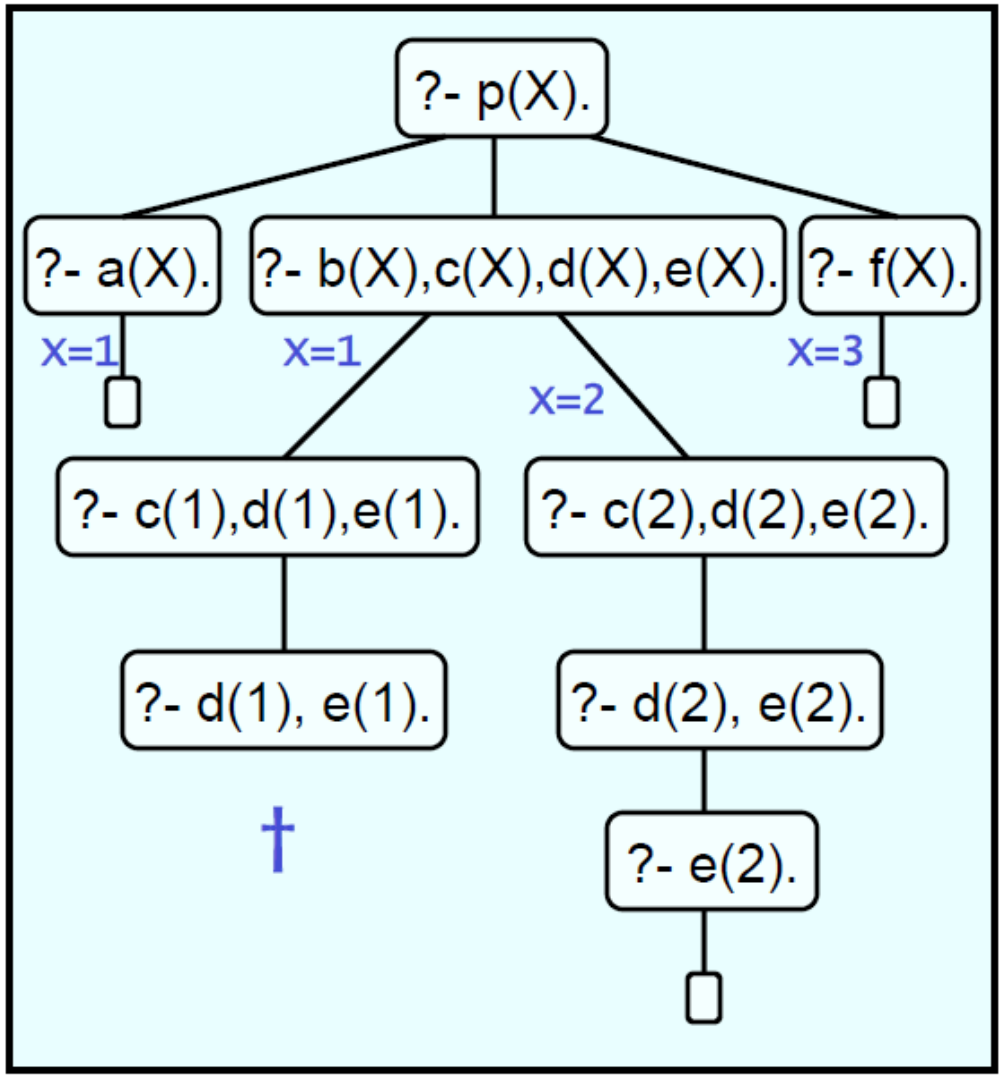
- Cut’i täitmine õnnestub alati
- Cuti kasutamine võimaldab säilitada otsingul lahendi, mis on leitud enne cut-operaatorini jõudmist.

Näide: reegel ilma cut-operaatorita

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

Kuidas toimub päringu täitmine?

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



cut-operaatori lisamine

- Lisame cut-operaatori reegli p/1 teise alternatiiv

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

- Päring ?- p(X). tagastab eelnevaga võrreldes erineva tulemuse.

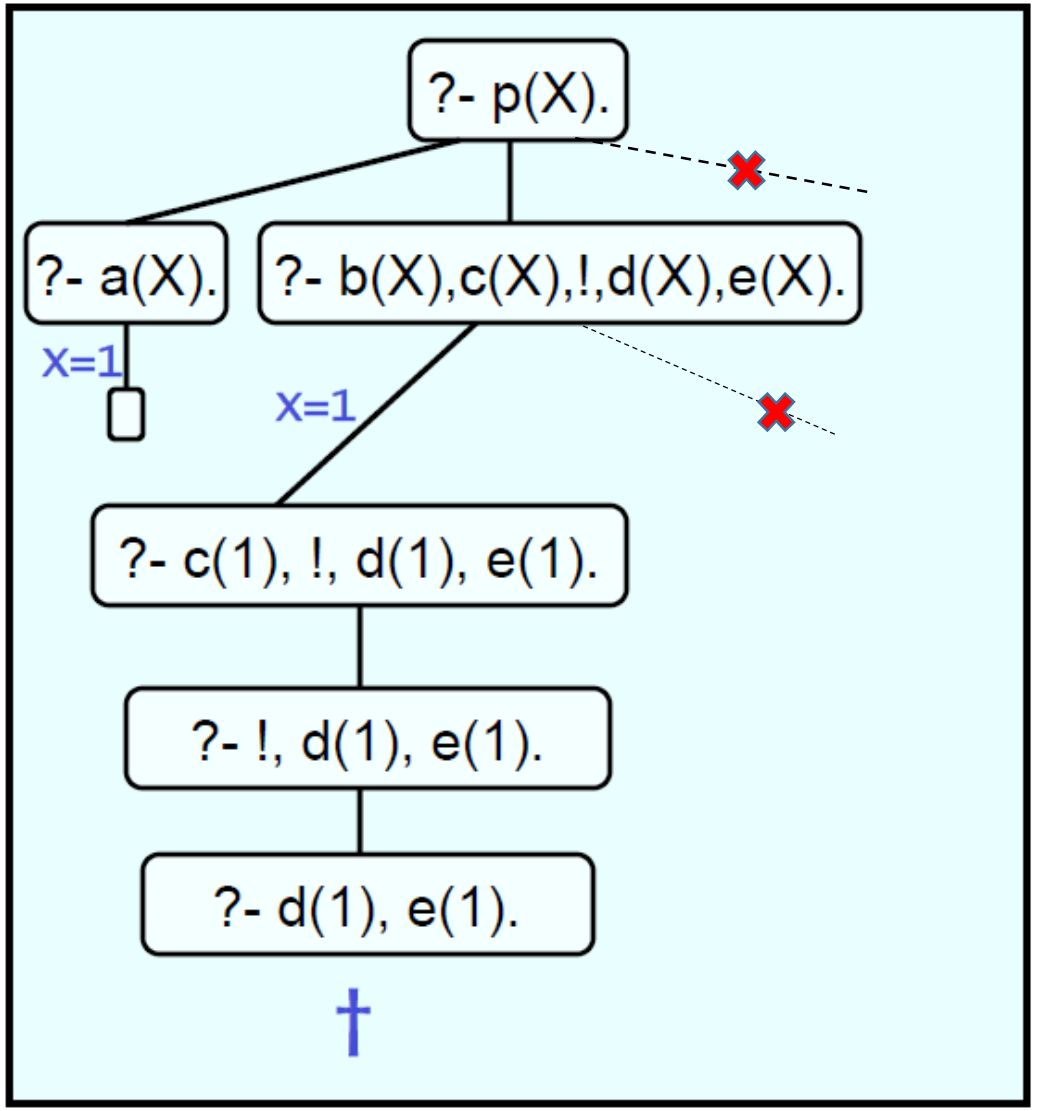
```
?- p(X).
X=1;
no
```

Näide reeglist, kus esineb cut-operaator

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

- Päring

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



Kuidas cut toimib?

- Olgu reegel

`q:-` q:- p1, … , pm, !, r1, … , rn.

`, !, r1, … , rn.`

- Tagasivõtu korral
  - toimub alternatiivsete lahenduste otsimine termidele `r1, …, rn`, st. termidele, mis on paremal poolt !-operaatorit,
  - termid p1, … , pm läbitakse tagasivõtu korral ilma neile uut lahendit otsimata,
  - samuti ei otsita uut lahendit reegli q/0 järgmisetele alternatiividele.

cut-operaatori kasutusnäiteid

- Vaatame predikaati max/3 , mis tagastab true, kui kolmas argument on kahe esimese argumenti maksimum.

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

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

?- max(7,3,7).
yes
```

```
?- max(2,3,2).
no

?- max(2,3,5).
no
```

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

Reeglis on liiasus:

- Päringu ?- max(3,4,Y). korral unifitseeritakse Y argumendi väärtusega 4.
- Kuid, kui nõuda järgmist lahendit, siis püütakse täita teist alternatiivi, mis antud argumentide väärtuste korral tagastab false.
- Tingimuse X > Y kontrollimine 2. alternatiivi kehas on liiasus.
- cut-operaatori lisamisega vabaneme sellest:

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

Kuidas cut toimib?

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

- Kui esimese alternatiivi tingimus on tõene, siis cut lõikab otsingupuust teise alternatiivi
- kui esimese alternatiivi tingimus ei kehti, siis täidetakse teine alternatiiv

Roheline ja punane cut

- **Roheline cut**'i kasutamine ei muuda programmi semantikat.
- Cut reeglis max/3 on roheline:
  - uus kood annab täpselt sama tulemuse, kui ilma cut'ta versioon
  - kuid on efektiivsem
- **Punane cut** muudab programmi semantikat.
- Näiteks lihtsustame max/3 reeglit eemaldades tingimuse 2.-st alternatiivist:

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

- Kuidas see mõjutab täitmist?

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

```
?- max(200,300,X).
X=300
yes
```

```
?- max(400,300,X).
X=400
yes
```

```
?- max(200,300,200).
yes
```

- Toome sisse unifitseerimise peale cut-i:

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

```
?- max(200,300,200).
no
```

Punase cut-ga kaasnevad ohud

- Punase cut-ga programmid
  - ei ole täisdeklaratiivsed
  - on rasked lugeda
  - võivad tekitada raskesti avastavaid vigu

fail/0

- Predikaat, mis alati tagastab false, kui Prolog proovib seda täita
- Kasulik, kui on vaja suunata Prolog tagasivõttu tegema.

Näide fail ja cut’i kasutamisest: Vincenti burgerid

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

burger(X):-  bigMac(X).
burger(X):-  bigKahunaBurger(X).
burger(X):-  whopper(X).

bigMac(a).
bigKahunaBurger(b).
bigMac(c).
whopper(d).
```

Tahame kodeerida erandi: „Vincent ei armasta suuri Kahuna burgereid“

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

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

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

Eitus kui fail

- Kodeerime eituse kui faili järgmise reegliga

```
neg(Goal):- Goal, !, fail.
neg(Goal).
```

- ja kodeerime Vincenti näite selle abil

```
enjoys(vincent,X):- burger(X),
                    neg(bigKahunaBurger(X)).

burger(X):-  bigMac(X).
burger(X):-  bigKahunaBurger(X).
burger(X):-  whopper(X).

bigMac(a).
bigKahunaBurger(b).
bigMac(c).
whopper(d).
```

```
?- enjoys(vincent,X).
X=a;
X=c;
X=d;
no
```

Standardpredikaat /+

- /+ on süsteemi predikaat, mis realiseerib predikaadi „neg“.

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

```
?- enjoys(vincent,X).
X=a;
X=c;
X=d;
no
```

- /+ on kohatundlik:

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

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