PROLOG
(CUTS E NEGAÇÃO)

Parcialmente adaptado de

http://www.learnprolognow.org/

O Cut

- O retrocesso (backtracking) é uma característica do Prolog
- Mas o retrocesso pode ser ineficiente:
  - O Prolog pode gastar muito tempo a explorar alternativas que não interessam
  - Seria interessante ter algum tipo de controlo
- O predicado cut !/0 oferece uma forma de controlar o retrocesso

Exemplo do cut

- O cut é um predicado Prolog, logo pode ser incluído no corpo das regras:
  - Exemplo:

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

- O cut sucede sempre
- Compromete o Prolog com as escolhas que foram feitas desde a chamada do predicado que unifica com a cabeça da regra

Explicação do cut

- Para explicar o cut vamos:
  - Analisar código Prolog sem cuts e observar o seu comportamento em termos de retrocesso
  - Adicionar cuts a esse código Prolog
  - Examinar como os cuts afectam o retrocesso

Exemplo: código sem cuts

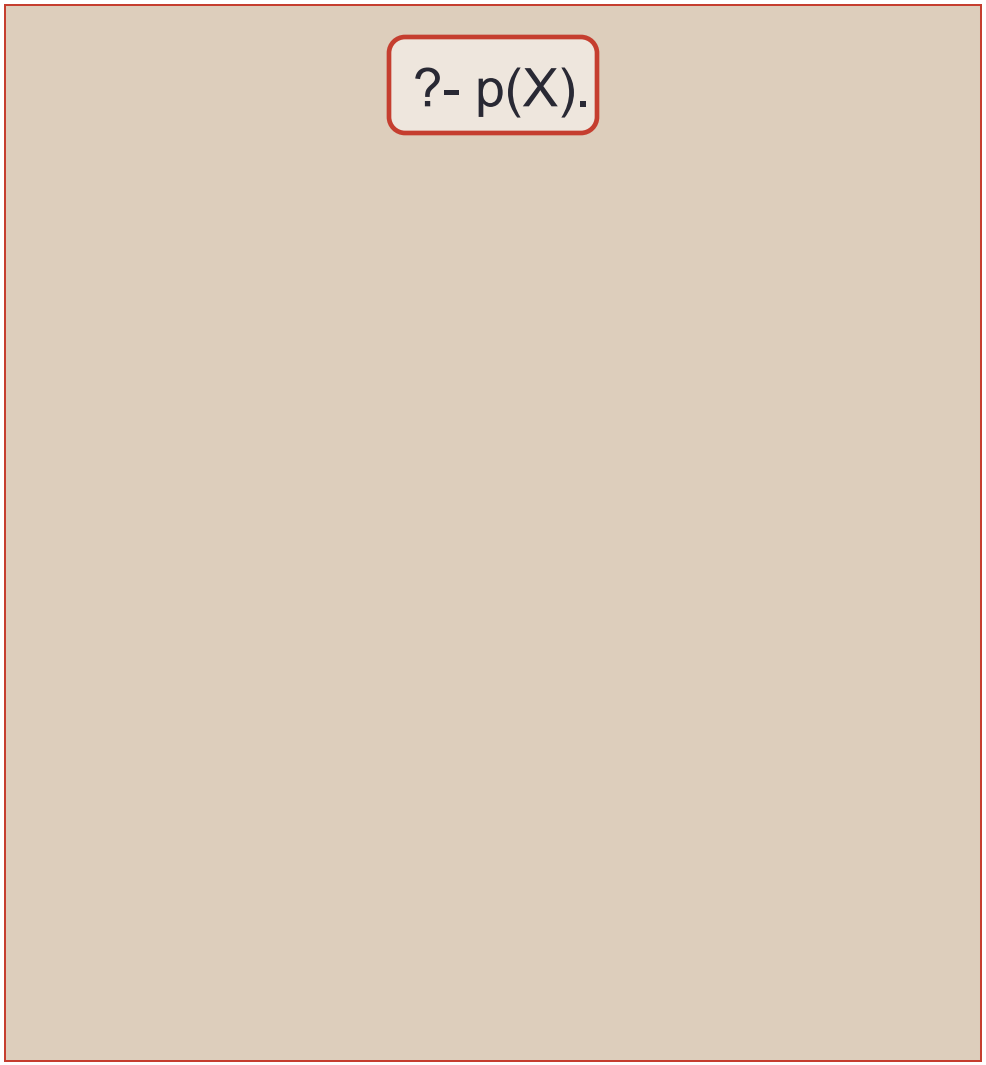
```
p(X):- a(X).
p(X):- b(X), c(X), d(X), e(X).
p(X):- f(X).
a(1).
b(1).   b(2).
c(1).   c(2).
d(2).
e(2).
f(3).
```

```
?- p(X).
```

Exemplo: código sem cuts

```
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).
```

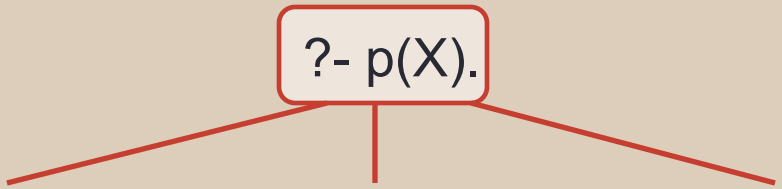
```
?- p(X).
```



Exemplo: código sem cuts

```
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).
```

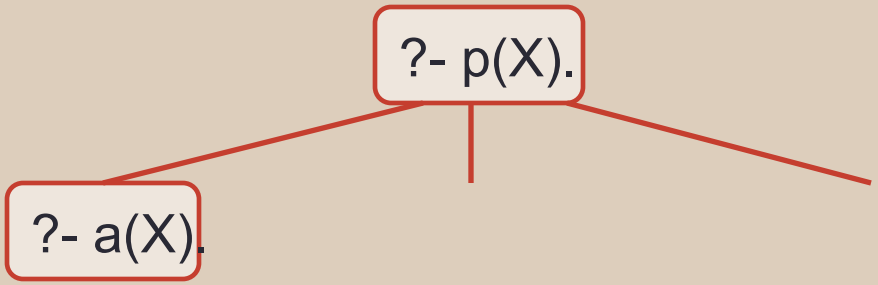
```
?- p(X).
```



Exemplo: código sem cuts

```
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).
```

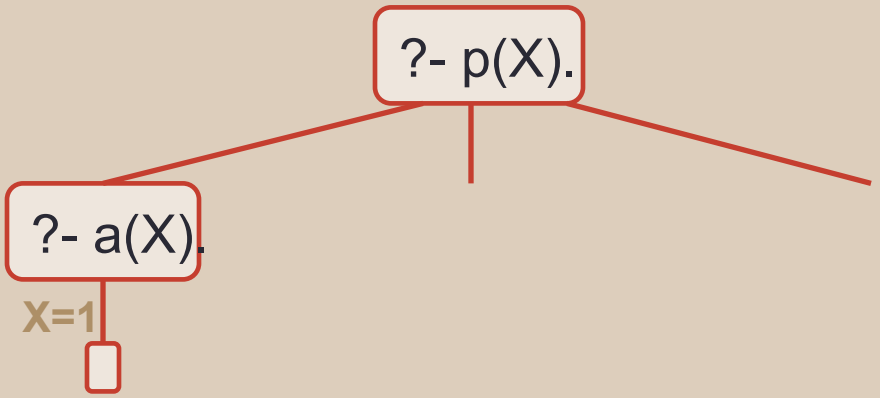
```
?- p(X).
```



Exemplo: código sem cuts

```
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).
```

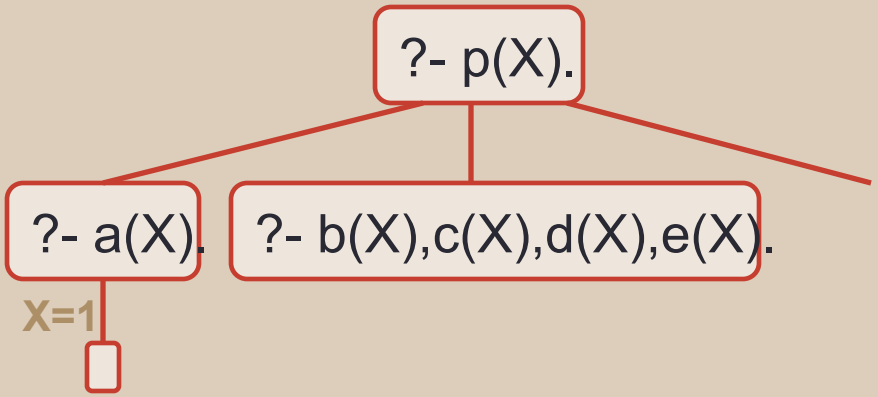
```
?- p(X).
X=1
```



Exemplo: código sem cuts

```
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).
```

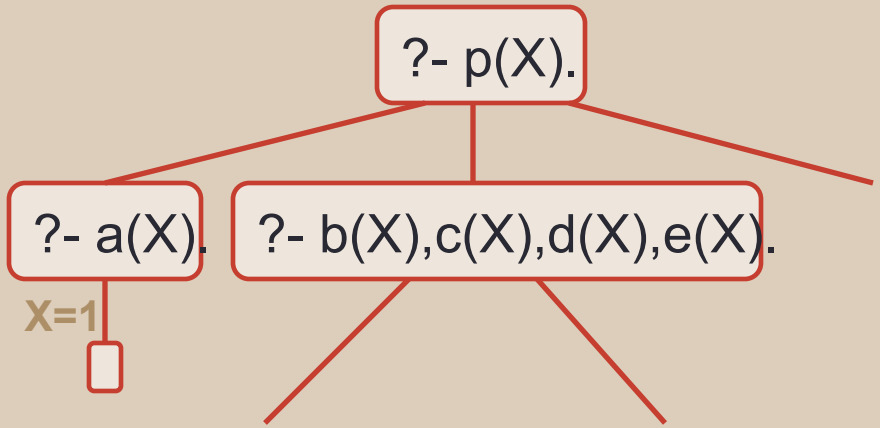
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

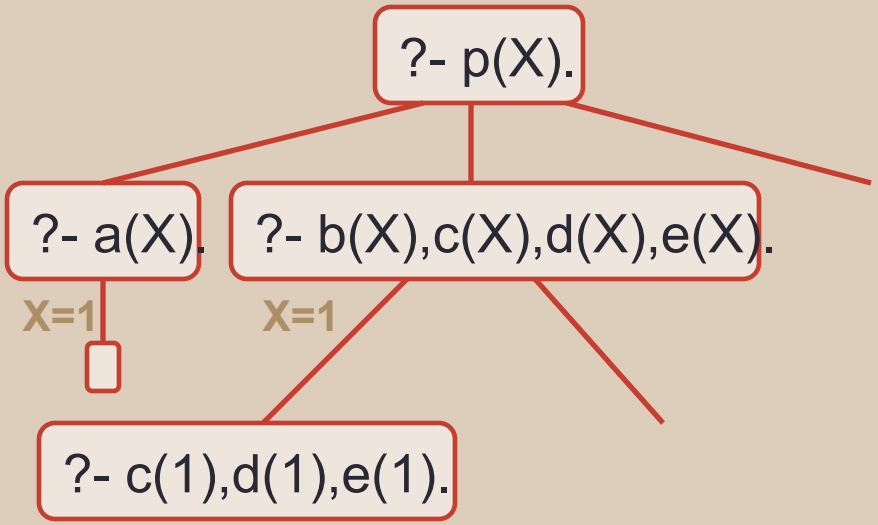
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

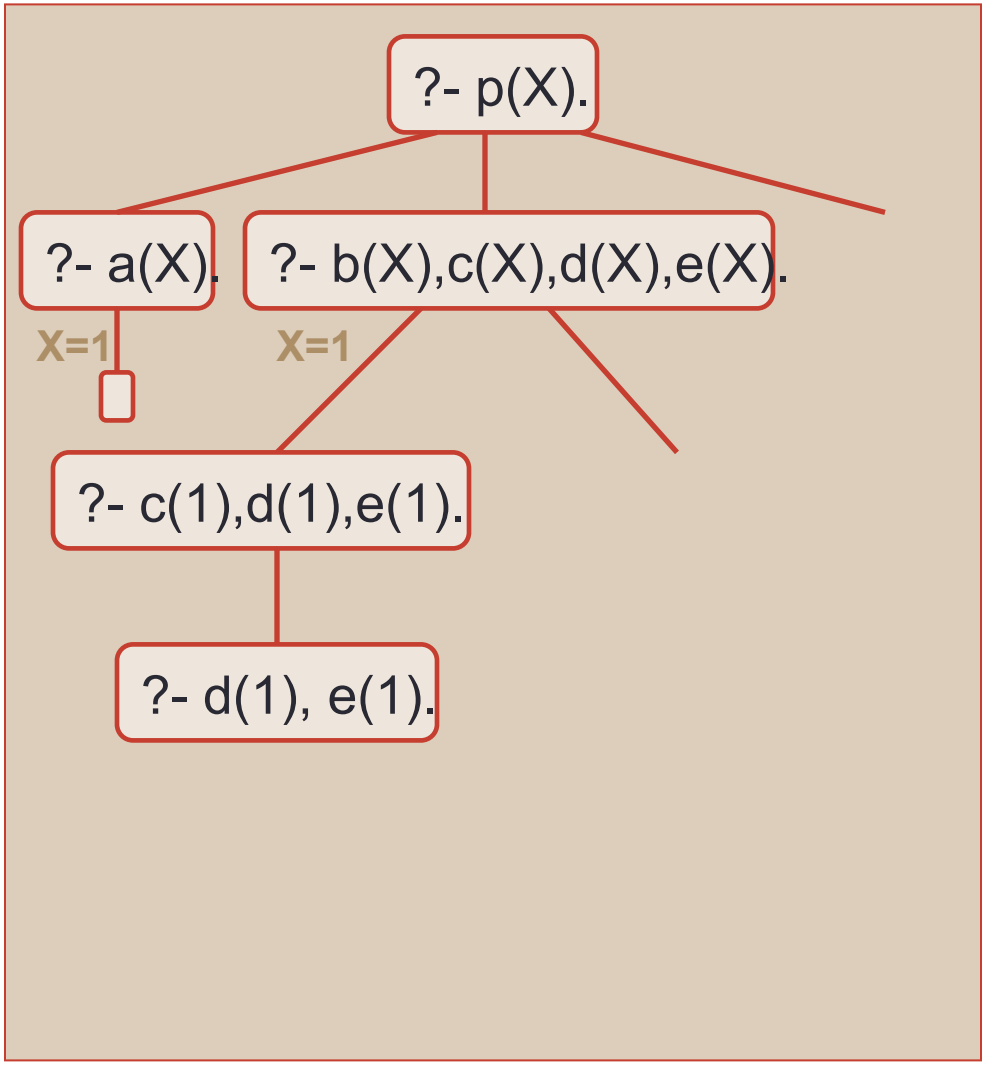
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

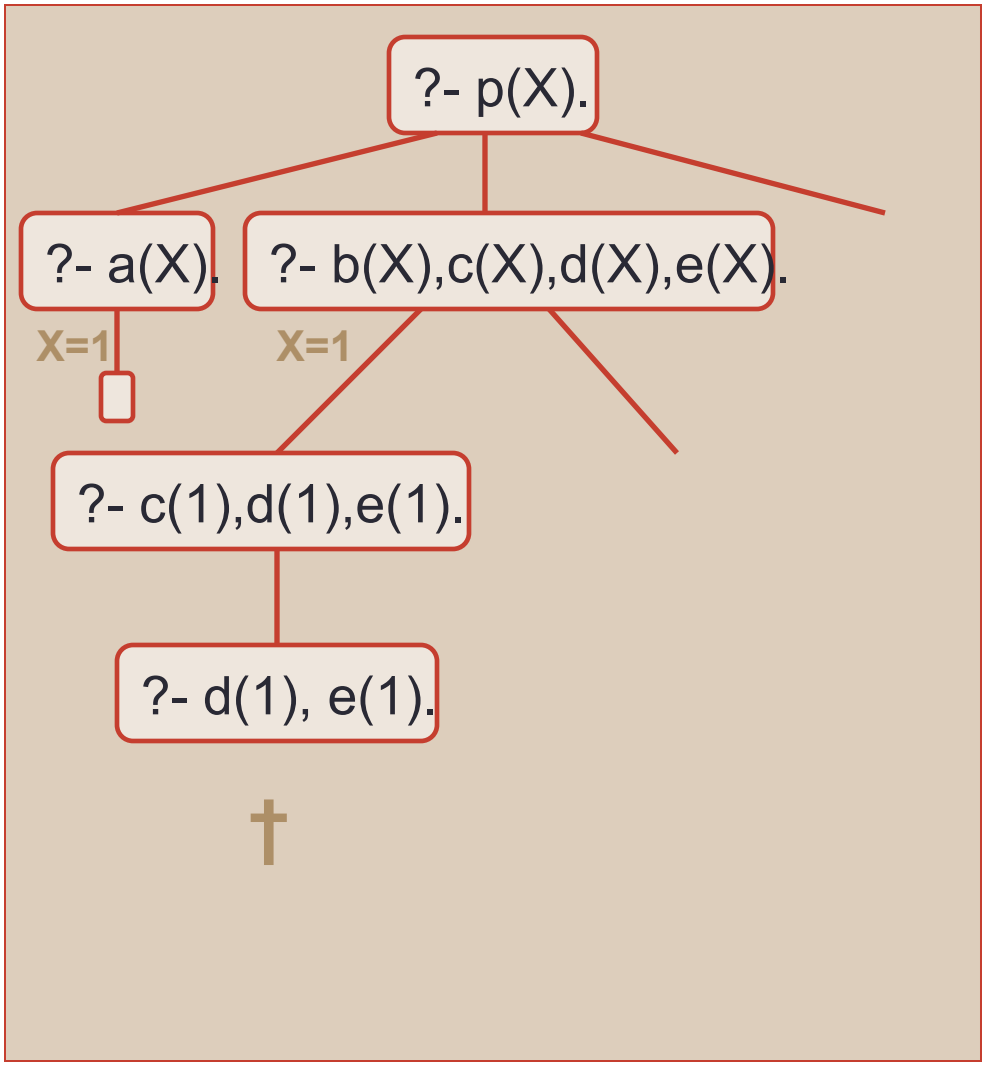
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

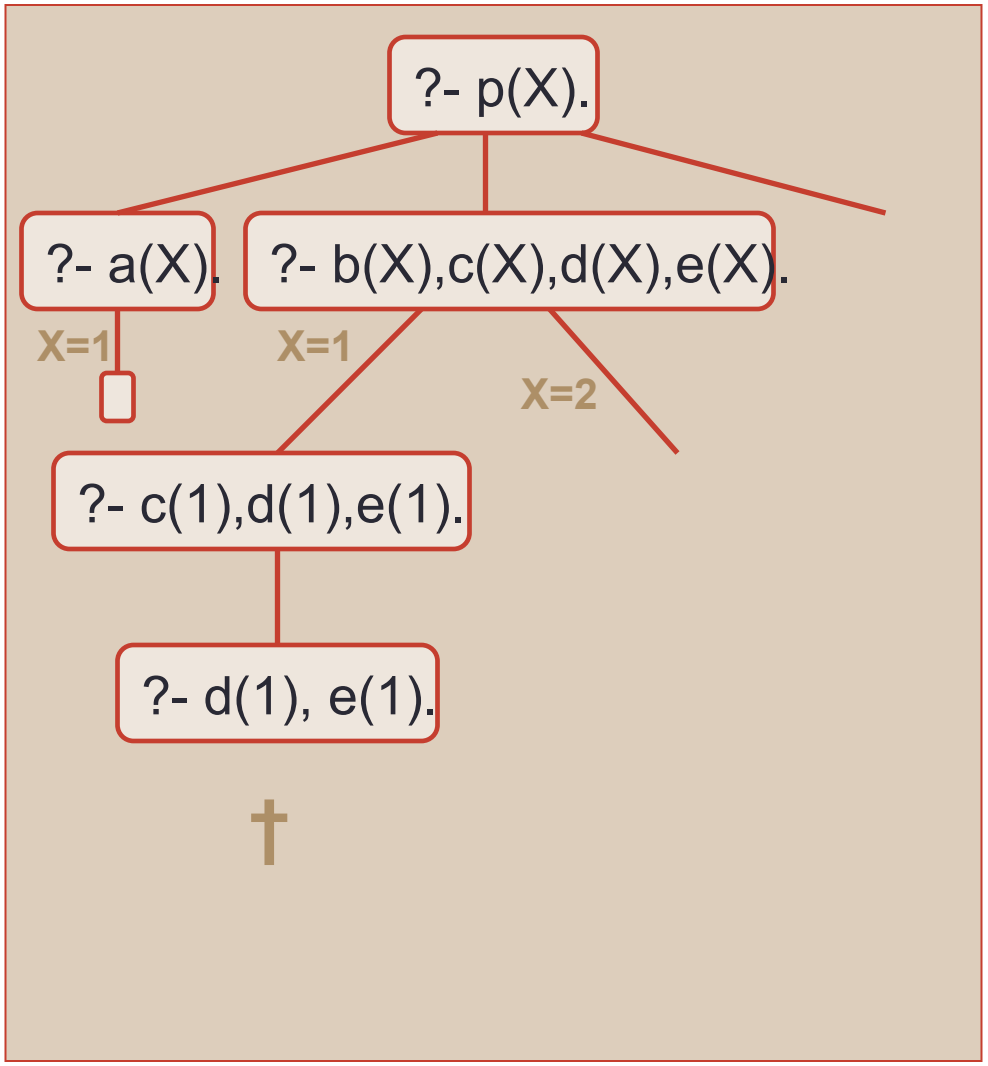
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

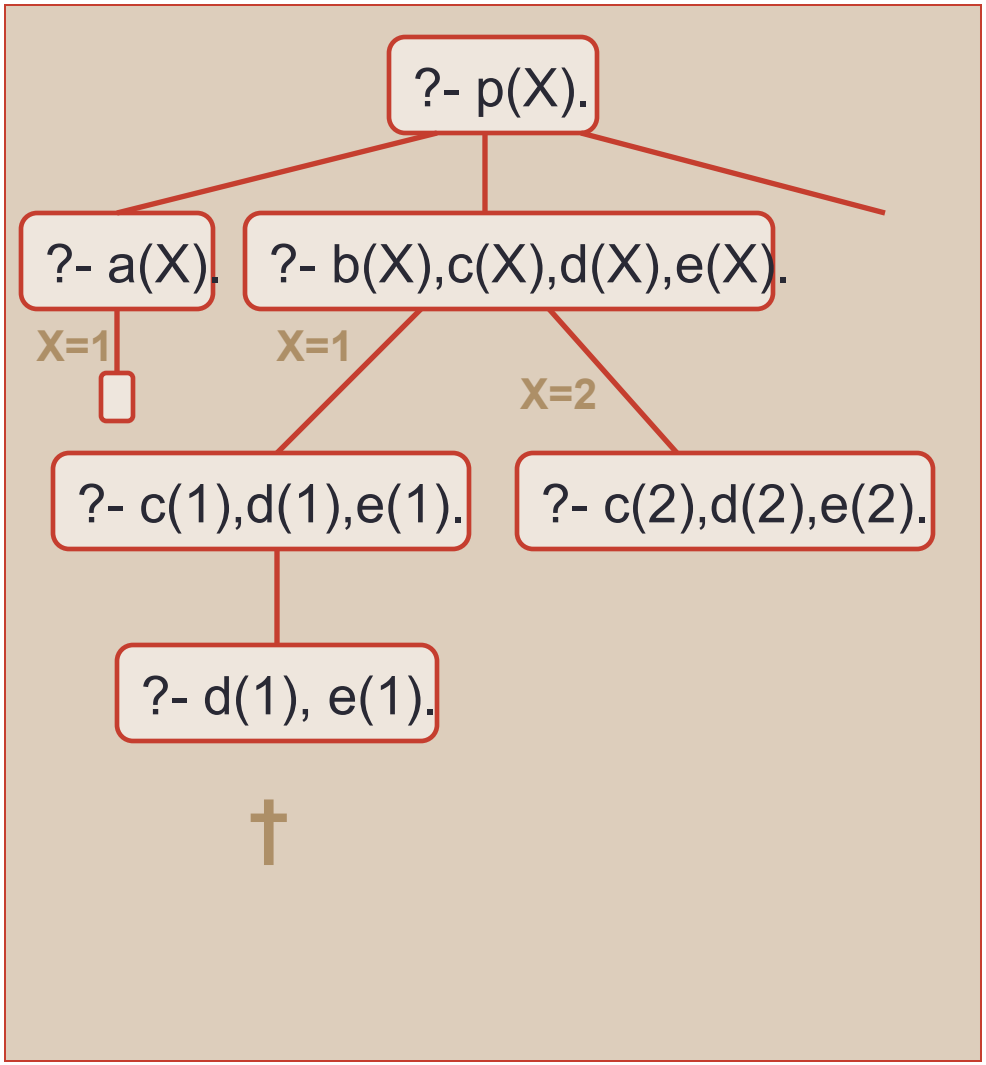
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

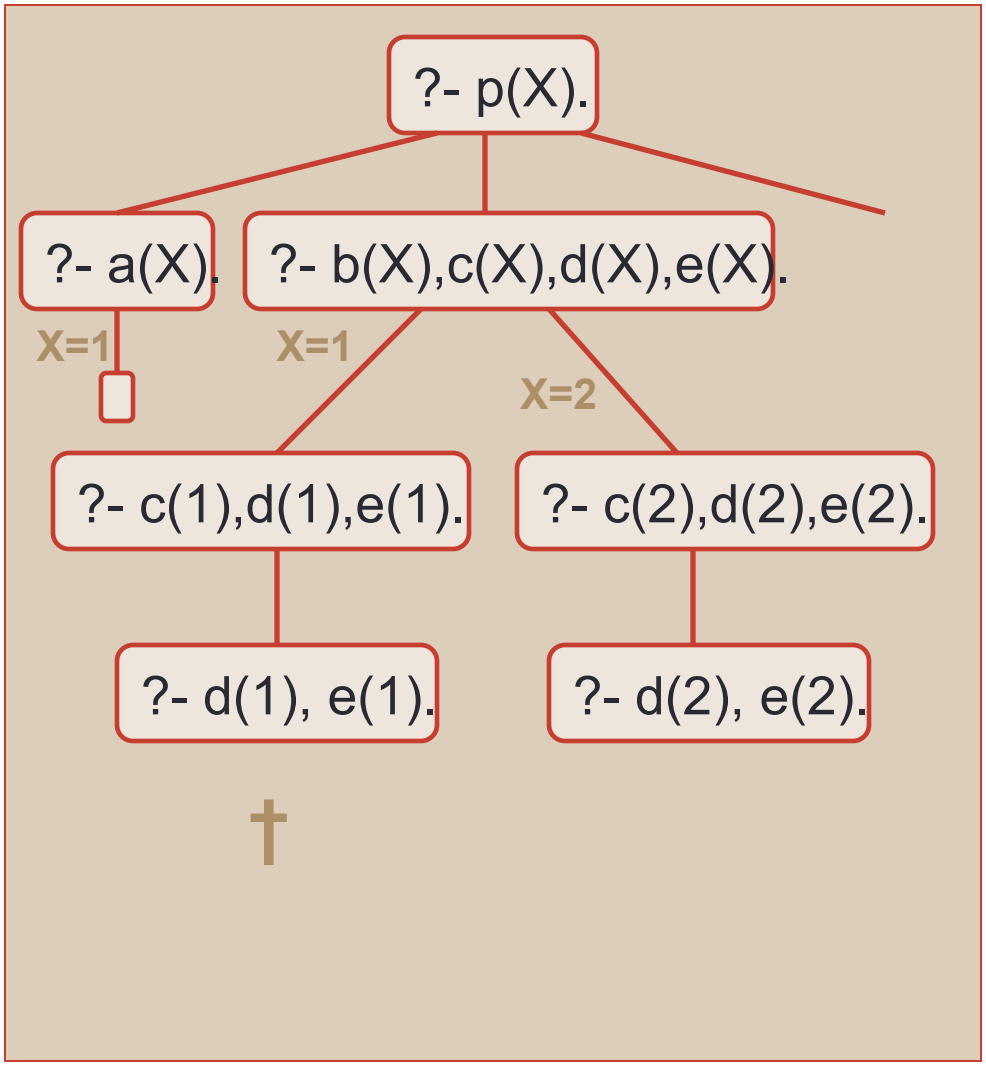
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

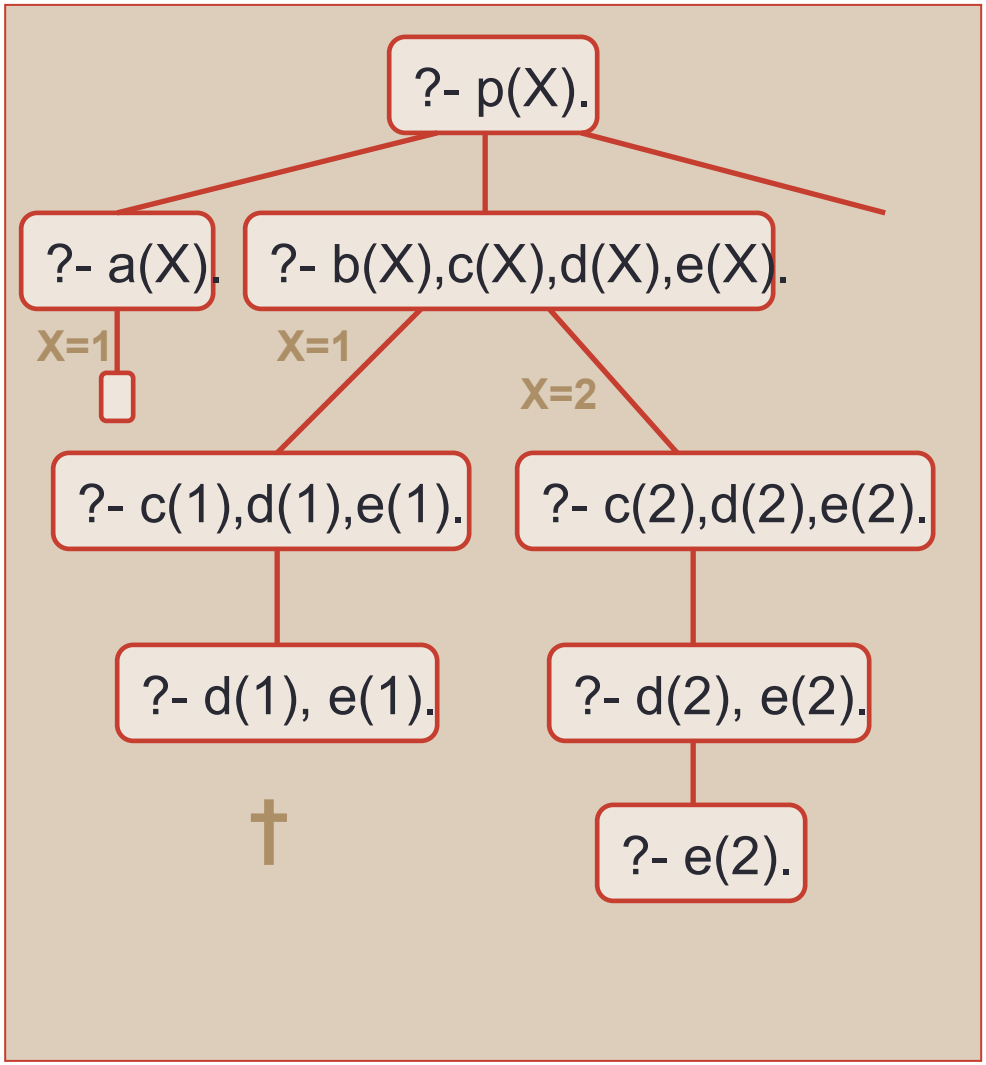
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

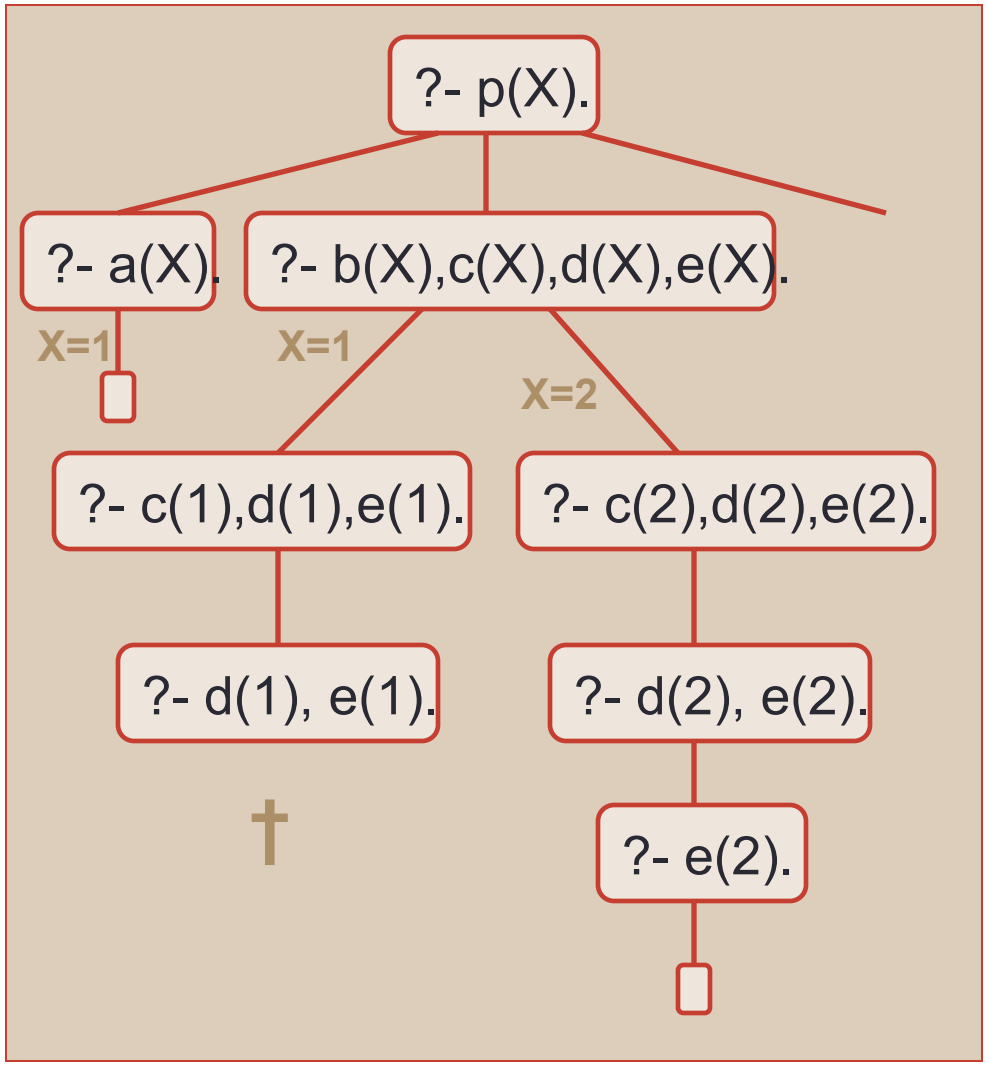
```
?- p(X).
X=1;
```



Exemplo: código sem cuts

```
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).
```

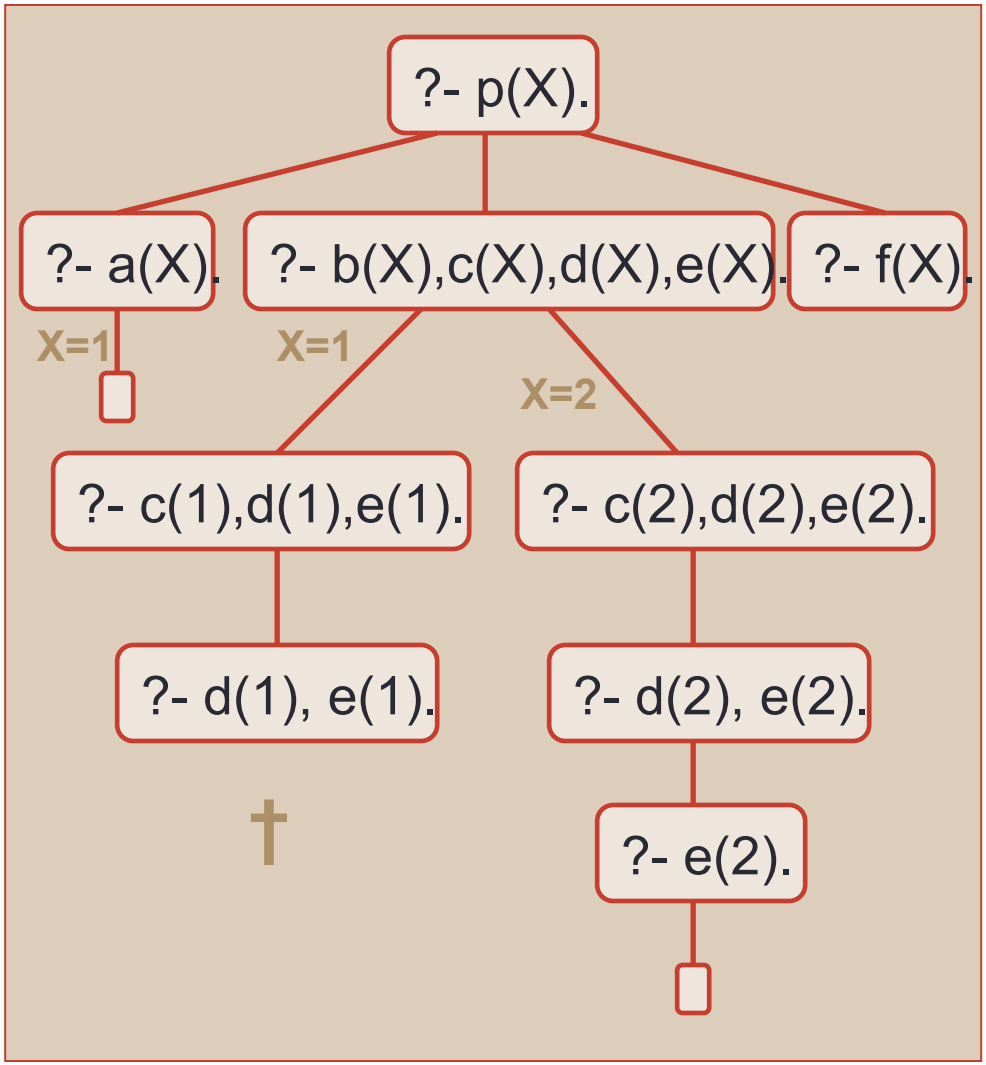
```
?- p(X).
X=1;
X=2
```



Exemplo: código sem cuts

```
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).
```

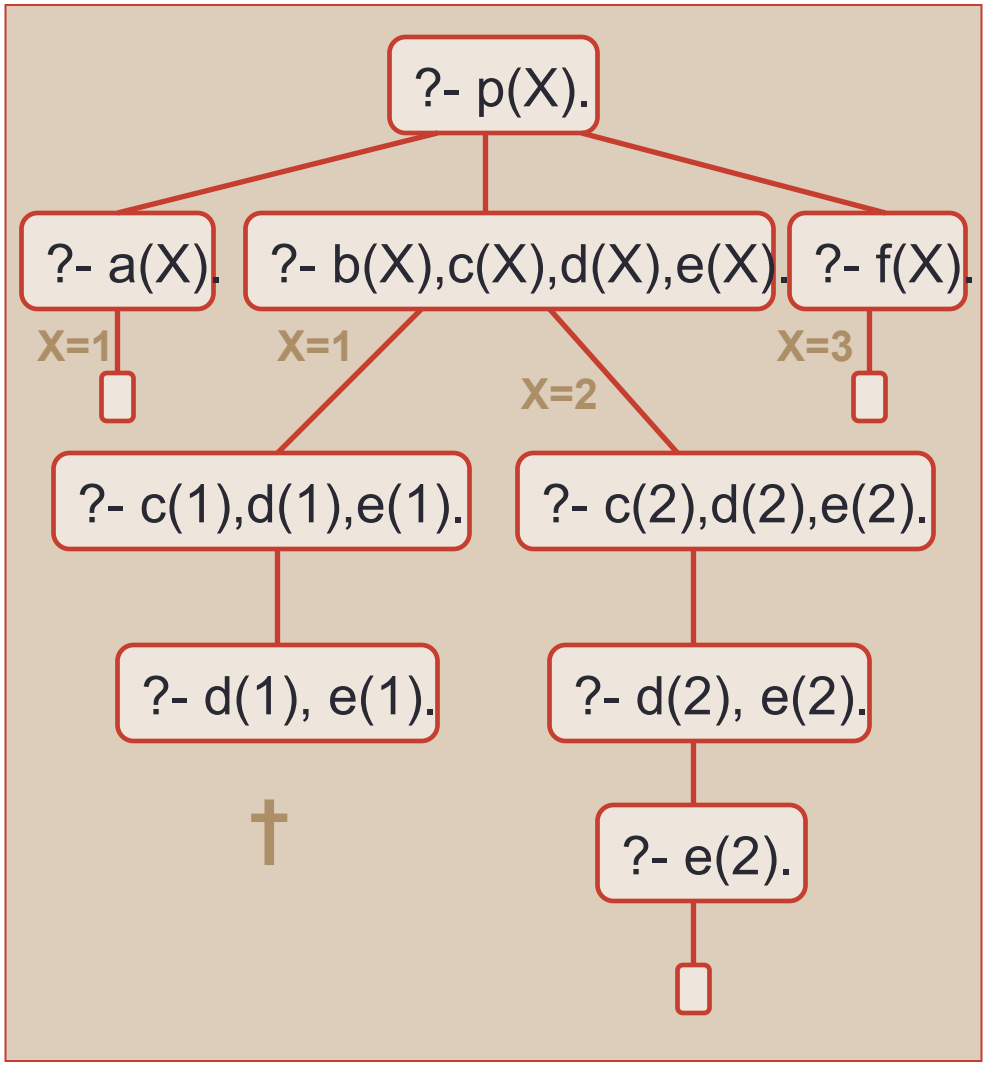
```
?- p(X).
X=1;
X=2;
```



Exemplo: código sem cuts

```
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).
```

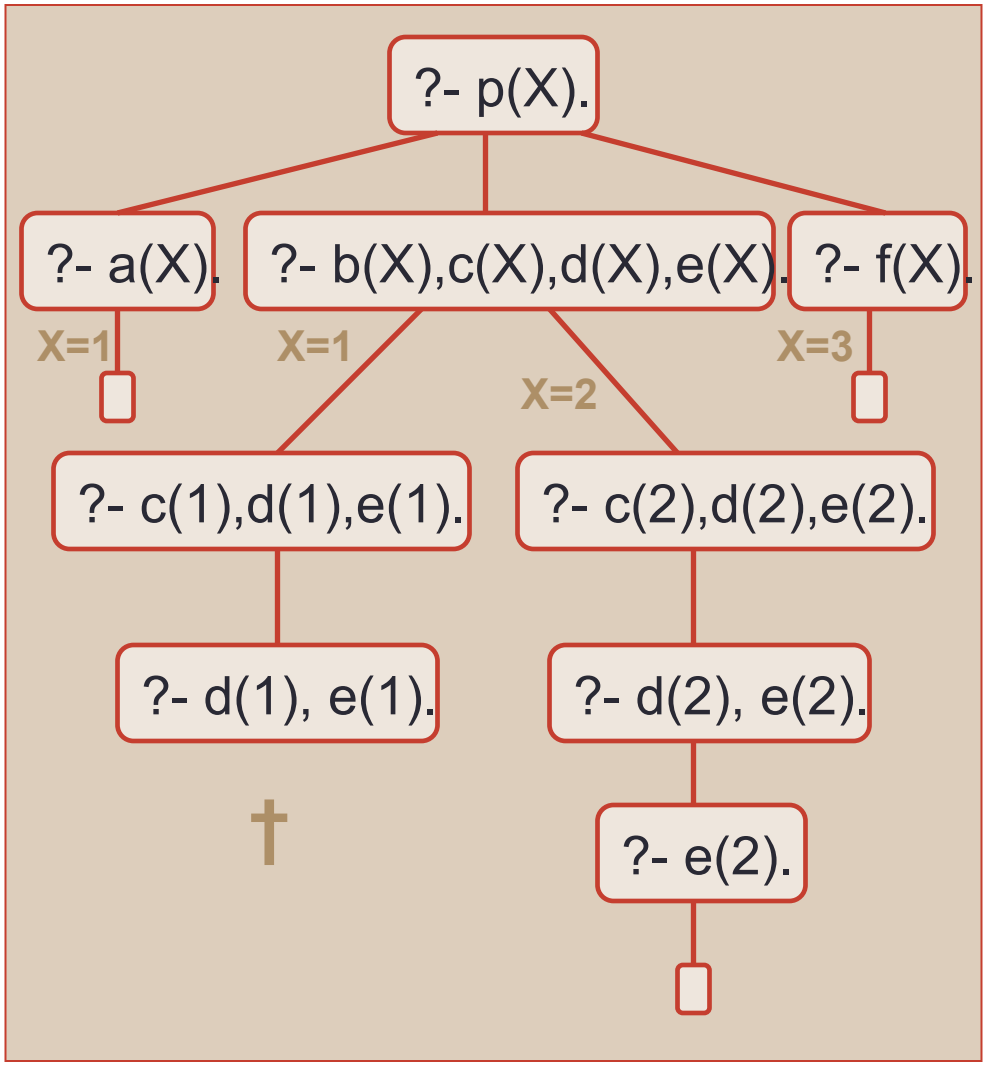
```
?- p(X).
X=1;
X=2;
X=3
```



Exemplo: código sem cuts

```
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).
```

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



Adicionar um cut

- Vamos supor que se insere um cut na segunda cláusula:

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

- Se agora fizermos a mesma interrogação obteremos a seguinte resposta:

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

Exemplo: cut

```
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).
```

```
?- p(X).
```

Exemplo: cut

```
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).
```

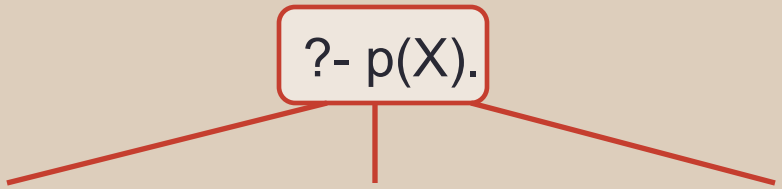
?- p(X).

?- p(X).

Exemplo: cut

```
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).
```

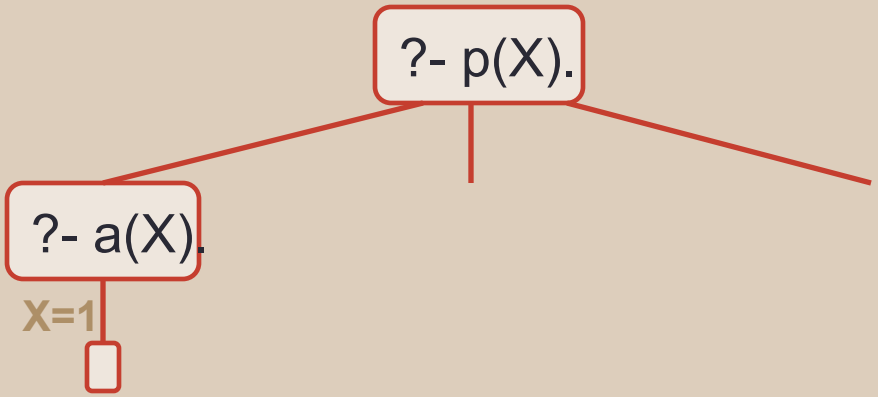
```
?- p(X).
```



Exemplo: cut

```
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).
```

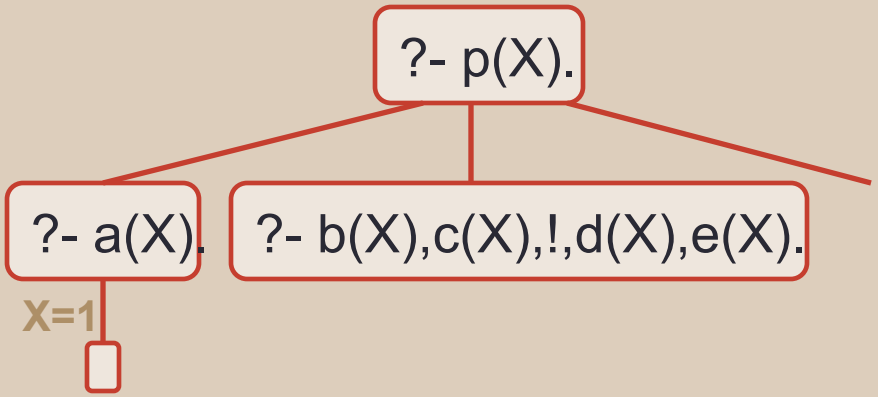
```
?- p(X).
X=1
```



Exemplo: cut

```
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).
```

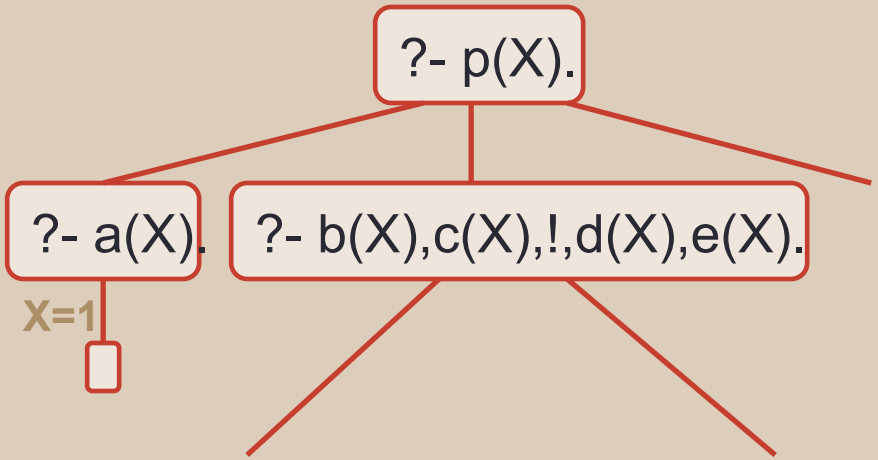
```
?- p(X).
X=1;
```



Exemplo: cut

```
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).
```

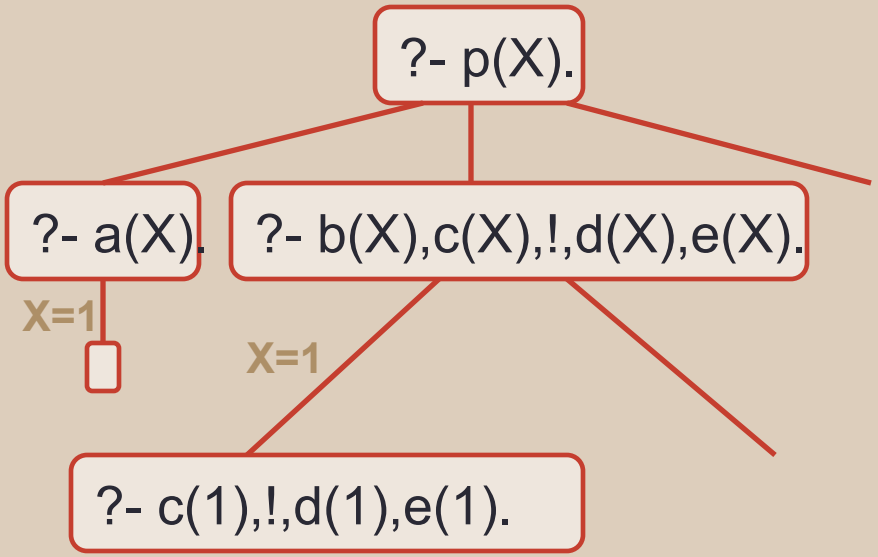
```
?- p(X).
X=1;
```



Exemplo: cut

```
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).
```

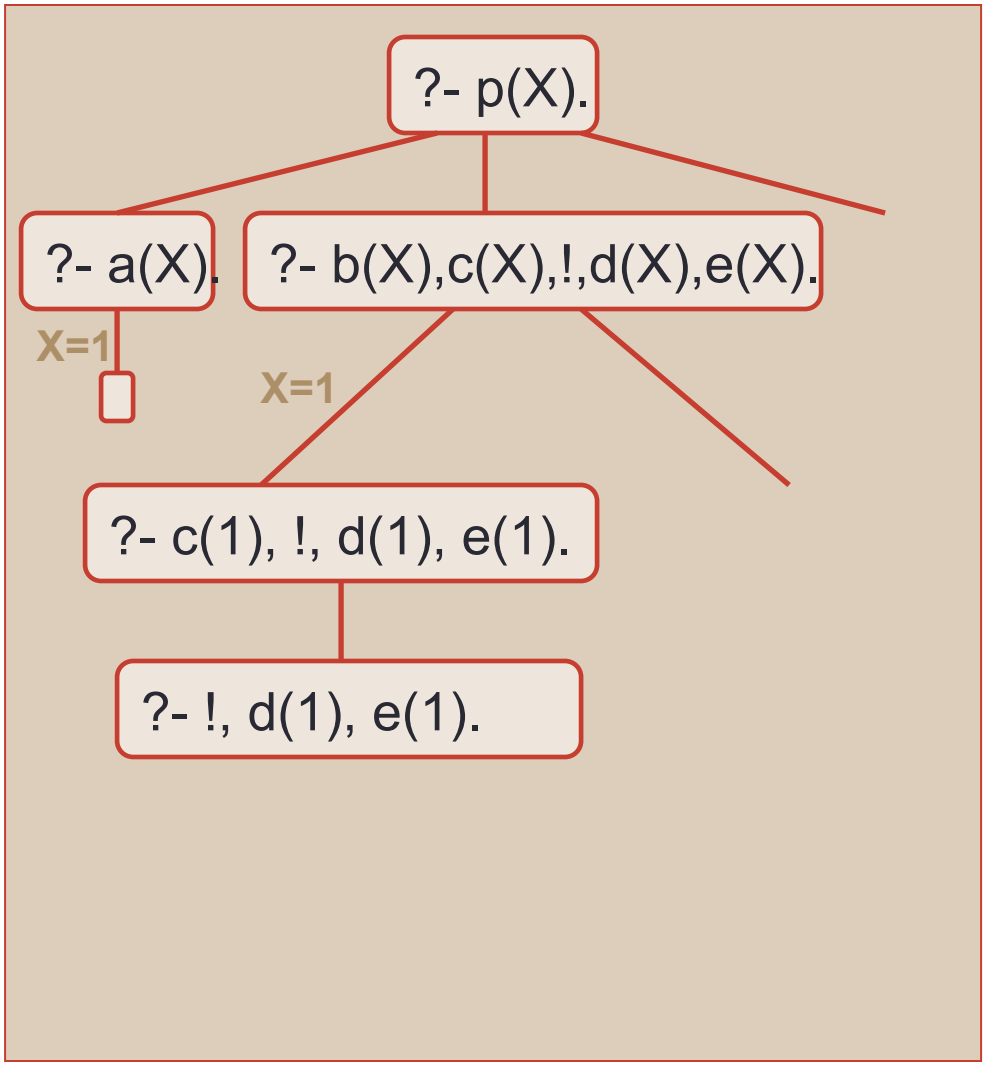
```
?- p(X).
X=1;
```



Exemplo: cut

```
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).
```

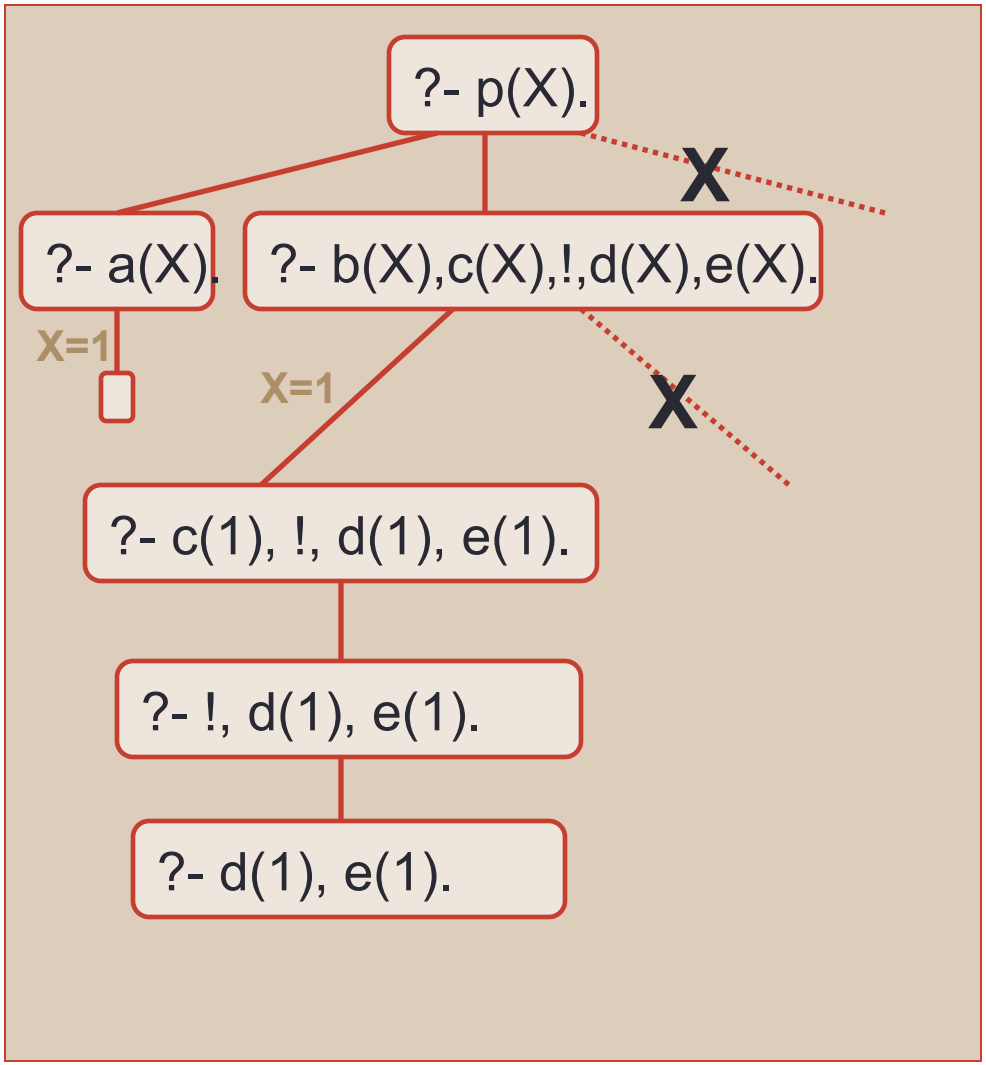
```
?- p(X).
X=1;
```



Exemplo: cut

```
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).
```

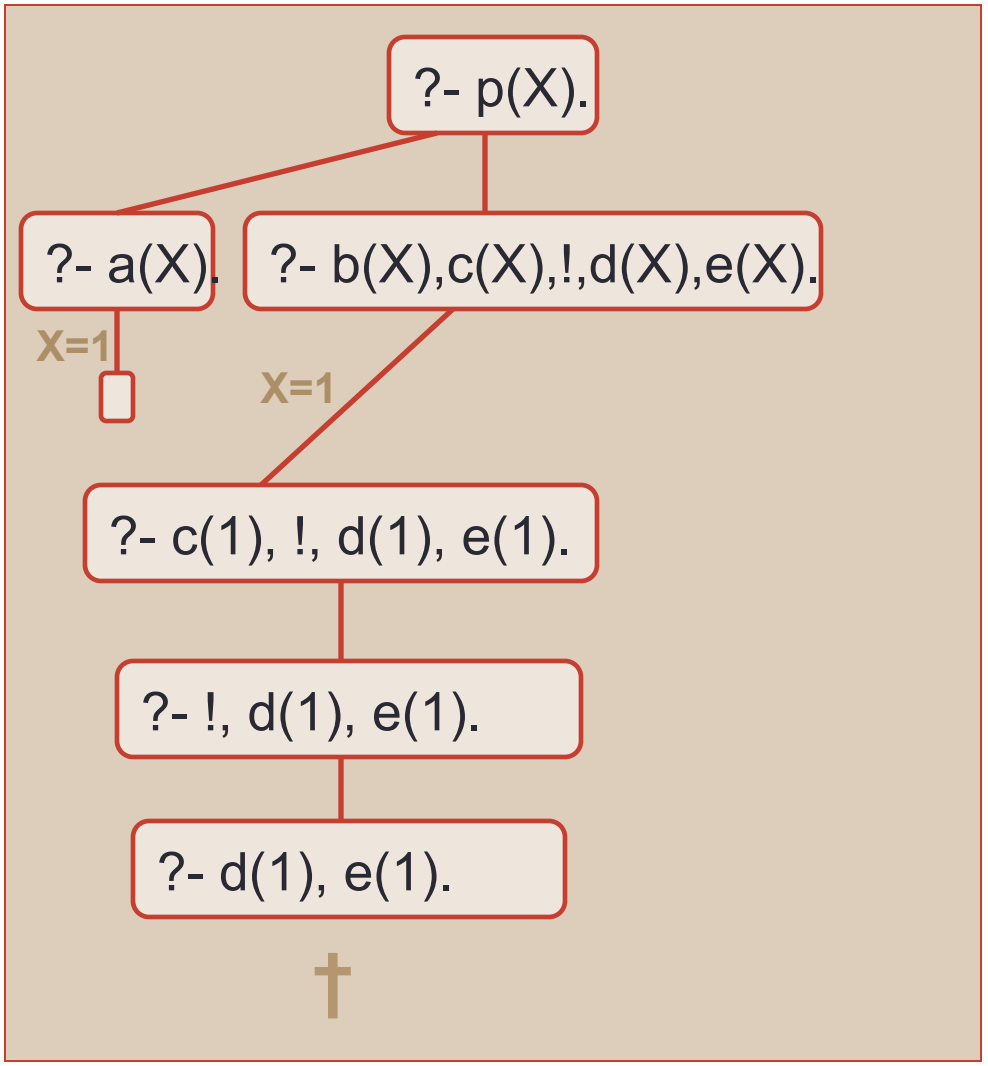
```
?- p(X).
X=1;
```



Exemplo: cut

```
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).
```

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



O que faz o cut

- O cut apenas compromete com as escolhas feitas desde a unificação do predicado com a cabeça da cláusula que contém o cut
- Por exemplo, numa regra da forma

  q:- p_1, … , p_n, !, r_1, … , r_n.

  Quando chega ao cut compromete-se:
  - com esta cláusula em particular para q
  - com as escolhas feitas por p_1, … , p_n
  - NÃO com as escolhas feitas por r_1, … , r_n

Usar o Cut

- Considere-se o predicado max/3 que sucede se o terceiro argumento é o máximo dos dois primeiros.

```
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
```

O predicado max/3

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

- Qual é o problema?
- Existe uma potencial ineficiência
  - Considere-se que é chamado com ?- max(3,4,Y).
  - Unifica correctamente Y com 4
  - Mas quando são pedidas alternativas, tenta satisfazer a segunda cláusula. Absolutamente desnecessário!

max/3 com cut

- Com o cut é possível resolver o problema

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

- Como funciona:
  - Se X =< Y sucede, o cut compromete com esta escolha, e a segunda cláusula de max/3 não é considerada
  - Se X =< Y falha, o Prolog prossegue para a segunda cláusula

Cuts verdes

- Os cuts que não alteram o significado de um predicado chamam-se **cuts verdes**
- O cut em max/3 é um exemplo de um cut verde:
  - o novo código dá exactamente as mesmas respostas que a versão anterior,
  - mas é mais eficiente

Outro max/3 com cut

- Porque não se remove o corpo de segunda cláusula? Afinal, ele é redundante.
- Será boa ideia?

```
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
```

Outro max/3 com cut

- Unificar depois de passar pelo cut.
- Agora já funciona.

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

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

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

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

Cuts vermelhos

- Cuts que alteram o significado de um predicado chamam-se **cuts vermelhos**
- O cut do max/3 revisto é um exemplo de um cut vermelho:
  - Se se remover o cut, não se obtém um programa equivalente
- Programas com cuts vermelhos
  - Não são puramente declarativos
  - Podem ser difíceis de ler
  - Podem levar a erros de programação subtis

Outro predicado Prolog: fail/0

- Como sugere o nome, este predicado falha sempre
- Não parece muito útil!
- No entanto: quando o Prolog falha, tenta novas alternativas por retrocesso

Vincent

- A combinação cut fail permite codificar excepções

```
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).
```

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

?- enjoys(vincent,b).
no

?- enjoys(vincent,c).
yes

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

Negação por falha

- A combinação cut-fail permite a implementação da negação por falha:

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

```
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
```

\+

- Em Prolog o operador prefixo \+ significa negação por falha
- As preferências do Vincent podem ser representadas da seguinte forma:

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

Negação por falha e lógica

- A negação por falha é diferente da negação em lógica
- Mudar a ordem dos predicados altera o resultado obtido:

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

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

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

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

Obter múltiplas soluções

- Podem existir muitas soluções para uma interrogação Prolog
- No entanto, o Prolog gera um solução de cada vez
- Por vezes necessitamos de obter *todas* as soluções para uma interrogação

Obter múltiplas soluções

- O Prolog tem três predicados que fazem isso: **findall**/3, **bagof**/3 e **setof**/3
- Todos estes predicados obtêm todas as soluções e juntam-nas numa lista
- Mas há diferenças importantes entre eles

findall/3

- A interrogação

  ?- findall(O,G,L)

- devolve uma lista L com todos os objectos O que satisfazem o golo G
  - Sucede sempre
  - Unifica L com a lista vazia se G não pode ser satisfeito

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- findall(X,descend(martha,X),L).
L=[charlotte,caroline,laura,rose]
yes

?- findall(f:X,descend(martha,X),L).
L=[f:charlotte,f:caroline,f:laura,f:rose]
yes

?- findall(X,descend(rose,X),L).
L=[ ]
yes

?- findall(d,descend(martha,X),L).
L=[d,d,d,d]
yes
```

findall/3

- A interrogação

  ?- findall(O,G,L)

- devolve uma lista L com todos os objectos O que satisfazem o golo G
  - Sucede sempre
  - Unifica L com a lista vazia se G não pode ser satisfeito
- Nem sempre é o adequado

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- findall(Chi,descend(Mot,Chi),L).
L=[charlotte,caroline,laura, rose,
caroline,laura,rose,laura,rose,rose]
yes
```

bagof/3

- A interrogação

  ?- bagof(O,G,L).

- devolve uma lista L com todos os objectos O que satisfazem o golo G
  - Só sucede se o golo G sucede
  - Instancia variáveis livres de G

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- bagof(Chi,descend(Mot,Chi),L).
Mot=caroline
L=[laura, rose];
Mot=charlotte
L=[caroline,laura,rose];
Mot=laura
L=[rose];
Mot=martha
L=[charlotte,caroline,laura,rose];
no
```

bagof/3 com ^

- A interrogação

  ?- bagof(O,G,L).

- devolve uma lista L com todos os objectos O que satisfazem o golo G
  - Só sucede se o golo G sucede
  - Instancia variáveis livres de G

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- bagof(Chi,Mot^descend(Mot,Chi),L).
L=[charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
yes
```

setof/3

- A interrogação

  ?- setof(O,G,L).

- devolve uma lista ordenada L com todos os objectos O que satisfazem o golo G
  - Só sucede se o golo G sucede
  - Instancia variáveis livres de G
  - Remove duplicados de L
  - Ordena os elementos de L

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- bagof(Chi,Mot^descend(Mot,Chi),L).
L=[charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
yes

?- setof(Chi,Mot^descend(Mot,Chi),L).
L=[caroline, charlotte, laura, rose]
yes
```