Prolog
(Factos, Regras e Interrogações)

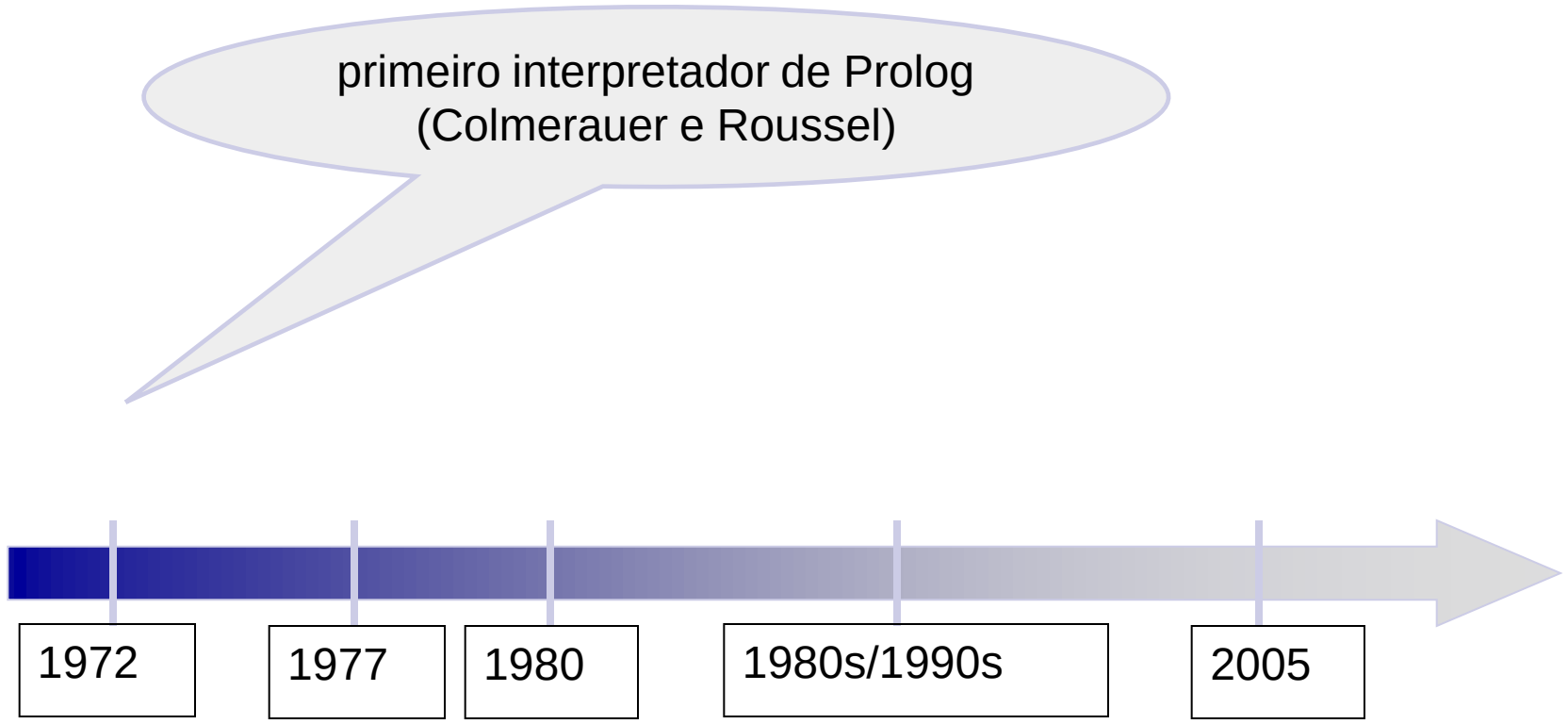
Parcialmente adaptado de

http://www.learnprolognow.org/

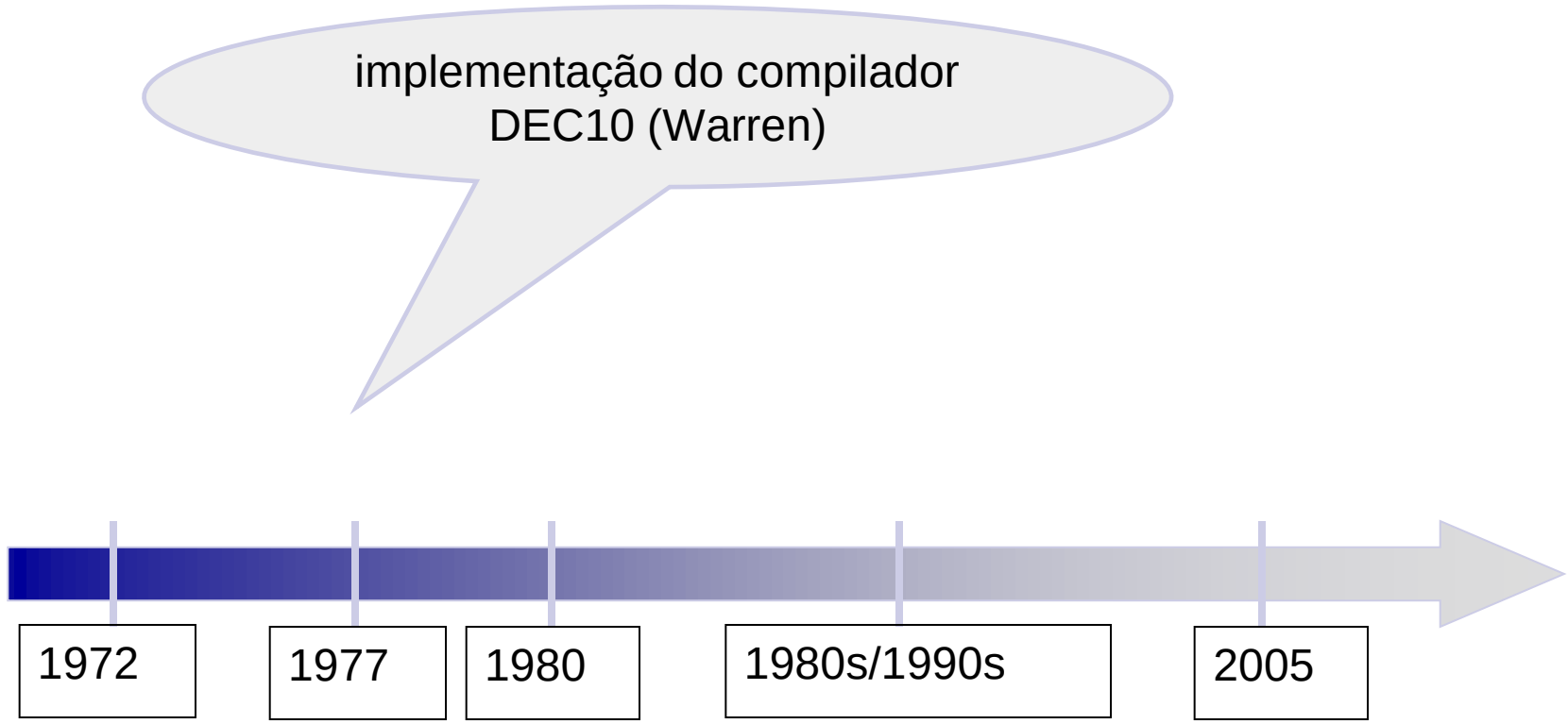
Prolog

- "PROgramação em LÓGica"
- Declarativa
- Muito diferente das outras linguagens de programação (procedimentais)
- Adequada para tarefas baseadas em conhecimento

História do Prolog

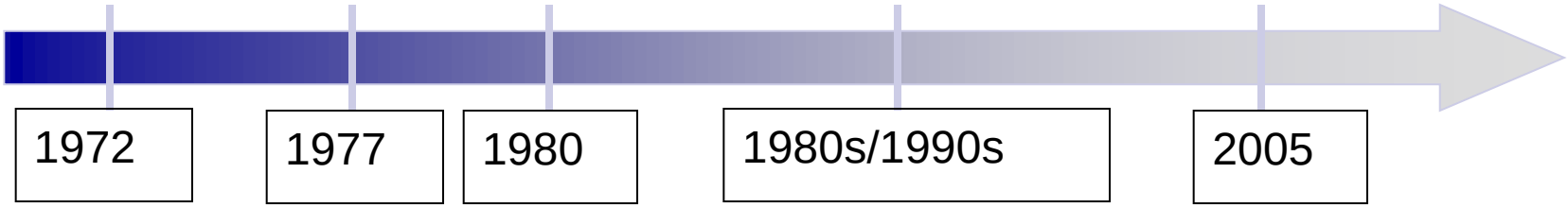


História do Prolog

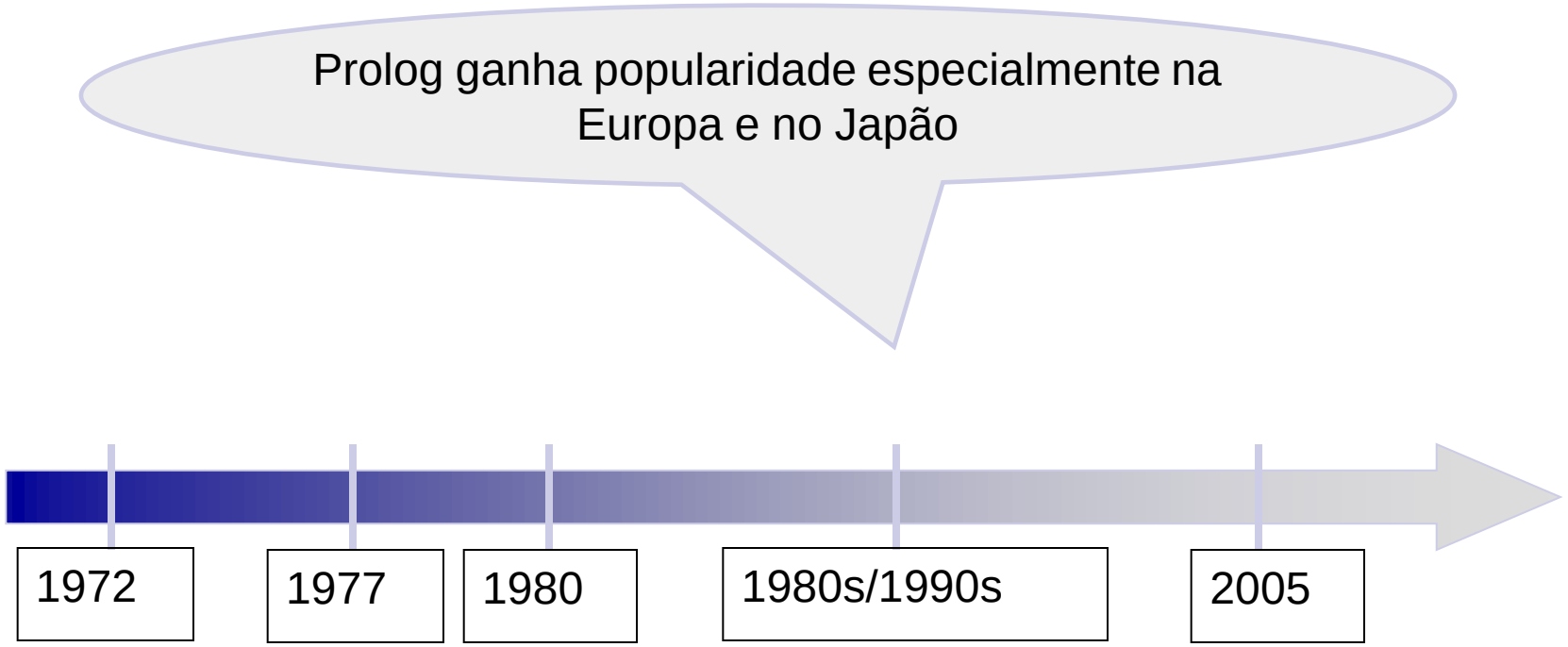


História do Prolog

implementação de
Gramáticas de Cláusulas Definidas
(Pereira e Warren)

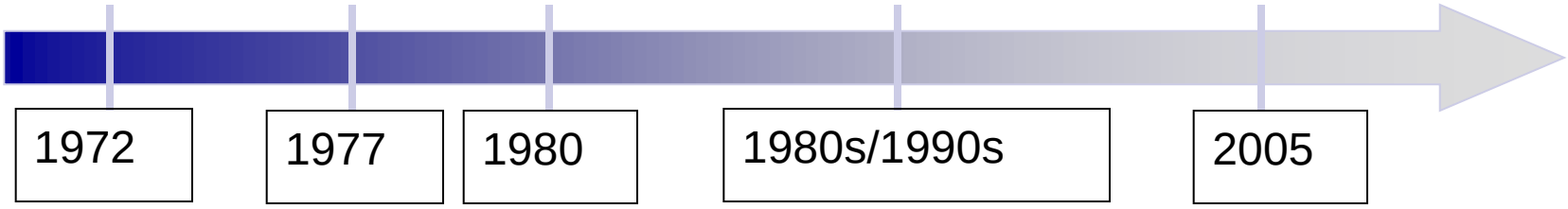


História do Prolog



História do Prolog

Prolog usado para programar interface de linguagen natural Estação Espacial International (NASA)



Ideia básica do Prolog

- Descrever o problema numa base de conhecimento em lógica
- Fazer interrogações
- O Prolog deduz novos factos que são consequência lógica
- O Prolog devolve as suas deduções como respostas às interrogações

Implicações

- Pensar declarativamente, não procedimentalmente
  - Difícil
  - Exige uma abordagem diferente
- Linguagem de alto-nível
  - Não tão eficiente como, por exemplo, C
  - Adequada para modelação rápida
  - Útil em muitas aplicações em IA

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?-
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?-
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?-
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
no
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- tattoed(jody).
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- tattoed(jody).
no
?-
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- tattoed(jody).
ERROR: predicate tattoed/1 not defined.
?-
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- party.
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- party.
yes
?-
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- rockConcert.
```

Base de Conhecimentos 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- rockConcert.
no
?-
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

Base de Conhecimentos 2

facto

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

facto

facto

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

facto

facto

regra

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

facto

facto

regra

regra

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

facto

facto

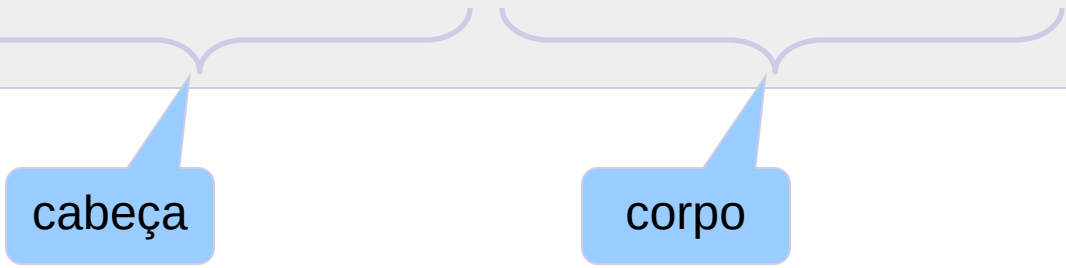
regra

regra

regra

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```



Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?-
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).
yes
?-
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).
yes
?- playsAirGuitar(yolanda).
yes
```

Cláusulas

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*Há 5 **cláusulas** nesta base de conhecimentos:*
2 factos e 3 regras.

O fim de uma cláusula é assinalado com um ponto.

Predicados

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*Há 3 **predicados** nesta base de conhecimentos:*
happy, listens2music, e playsAirGuitar

Base de Conhecimentos 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

Representação da Conjunção

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

A virgula "," representa a conjunção em Prolog

Base de Conhecimentos 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(vincent).
no
?-
```

Base de Conhecimentos 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(butch).
yes
?-
```

Representação da Disjunção

happy(vincent).

listens2music(butch).

playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).

playsAirGuitar(butch):- happy(butch).

playsAirGuitar(butch):- listens2music(butch).

happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch); listens2music(butch).

Prolog e Lógica

- Operadores
  - Implicação **:-**
  - Conjunção **,**
  - Disjunção **;**
- Uso do modus ponens
- Negação

Base de Conhecimentos 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

Variáveis Prolog

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
```

Instanciação de Variáveis

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia
```

Pedido de Alternativas

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
```

Pedido de Alternativas

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
X=jody
```

Pedido de Alternativas

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
X=jody;
X=yolanda
```

Pedido de Alternativas

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no
```

Base de Conhecimentos 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(marsellus,X), woman(X).
```

Base de Conhecimentos 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(marsellus,X), woman(X).
X=mia
yes
?-
```

Base de Conhecimentos 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(pumpkin,X), woman(X).
```

Base de Conhecimentos 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(pumpkin,X), woman(X).
no
?-
```

Base de Conhecimentos 5

```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

Base de Conhecimentos 5

```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).
```

Base de Conhecimentos 5

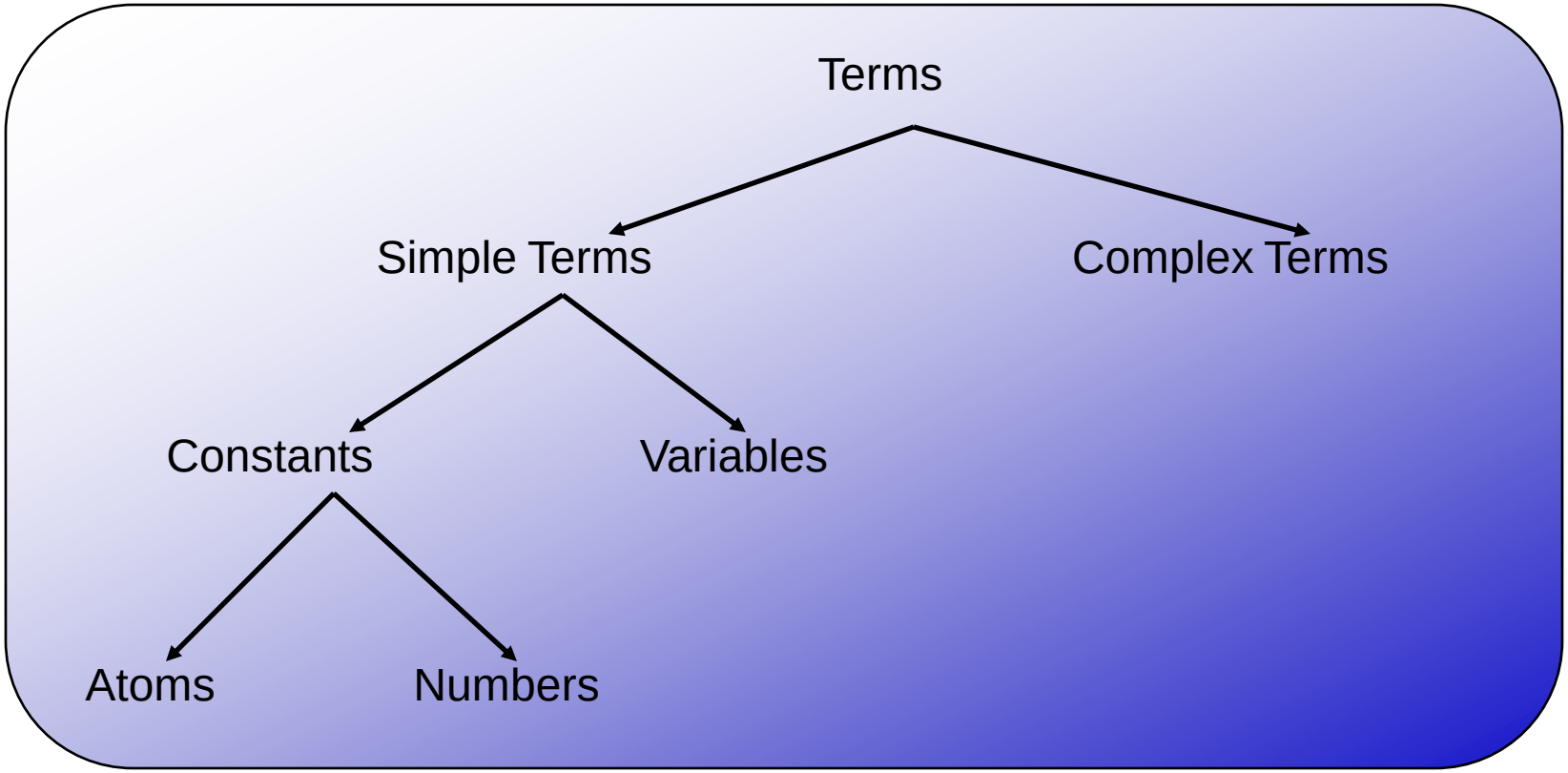
```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).
W=vincent
?-
```

Sintaxe do Prolog

- O que são os factos, regras e interrogações?



Átomos

- Uma sequência de caracteres de letras maiúsculas, minúsculas, números ou ‘underscore’, começando com uma letra minúscula
  - *Exemplos*: **butch**, **big_kahuna_burger**, **playGuitar**

- Uma sequência arbitrária de caracteres entre plicas
  - *Exemplos*: **'Vincent'**, **'Five dollar shake'**, **'@$%'**
- Uma sequência de caracteres especiais
  - *Exemplos*: **: , ; . :-**

Números

- Inteiros: 12, -34, 22342
- Reais: 34573.3234

Variáveis

- Uma sequência de caracteres de letras maiúsculas, minúsculas, números ou ‘underscore’, começando com uma letra maiúsculas ou ‘underscore’

- Exemplos:
  X, Y, Variable, Vincent, _tag

Termos Complexos

- Átomos, números e variáveis são a base dos termos complexos
- Os termos complexos são construídos a partir de um functor seguido de uma sequência de argumentos
- Os argumentos são colocados entre parênteses, separados por vírgulas
- O functor tem que ser um átomo

Exemplos de termos complexos

- Exemplos anteriores:
  - playsAirGuitar(jody)
  - loves(vincent, mia)
  - jealous(marsellus, W)
- Termos complexos dentro de termos complexos :
  - hide(X,father(father(father(butch))))

Aridade

- Ao número de argumentos de um termo complexo chama-se a sua aridade

- Exemplos:

woman(mia) é um termo com aridade 1
loves(vincent,mia) tem aridade 2
father(father(butch)) aridade 1

A aridade é importante

- Podem-se definir dois predicados com o mesmo functor mas com aridade diferente
- Neste caso, são dois predicados diferentes
- A aridade de um predicado é geralmente indicada com o sufixo "/" seguido de um número para indicar a aridade

Exemplo de Aridade

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

- Esta base de conhecimento define
  - happy/1
  - listens2music/1
  - playsAirGuitar/1

Unificação

- O Prolog unifica

  woman(X)

  com

  woman(mia)

  instanciando a variável **X** com o átomo **mia**.

Unificação

- Definição:
  - Dois termos unificam se forem iguais, ou se contêm variáveis que podem ser instanciadas de maneira a que os termos resultantes fiquem iguais

Unificação

- Significa que:
  - **mia** e **mia** unificam
  - **42** e **42** unificam
  - **woman(mia)** e **woman(mia)** unificam
- E que:
  - **vincent** e **mia** não unificam
  - **woman(mia)** e **woman(jody)** não unificam

Unificação

- E os termos:
  - **mia** e **X**

Unificação

- E os termos:
  - **mia** e **X**
  - **woman(Z)** e **woman(mia)**

Unificação

- E os termos:
  - **mia** e **X**
  - **woman(Z)** e **woman(mia)**
  - **loves(mia,X)** e **loves(X,vincent)**

Instanciações

- Quando o Prolog unifica dois termos faz todas as instanciações necessárias de modo que os termos fiquem iguais

Unificação em Prolog

1. Se T_1 e T_2 forem constantes, então
   T_1 e T_2 unificam se forem o mesmo átomo, ou
   o mesmo número.

Unificação em Prolog

1. Se T_1 e T_2 forem constantes, então T_1 e T_2 unificam se forem o mesmo átomo, ou o mesmo número.
2. Se T_1 for uma variável e T_2 um termo, então T_1 e T_2 unificam, e T_1 é instanciado com T_2. (e vice versa)

Unificação em Prolog

1. Se T_1 e T_2 forem constantes, então T_1 e T_2 unificam se forem o mesmo átomo, ou o mesmo número.
2. Se T_1 for uma variável e T_2 um termo, então T_1 e T_2 unificam, e T_1 é instanciado com T_2. (e vice versa)
3. Se T_1 e T_2 são termos complexos, unificam se:
   a) Tiverem o mesmo functor e aridade, e
   b) Todos os argumentos correspondentes unificam, e
   c) As instanciações das variáveis são compatíveis.

Unificação em Prolog: =/2

?- mia = mia.

yes

?-

Unificação em Prolog: =/2

?- mia = mia.

yes

?- mia = vincent.

no

?-

Unificação em Prolog: =/2

```
?- mia = X.
X=mia
yes
?-
```

Qual será a resposta?

?- X=mia, X=vincent.

Qual será a resposta?

?- X=mia, X=vincent.

no

?-

Porquê? Ao passar pelo primeiro objectivo, instancia X com **mia**, por isso já não pode unificar com **vincent**. Logo o segundo objectivo falha.

Exemplo com termos complexos

?- k(s(g),Y) = k(X,t(k)).

Exemplo com termos complexos

?- k(s(g),Y) = k(X,t(k)).

X=s(g)

Y=t(k)

yes

?-

Exemplo com termos complexos

?- k(s(g),t(k)) = k(X,t(Y)).

Exemplo com termos complexos

?- k(s(g),t(k)) = k(X,t(Y)).

X=s(g)

Y=k

yes

?-

Mais um exemplo

?- loves(X,X) = loves(marsellus,mia).

Prolog e Unificação

- O Prolog não usa o algoritmo standard de unificação
- Exemplo:

  ?- father(X) = X.

- Os termos unificam ou não?

Termos infinitos

?- father(X) = X.

X=father(

Termos infinitos

?- father(X) = X.

X=father(father(father(…))))

yes

?-

Occurs Check

- O algoritmo standard de unificação executa um occurs check
- Se se quiser unificar uma variável com outro termo verifica se a variável ocorre nesse termo
- Em Prolog:

```
?- unify_with_occurs_check(father(X), X).
no
```

Programação com Unificação

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

Programação com Unificação

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?-
```

Programação com Unificação

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?- vertical(line(point(1,1),point(1,3))).
yes
?-
```

Programação com Unificação

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?- vertical(line(point(1,1),point(1,3))).
yes
?- vertical(line(point(1,1),point(3,2))).
no
?-
```

Programação com Unificação

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?- horizontal(line(point(1,1),point(1,Y))).
Y = 1;
no
?-
```

Programação com Unificação

```
vertical( line(point(X,Y),
               point(X,Z))).

horizontal( line(point(X,Y),
                 point(Z,Y))).
```

```
?- horizontal(line(point(2,3),Point)).
Point = point(_554,3);
no
?-
```

Pesquisa em Prolog

- Sabendo o processo de unificação, podemos agora aprender como o Prolog pesquisa a base de conhecimentos para responder às interrogações.

Exemplo

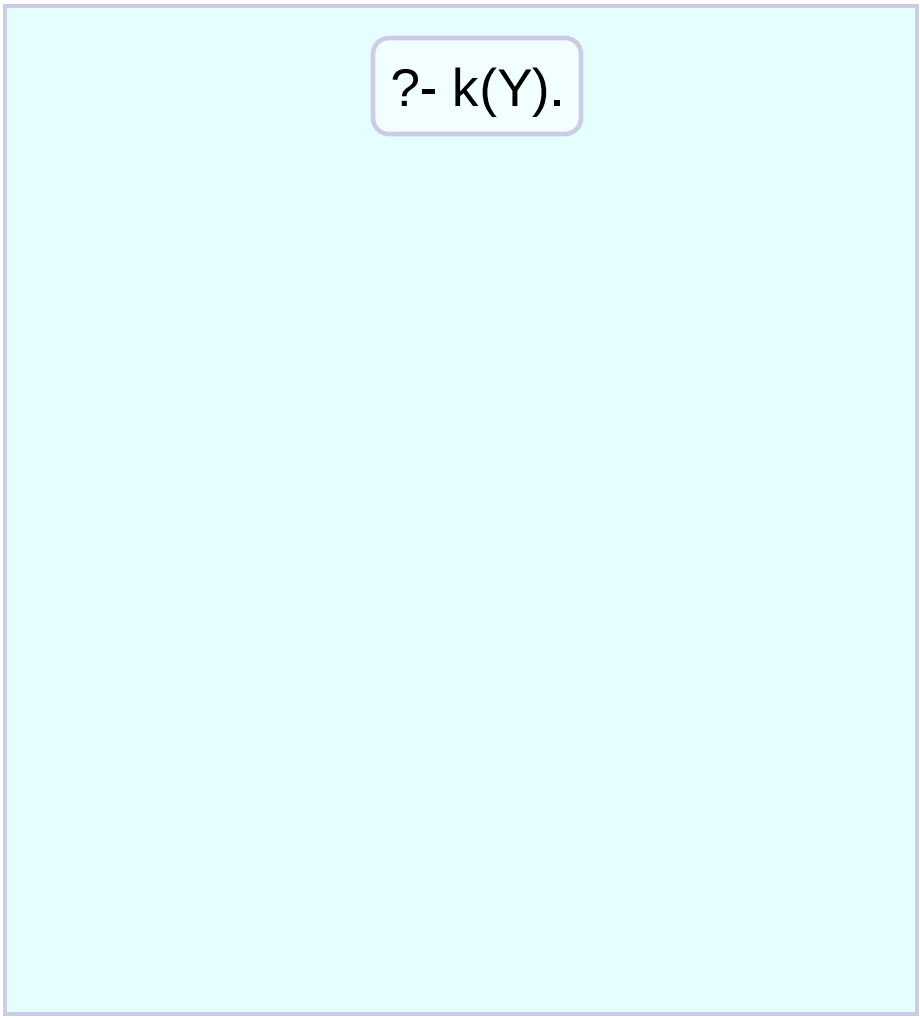
```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
```

Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

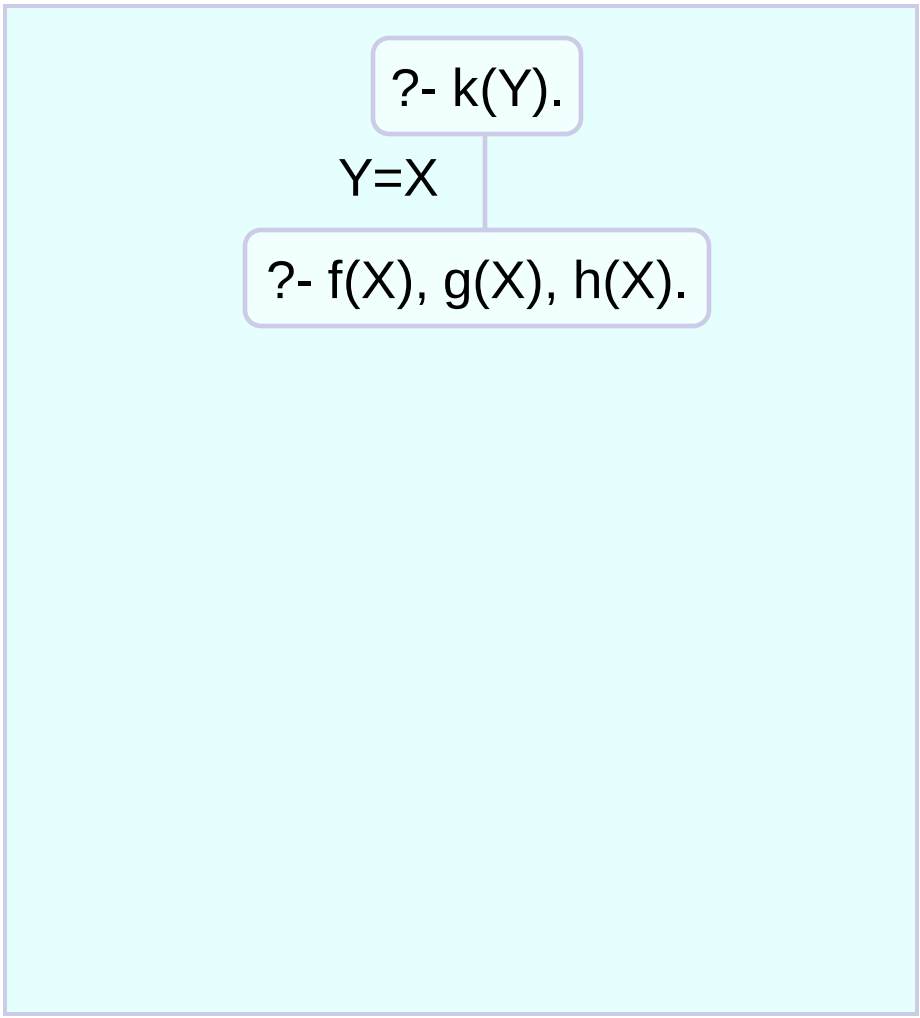
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

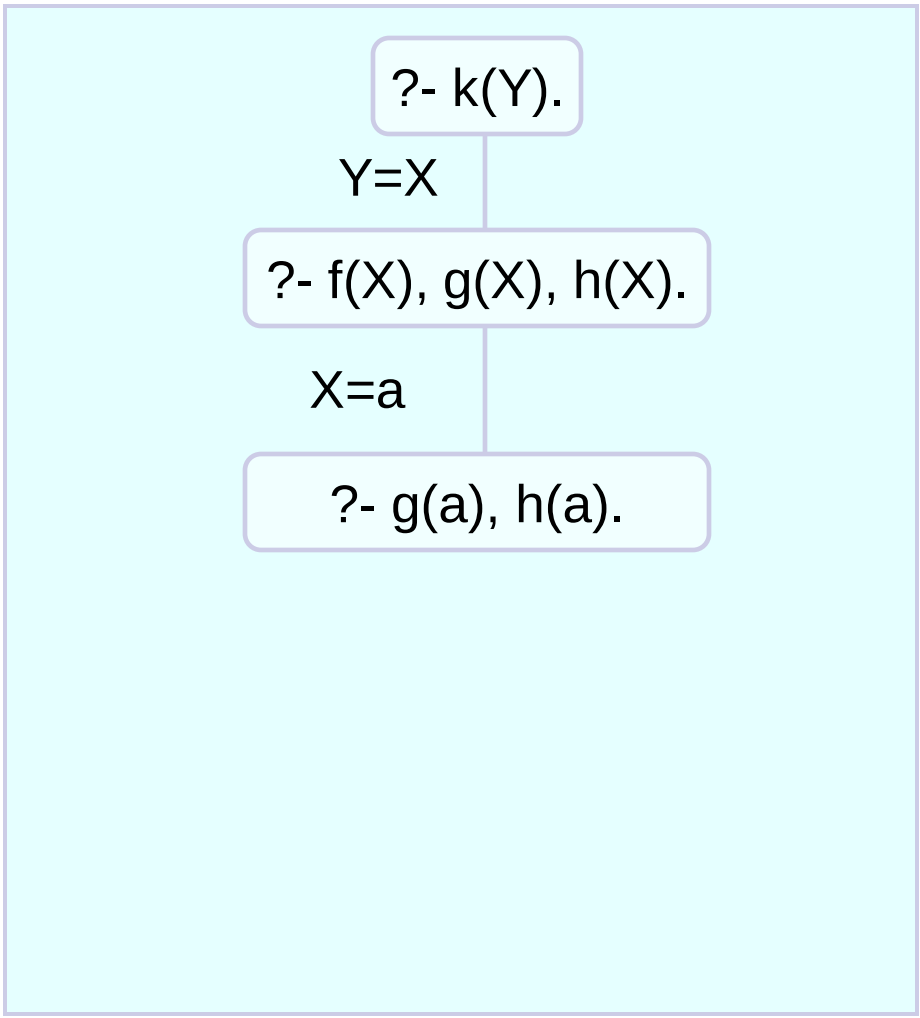
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

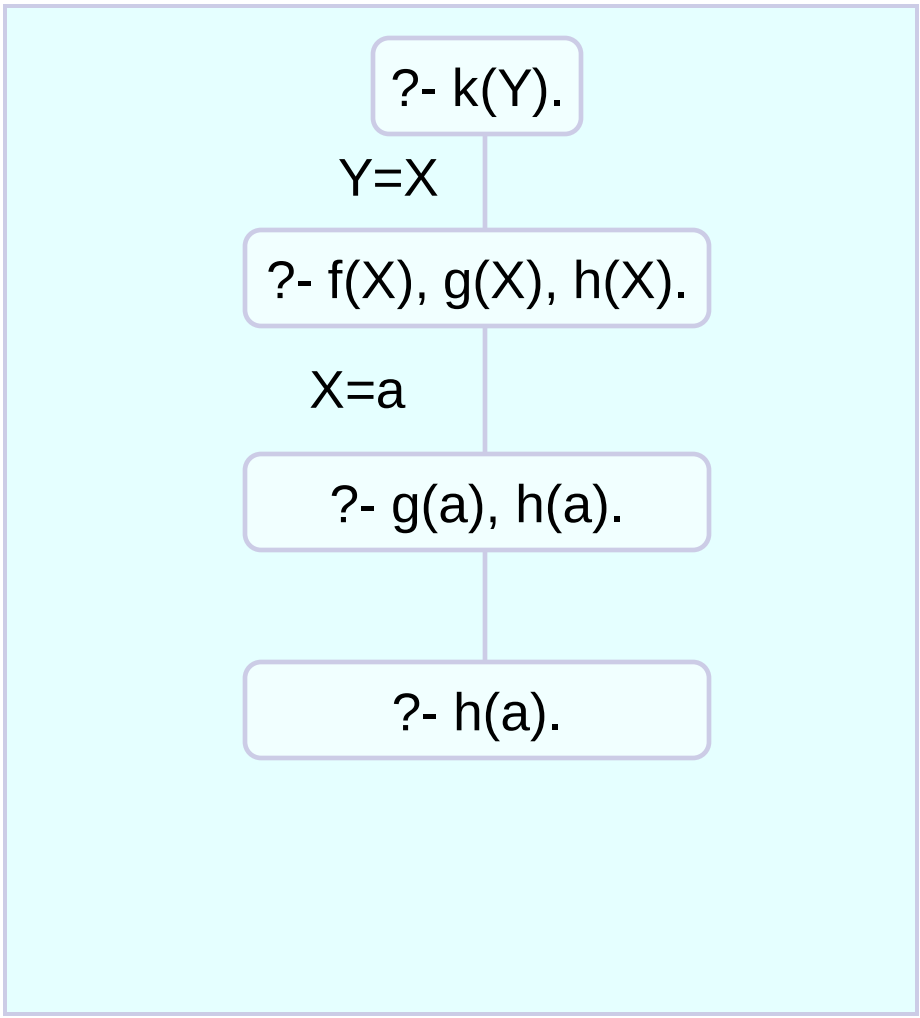
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

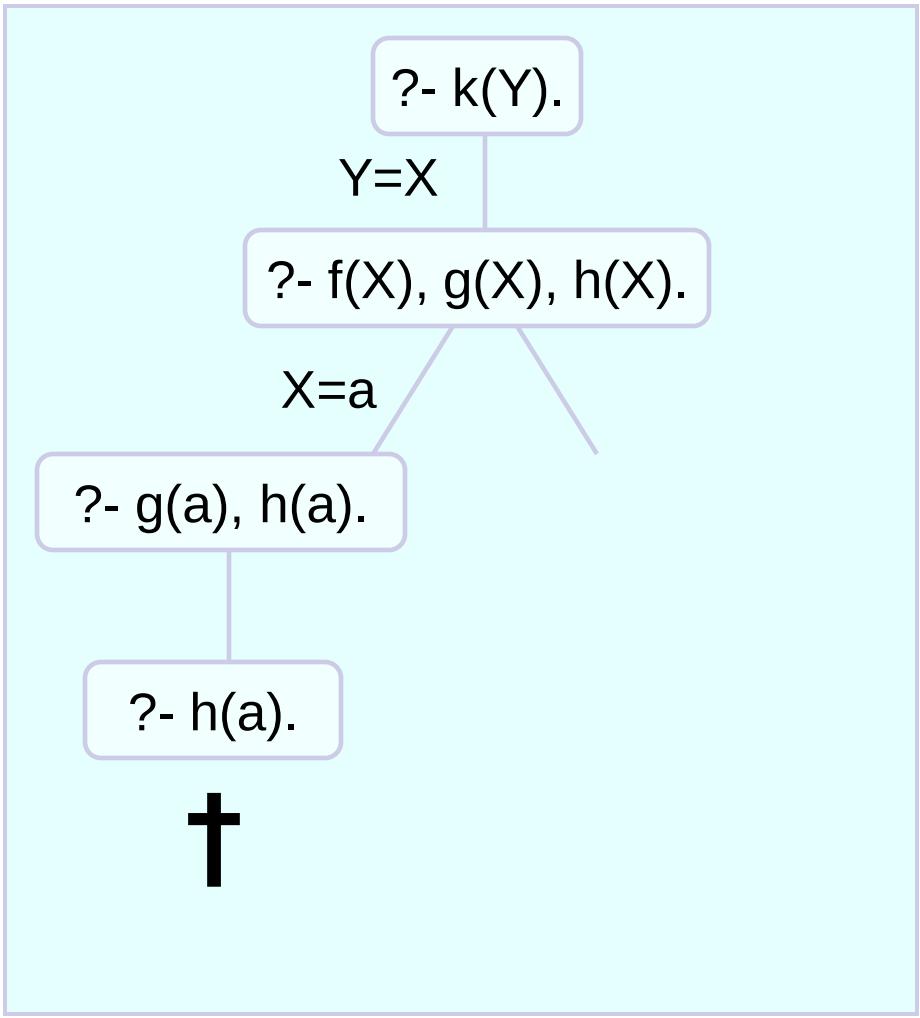
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

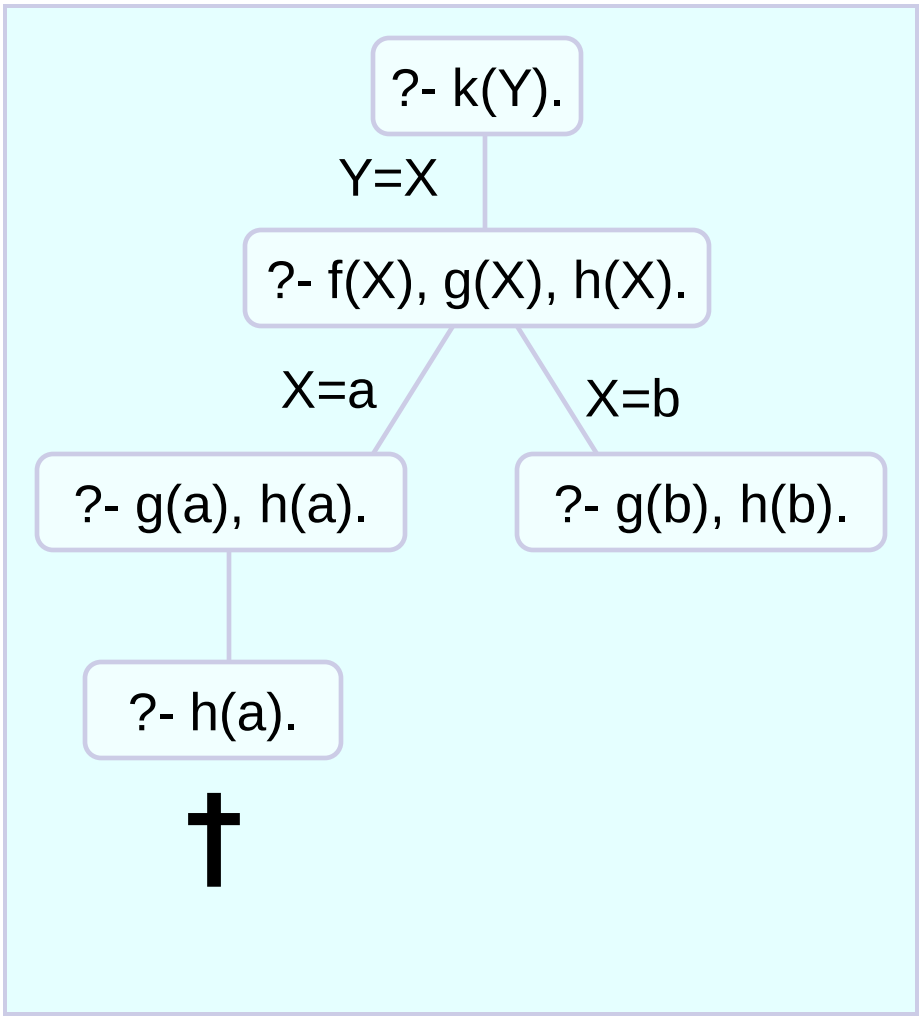
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

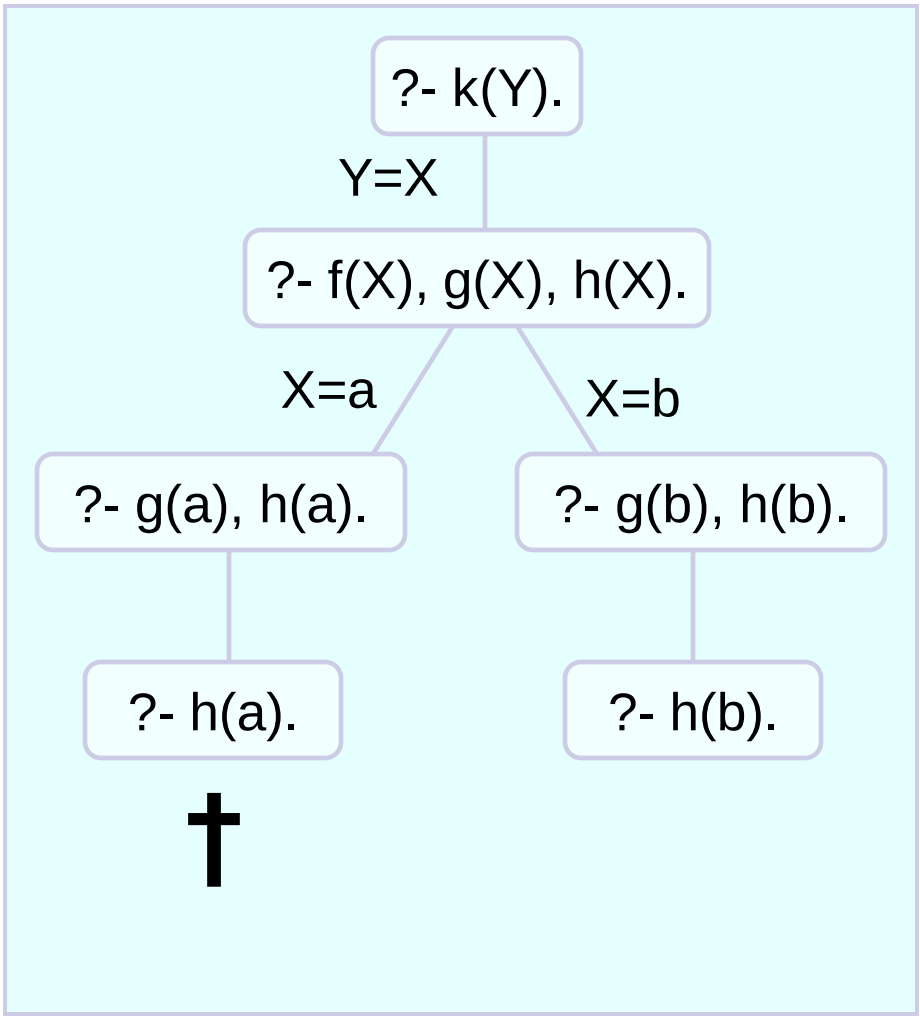
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

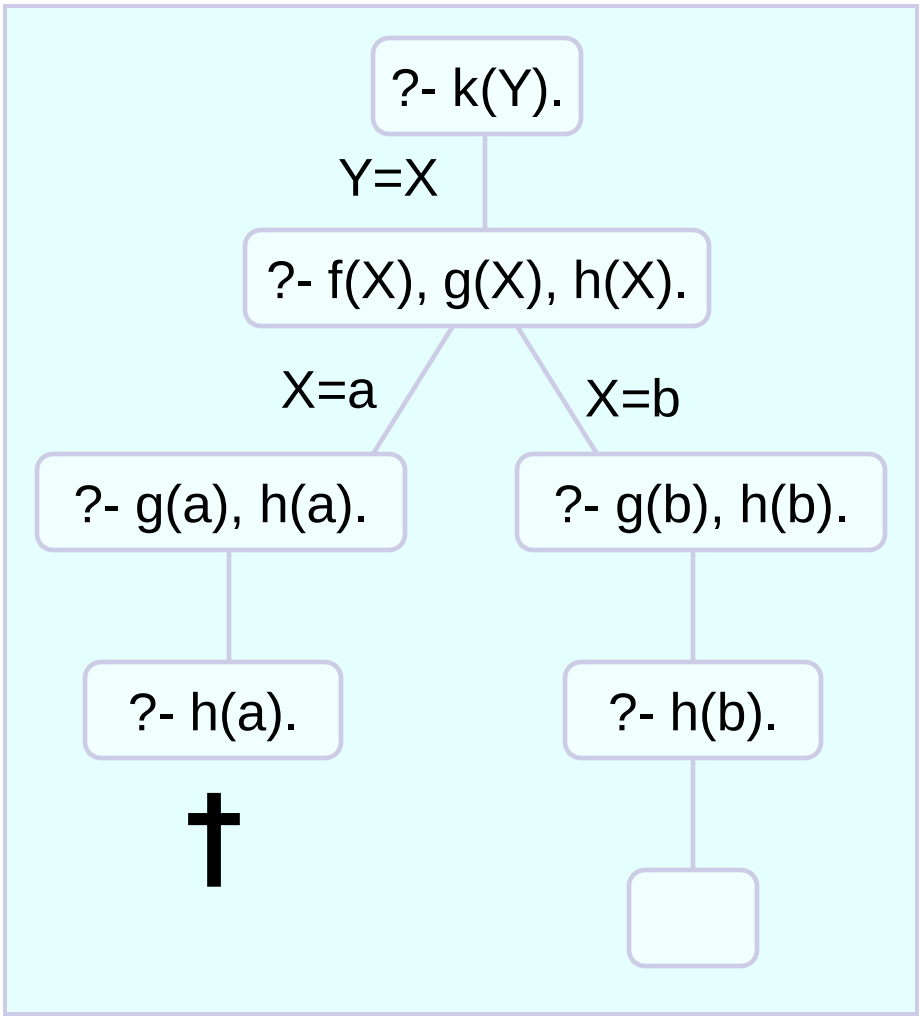
```
?- k(Y).
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

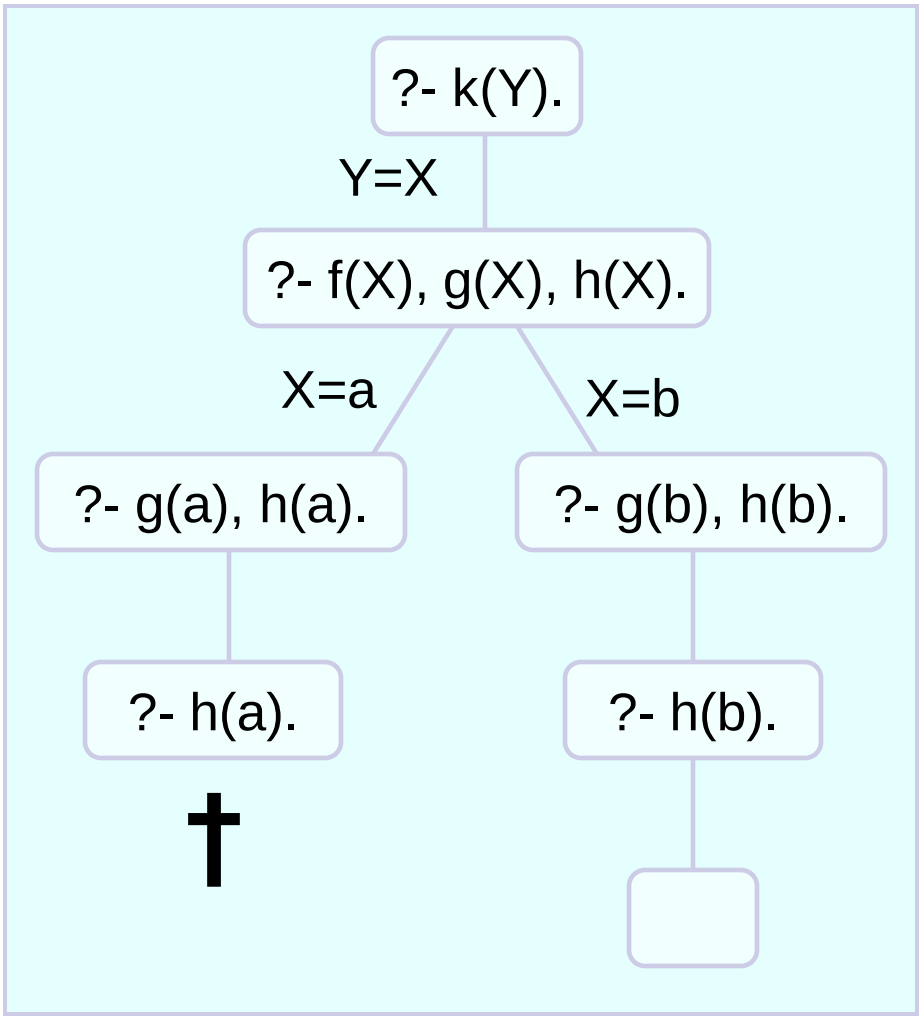
```
?- k(Y).
Y=b
```



Exemplo: árvore de pesquisa

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
Y=b;
no
?-
```



Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

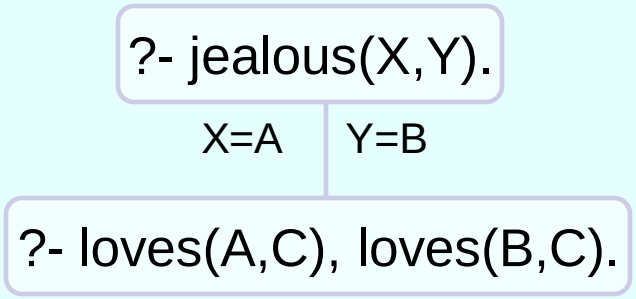
```
?- jealous(X,Y).
```

Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

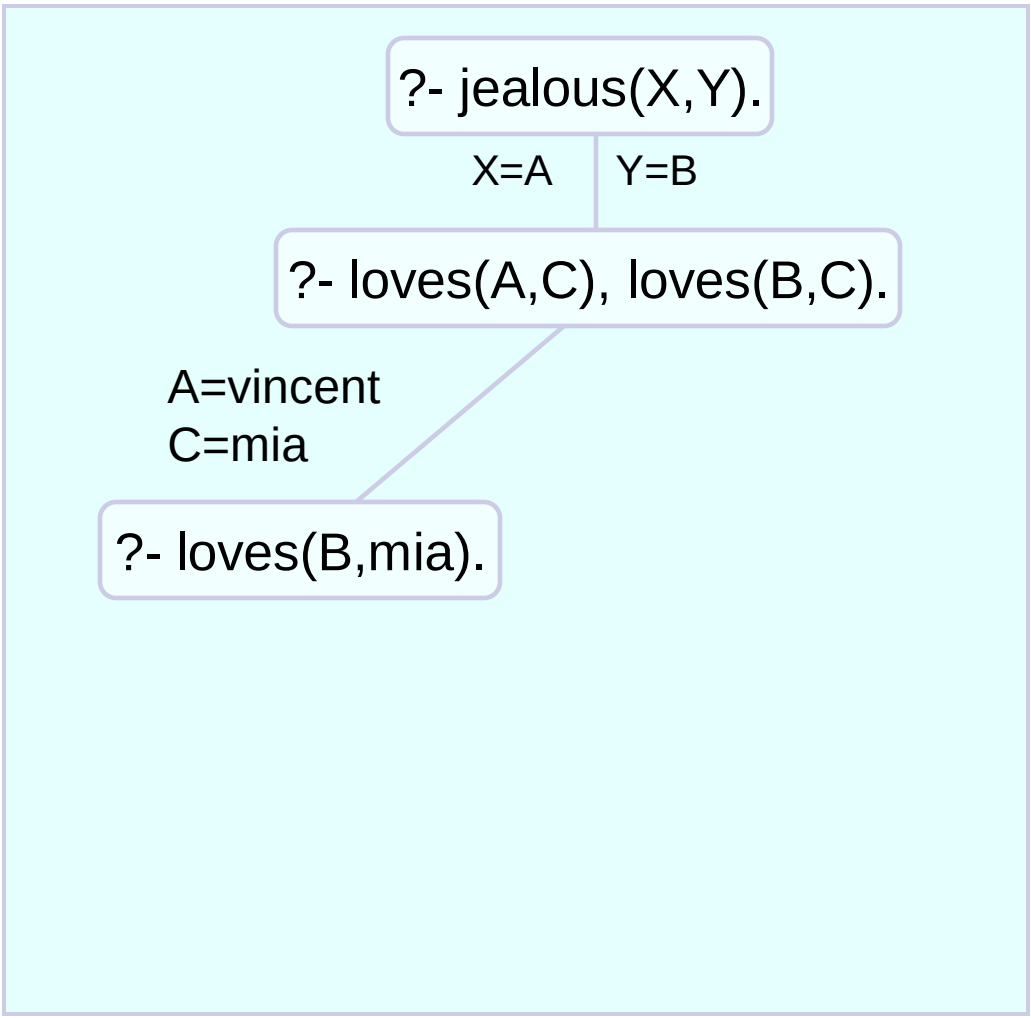


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
```

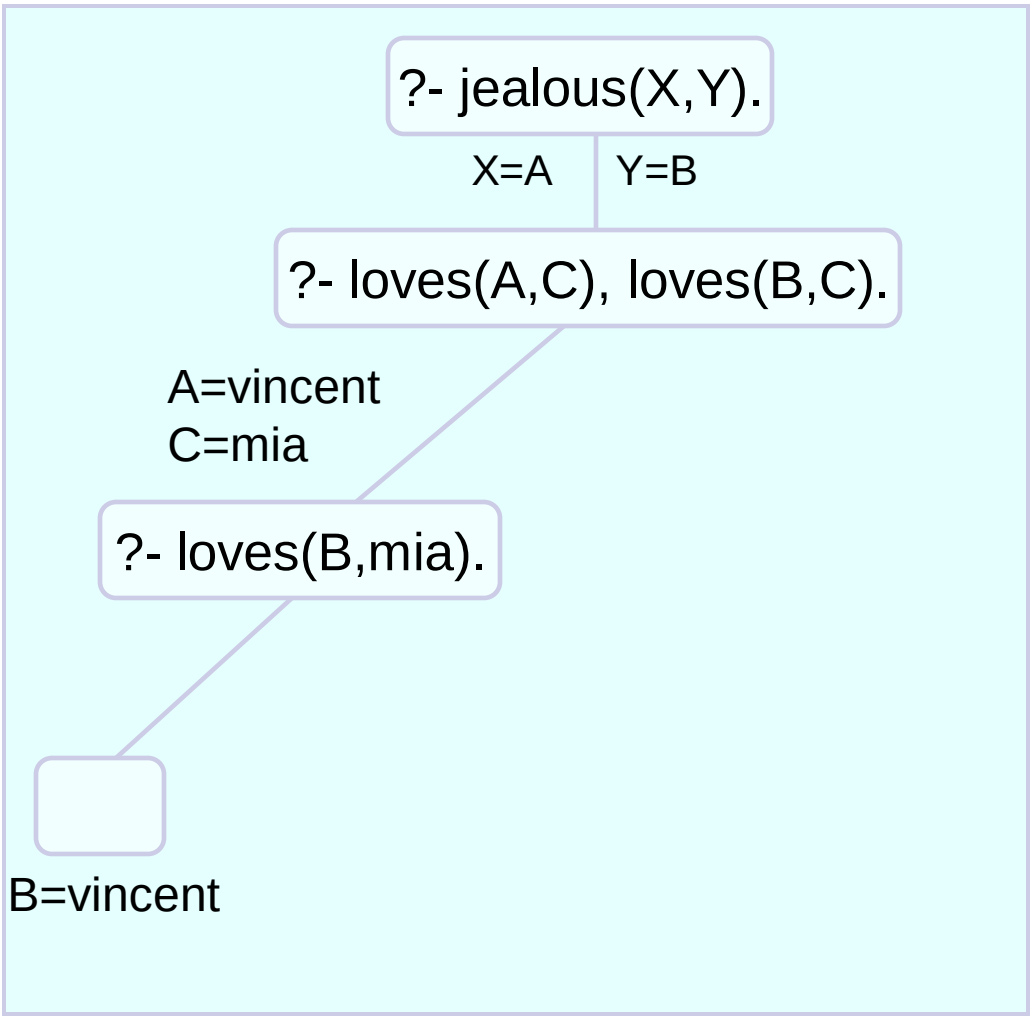


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
X=vincent
Y=vincent
```

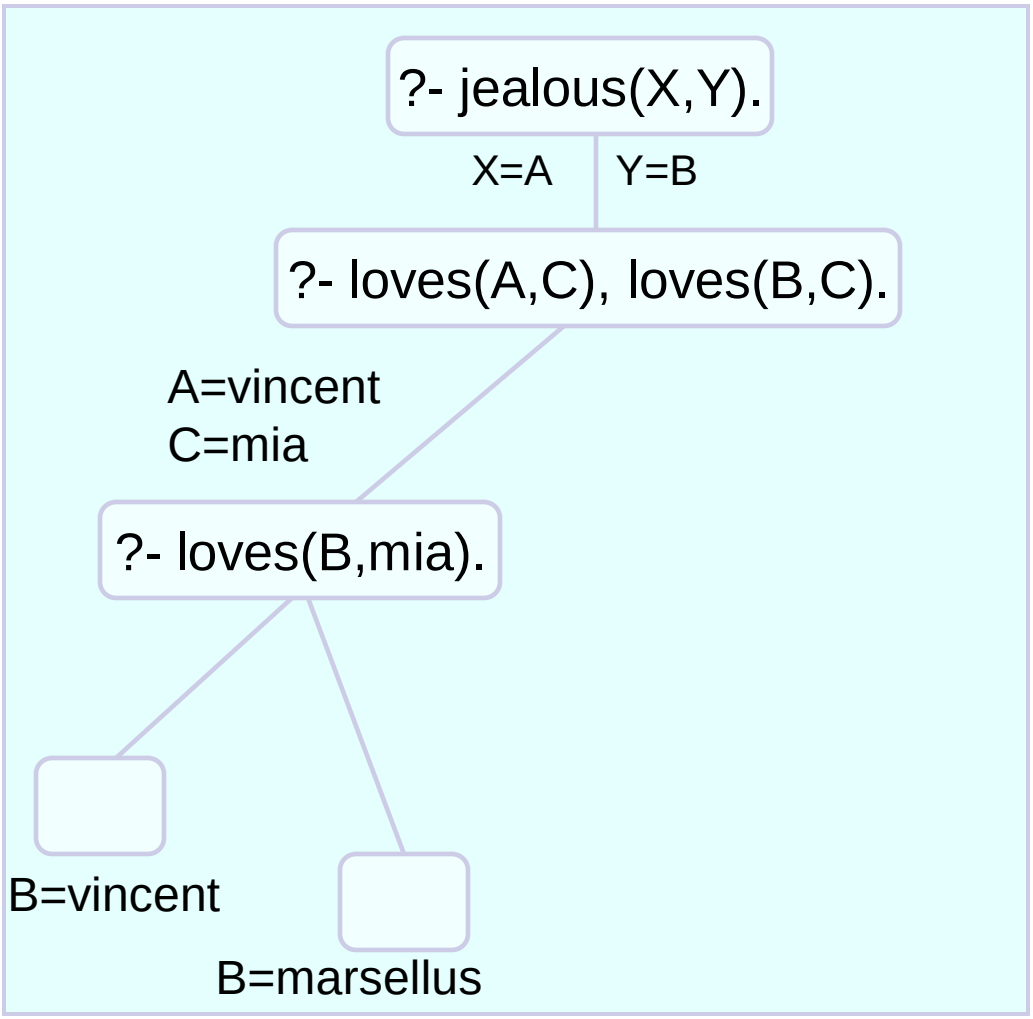


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
X=vincent
Y=vincent;
X=vincent
Y=marsellus
```

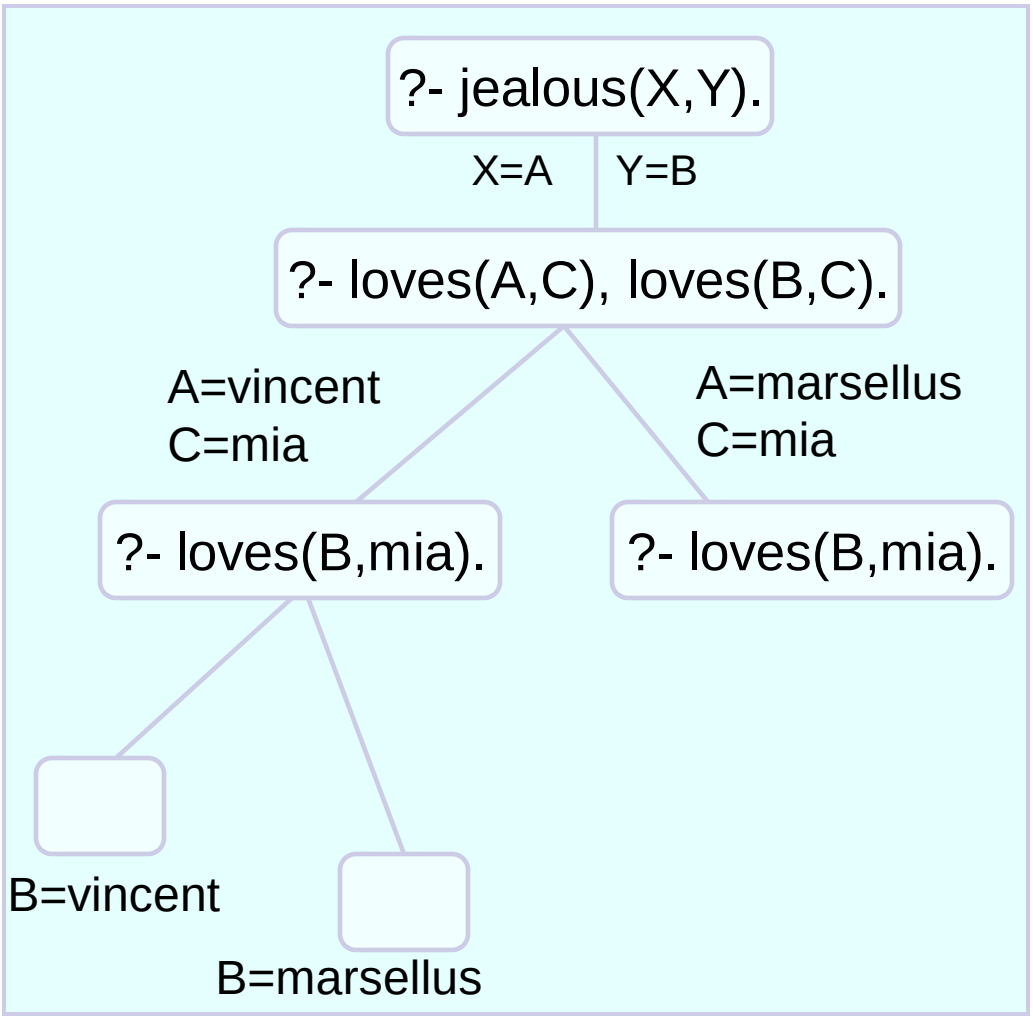


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
?- jealous(X,Y).
X=vincent
Y=vincent;
X=vincent
Y=marsellus;
```

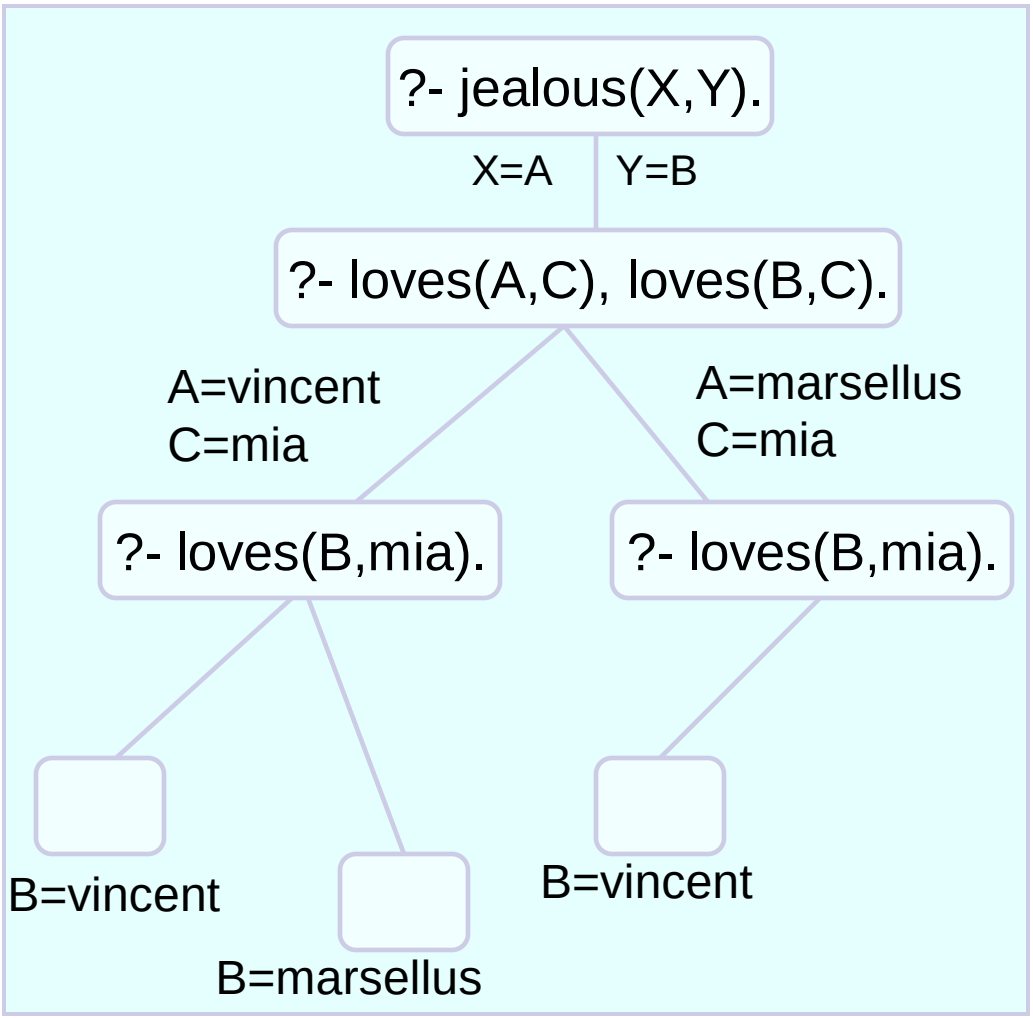


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
….
X=vincent
Y=marsellus;
X=marsellus
Y=vincent
```

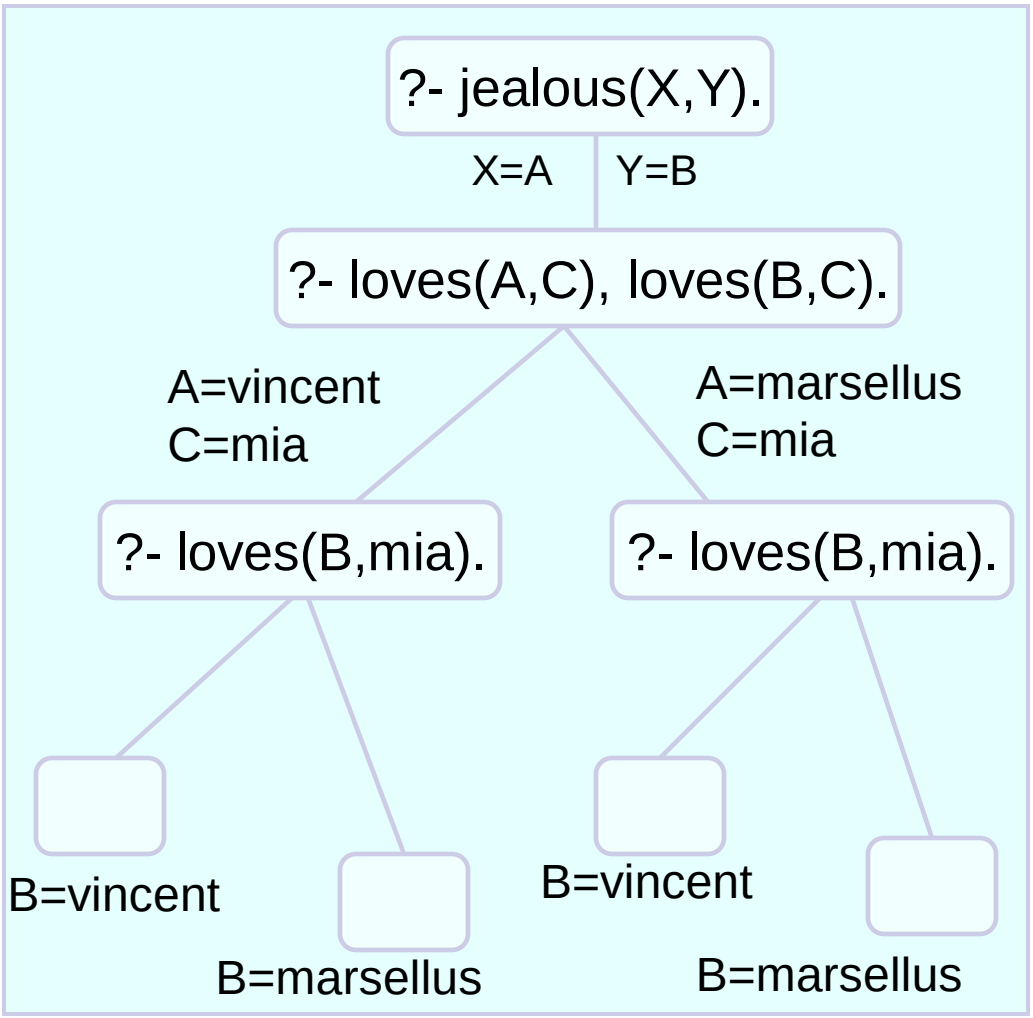


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
….
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus
```

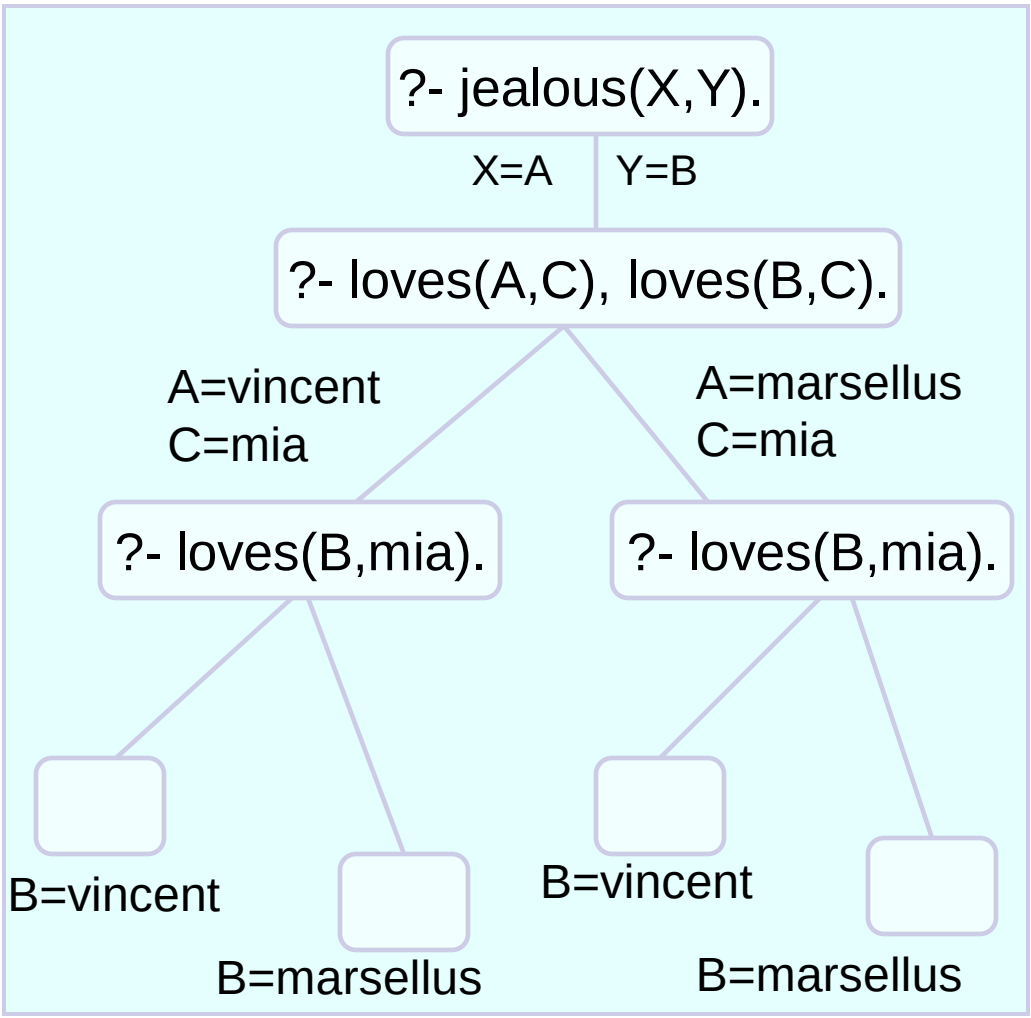


Outro exemplo

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
    loves(A,C),
    loves(B,C).
```

```
….
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus;
no
```



Definições Recursivas

- Os predicados podem ser definidos recursivamente em Prolog
- Um predicado é definido recursivamente se uma ou mais regras da sua definição se referem a si próprias

Exemplo 1: Comida

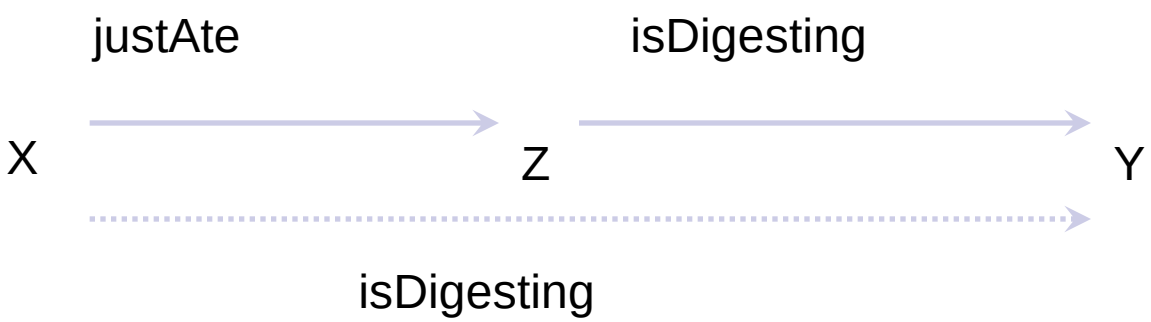
```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

justAte(mosquito,blood(john)).
justAte(frog,mosquito).
justAte(stork,frog).
```

```
?-
```

Retrato da situação

Retrato da situação



Exemplo 1: Comida

```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

justAte(mosquito,blood(john)).
justAte(frog,mosquito).
justAte(stork,frog).
```

```
?- isDigesting(stork,mosquito).
```

Outra definição recursiva

```
p:- p.
```

```
?-
```

Outra definição recursiva

```
p:- p.
```

```
?- p.
```

Outra definição recursiva

```
p:- p.
```

```
?- p.
ERROR: out of memory
```

Exemplo 2: Descendência

```
child(bridget,caroline).
child(caroline,donna).

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

Exemplo 2: Descendência

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


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

Exemplo 2: Descendência

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).



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

```
?- descend(anna,donna).
no
?-
```

Exemplo 2: Descendência

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


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

```
?-
```

Exemplo 2: Descendência

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


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

```
?-
```

Exemplo 2: Descendência

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


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

```
?- descend(anna,donna).
```

Exemplo 3: Sucessor

- Considere-se a seguinte definição de números naturais:
  1. **0** é um natural.
  2. Se **X** é um natural, então **succ(X)** também.

Exemplo 3: Sucessor

```
natural(0).
natural(succ(X)):- natural(X).
```

Exemplo 3: Sucessor

```
natural(0).
natural(succ(X)):- natural(X).
```

```
?- natural(succ(succ(succ(0)))).
yes
?-
```

Exemplo 3: Sucessor

```
natural(0).
natural(succ(X)):- natural(X).
```

```
?- natural(X).
```

Exemplo 3: Sucessor

```
natural(0).
natural(succ(X)):- natural(X).
```

```
?- natural(X).
X=0;
X=succ(0);
X=succ(succ(0));
X=succ(succ(succ(0)));
X=succ(succ(succ(succ(0))))
```

Exemplo 4: Adição

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Exemplo 4: Adição

```
add(0,X,X).                          %%% base clause
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Exemplo 4: Adição

```
add(0,X,X).                          %%% base clause

add(succ(X),Y,succ(Z)):-            %%% recursive clause
      add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Prolog e Lógica

- O Prolog foi a primeira tentativa de criar uma linguagem de programação em lógica
  - O programador fornece uma especificação declarativa do problema usando uma linguagem lógica
  - O programador não tem que dizer o que o computador tem que fazer
  - Para obter informação, o programador apenas formula uma interrogação

Prolog e Lógica

- O Prolog dá alguns passos importantes nesta direcção, mas não é uma pura linguagem de programação em lógica!
- O Prolog tem uma forma específica para responder às interrogações:
  - Pesquisa a base de conhecimentos de cima para baixo
  - Processa as cláusulas da esquerda para a direita
  - Retrocede para refazer escolhas alternativas

descend1.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

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

```
?- descend(A,B).
A=anna
B=bridget
```

descend2.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

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

```
?- descend(A,B).
A=anna
B=emily
```

descend3.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

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

```
?- descend(A,B).
ERROR: OUT OF LOCAL STACK
```

descend4.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

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

```
?- descend(A,B).
```