Aula 2

- Teoria
  - Unificação
  - Unificação em Prolog
  - Busca pela prova

Objetivos desta aula

- Discutir **unificação** em Prolog
  - Mostrar como a unificação em Prolog difere da unificação padrão
- Explicar a estratégia de busca que o Prolog utiliza quando ele tenta deduzir novas informações a partir de outras informações, usando *modus ponens*

Unificação

- Relembrando o exemplo visto na aula anterior, onde foi dito que Prolog unifica

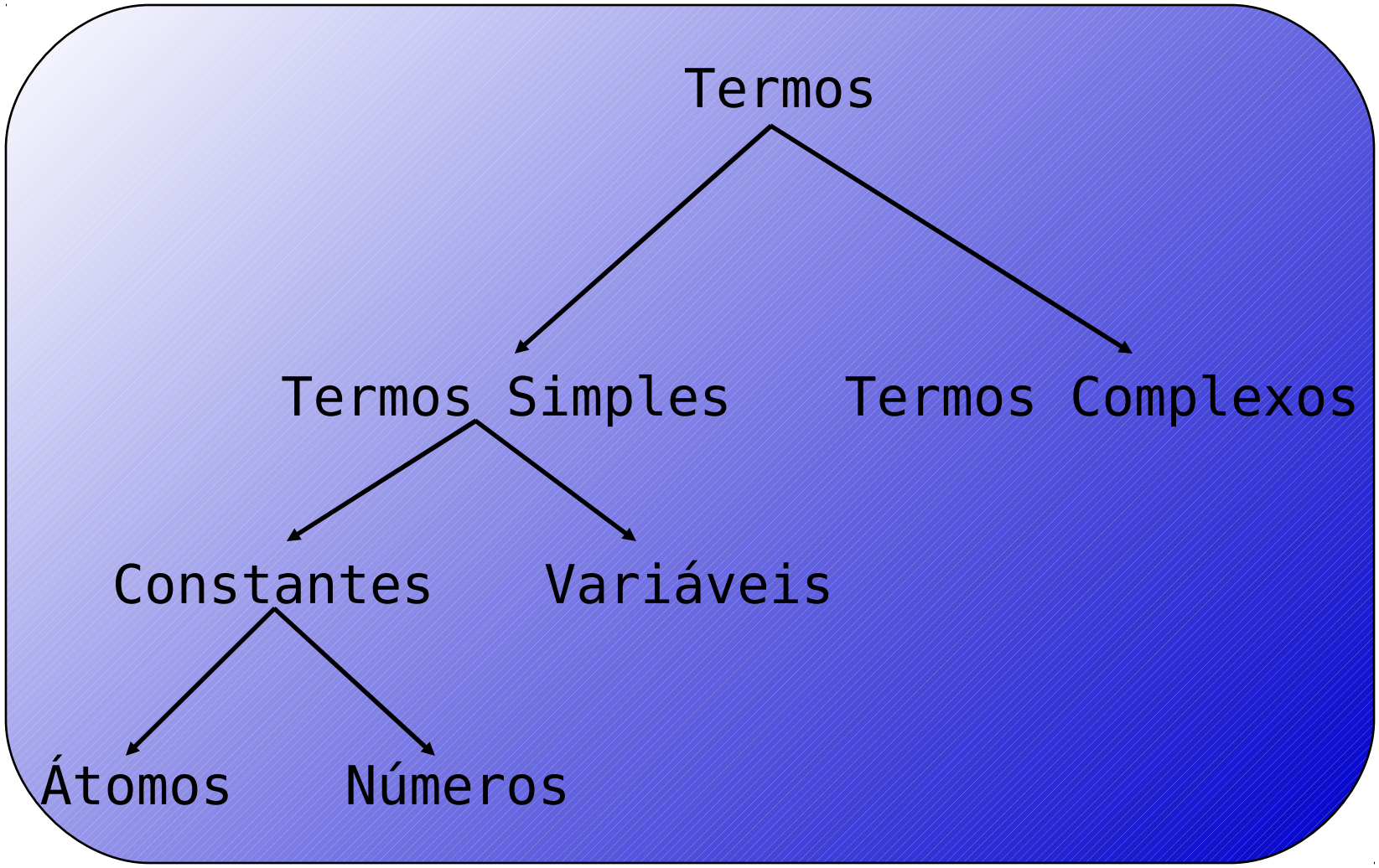
  mulher(X)

  com

  mulher(maria)

  instanciando assim a variável **X** com o átomo **maria**.

Revisão dos termos em Prolog



Unificação

- Definição operacional:
  - Dois termos se unificam se eles são o mesmo termo ou se eles contém variáveis que podem ser uniformemente instanciadas com termos de tal forma que os termos resultantes sejam iguais.

Unificação

- Isto significa que:
  - **maria** e **maria** se unificam
  - **42** e **42** se unificam
  - **mulher(maria)** e **mulher(maria)** se unificam
- Isto também significa que:
  - **vicente** e **maria** não se unificam
  - **mulher(maria)** e **mulher(joana)** não se unificam

Unificação

- Os termos abaixo se unificam?
  - **maria** e **X**

Unificação

- Os termos abaixo se unificam?
  - **maria** e **X**
  - **mulher(Z)** e **mulher(maria)**

Unificação

- Os termos abaixo se unificam?
  - **maria** e **X**
  - **mulher(Z)** e **mulher(maria)**
  - **ama(maria,X)** e **ama(X,vicente)**

Instanciações

- Quando Prolog unifica dois termos ele realiza todas as instaciações necessárias, tal que os termos se igualem deste ponto em diante.
- Isto torna a unificação um mecanismo poderoso de programação.

Definição Revisada 1/3

- Se T_1 e T_2 são constantes, então T_1 e T_2u se unificam se eles são os mesmos átomos ou o mesmo número.

Definição Revisada 2/3

- Se T_1 e T_2 são constantes, então T_1 e T_2 se unificam se eles são os mesmos átomos ou o mesmo número.
- Se T_1 é uma variável e T_2 é um tipo qualquer de temo, então T_1 e T_2 se unificam e T_1 é instanciado para T_2 (e vice-versa).

Definição Revisada 3/3

- Se T_1 e T_2 são constantes, então T_1 e T_2 se unificam se eles são os mesmos átomos ou o mesmo número.
- Se T_1 é uma variável e T_2 é um tipo qualquer de temo, então T_1 e T_2 se unificam e T_1 é instanciado para T_2 (e vice-versa).
- Se T_1 e T_2 são termos complexos, então eles se unificam se:
  - Eles possuem o mesmo funtor e a mesma aridade, e
  - Todos os seus argumentos correspondentes se unificam e
  - As instanciações das variáveis são compatíveis.

Unificação em Prolog: =/2

```
?- maria = maria.
true
?-
```

Unificação em Prolog: =/2

```
?- maria = maria.
true
?- maria = vicente.
false
?-
```

Unificação em Prolog: =/2

```
?- maria = X.
X=maria
true
?-
```

Como Prolog responderá?

?- X=maria, X=vicente.

Como Prolog responderá?

?- X=maria, X=vicente.

false

?-

Por quê? Após trabalhar na primeira meta, Prolog instanciou a variável X com **maria**, e assim ele não pode mais unificá-la com **vicente**. Logo, a segunda meta 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)
true
?-
```

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
true
?-
```

Um último exemplo

?- ama(X,X) = ama(marcelo,maria).

Um último exemplo

?- ama(X,X) = ama(marcelo,maria).

false

Prolog e unificação

- Prolog não usa um algoritmo de unficação padrão
- Considere a seguinte consulta:

  ?- pai(X) = X.

- Estes termos se unificam ou não?

Termos infinitos

?- pai(X) = X.

X=pai(pai(pai(pai(pai(pai(pai(pai(pai(pai(
pai(pai(pai(pai(pai(pai(pai(pai(pai(pai(p
ai(pai(pai(pai(pai(pai(pai(pai(pai(pai(pai
(pai(pai(pai(pai(pai(pai(pai(pai(pai(pai(p
ai(pai(pai(pai(pai(pai(

Termos infinitos

```
?- pai(X) = X.
X=pai(pai(pai(…))))
true
?-
```

Verificação de ocorrência

- Um algoritmo de unificação padrão realiza uma verificação de ocorrência
- Se a ele for pedido para unificar uma variável com um outro termo, ele verificará se a variável ocorre no termo
- Em Prolog:

```
?- unify_with_occurs_check(pai(X), X).
false
```

Programando com Unificação

```
vertical( linha(ponto(X,Y),
                ponto(X,Z))).

horizontal( linha(ponto(X,Y),
                  ponto(Z,Y))).
```

Programando com Unificação

```
vertical( linha(ponto(X,Y),
                ponto(X,Z))).

horizontal( linha(ponto(X,Y),
                  ponto(Z,Y))).
```

```
?-
```

Programando com Unificação

```
vertical( linha(ponto(X,Y),
                ponto(X,Z))).

horizontal( linha(ponto(X,Y),
                  ponto(Z,Y))).
```

```
?- vertical(linha(ponto(1,1),ponto(1,3))).
true
?-
```

Programando com Unificação

```
vertical( linha(ponto(X,Y),
                ponto(X,Z))).

horizontal( linha(ponto(X,Y),
                  ponto(Z,Y))).
```

```
?- vertical(linha(ponto(1,1),ponto(1,3))).
true
?- vertical(linha(ponto(1,1),ponto(3,2))).
false
?-
```

Programando com Unificação

```
vertical( linha(ponto(X,Y),
                ponto(X,Z))).

horizontal( linha(ponto(X,Y),
                  ponto(Z,Y))).
```

```
?- horizontal(linha(ponto(1,1),ponto(1,Y))).
Y = 1;
false
?-
```

Programando com Unificação

```
vertical( linha(ponto(X,Y),
                ponto(X,Z))).

horizontal( linha(ponto(X,Y),
                  ponto(Z,Y))).
```

```
?- horizontal(linha(ponto(2,3),Ponto)).
Ponto = ponto(_G554,3);
false
?-
```

Exercício: unificação

Busca pela prova

- Agora que conhecemos a unificação, começaremos a aprender como Prolog busca em base de conhecimento para ver se uma consulta é satisfeita.
- Em outras palavras: nós começaremos a aprender sobre a busca pela prova

Exemplo

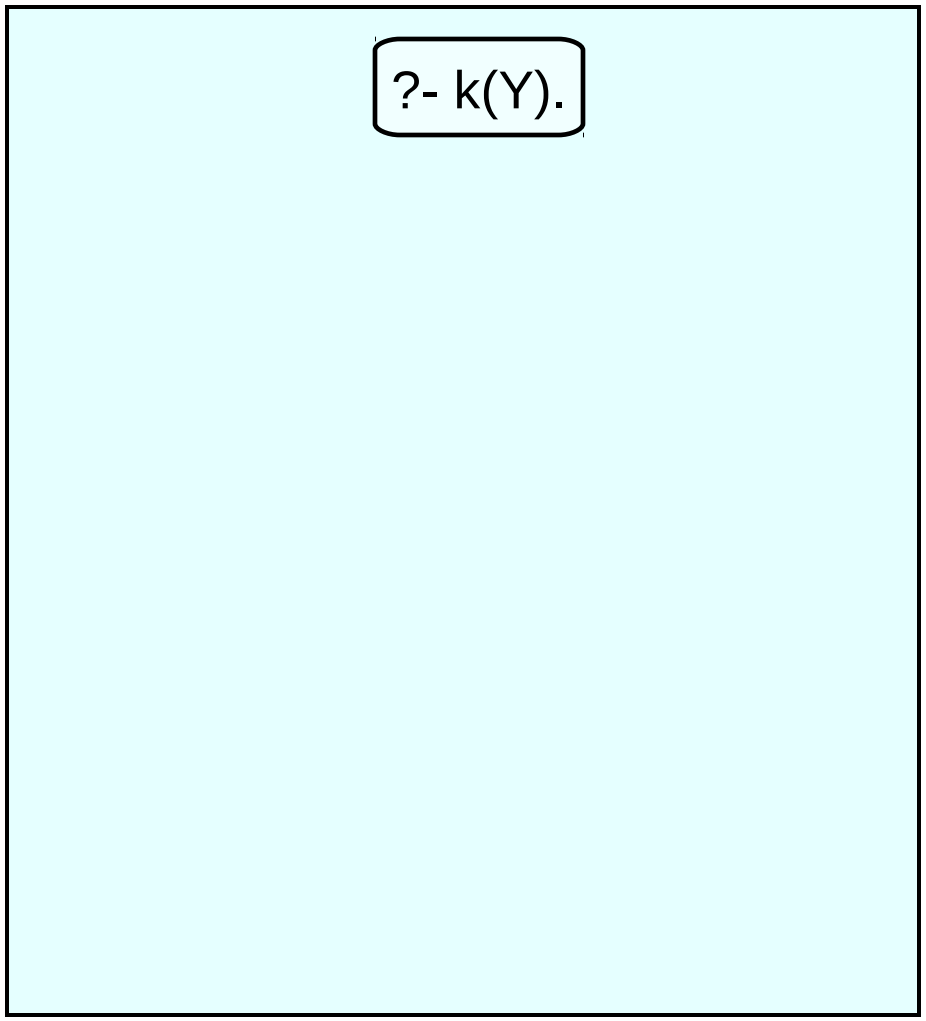
```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

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

Exemplo: árvore de busca

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

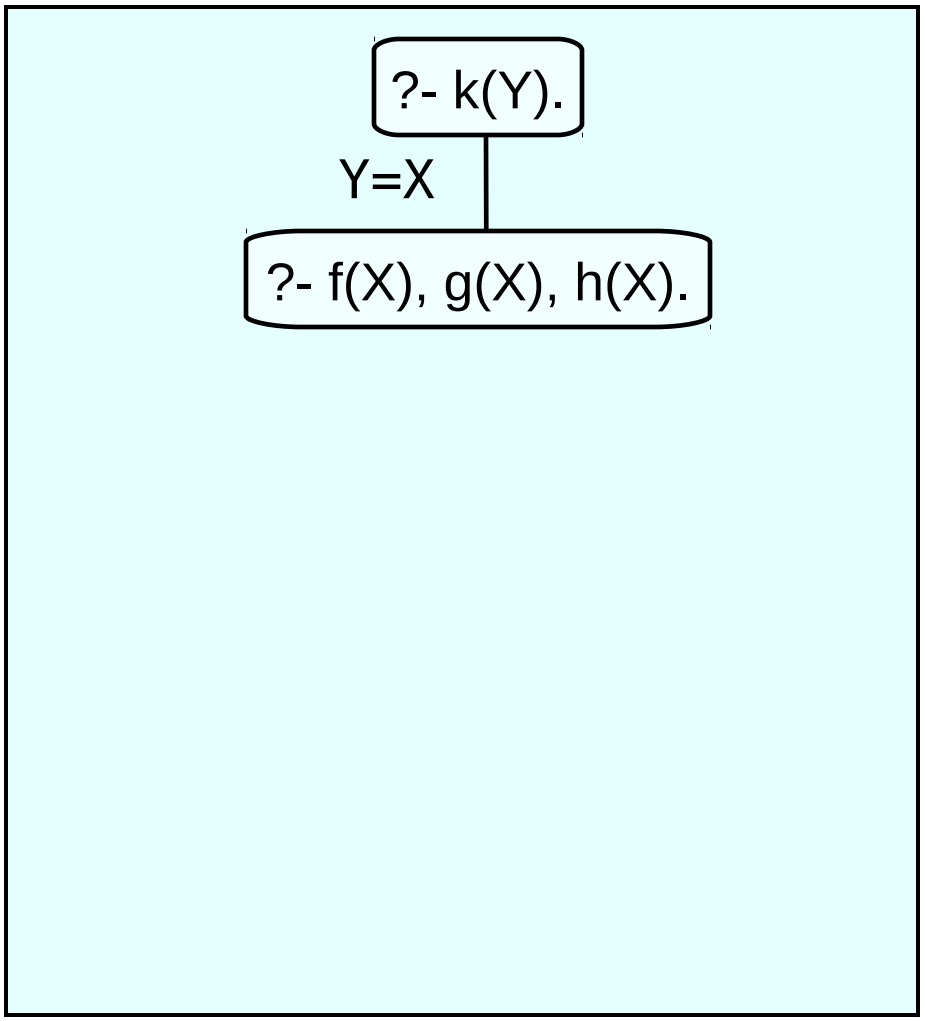
```
?- k(Y).
```



Exemplo: árvore de busca

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

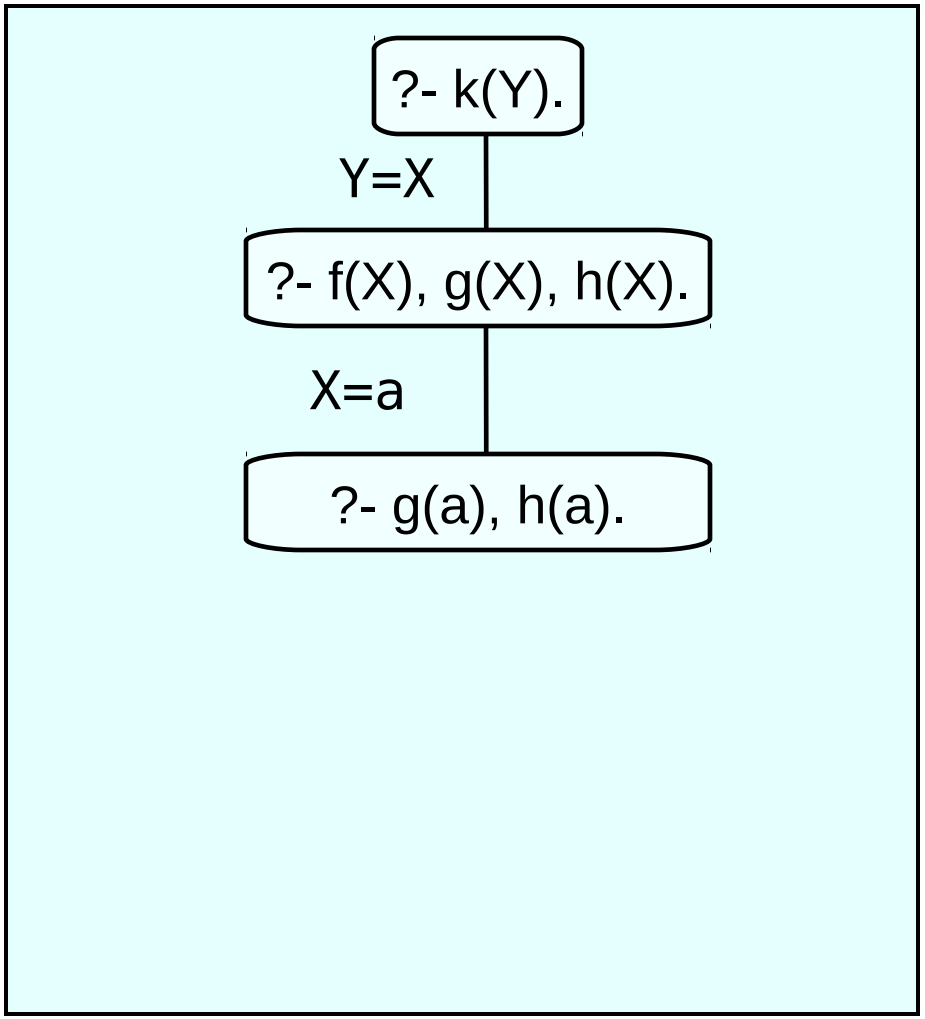
```
?- k(Y).
```



Exemplo: árvore de busca

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

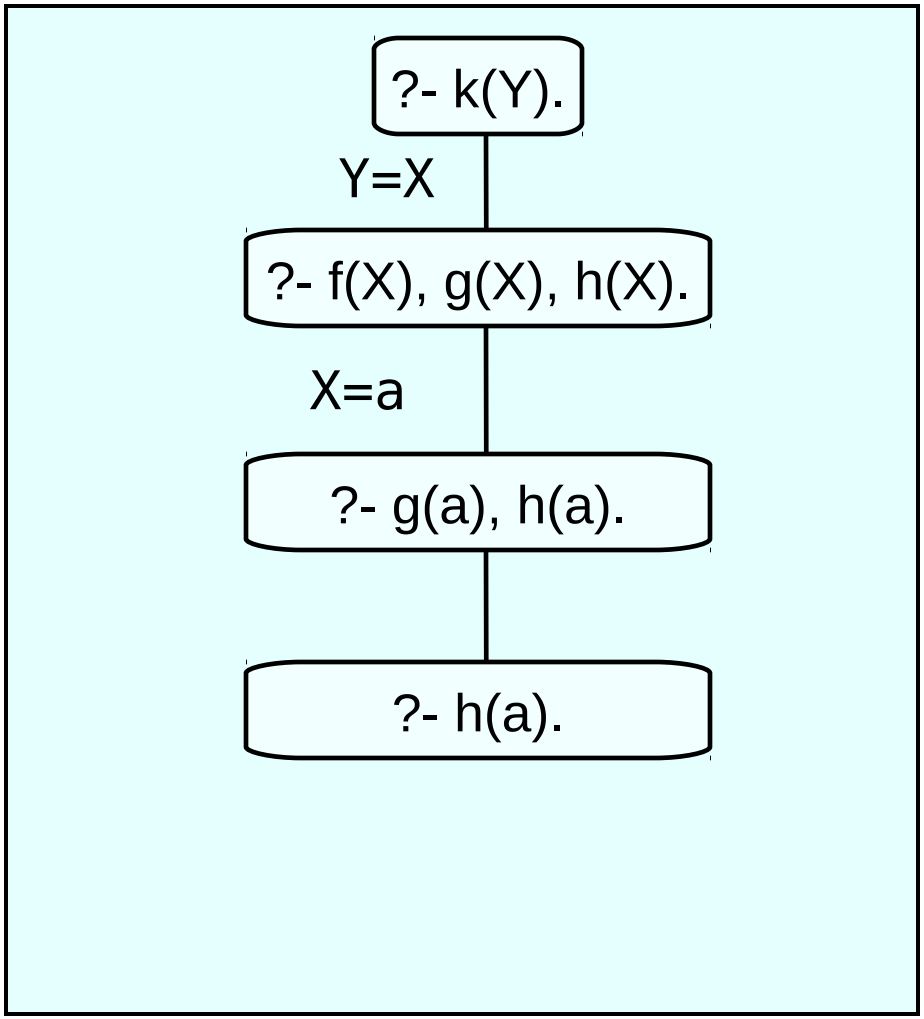
```
?- k(Y).
```



Exemplo: árvore de busca

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

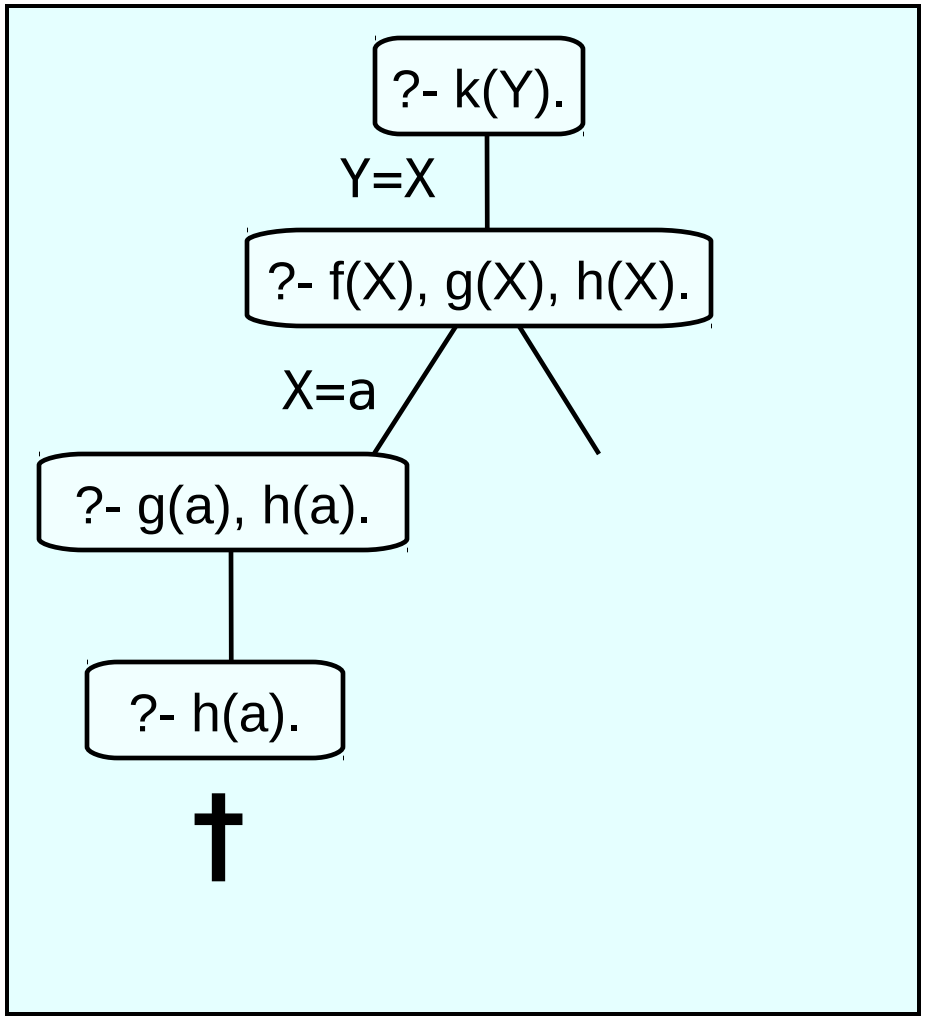
```
?- k(Y).
```



Exemplo: árvore de busca

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

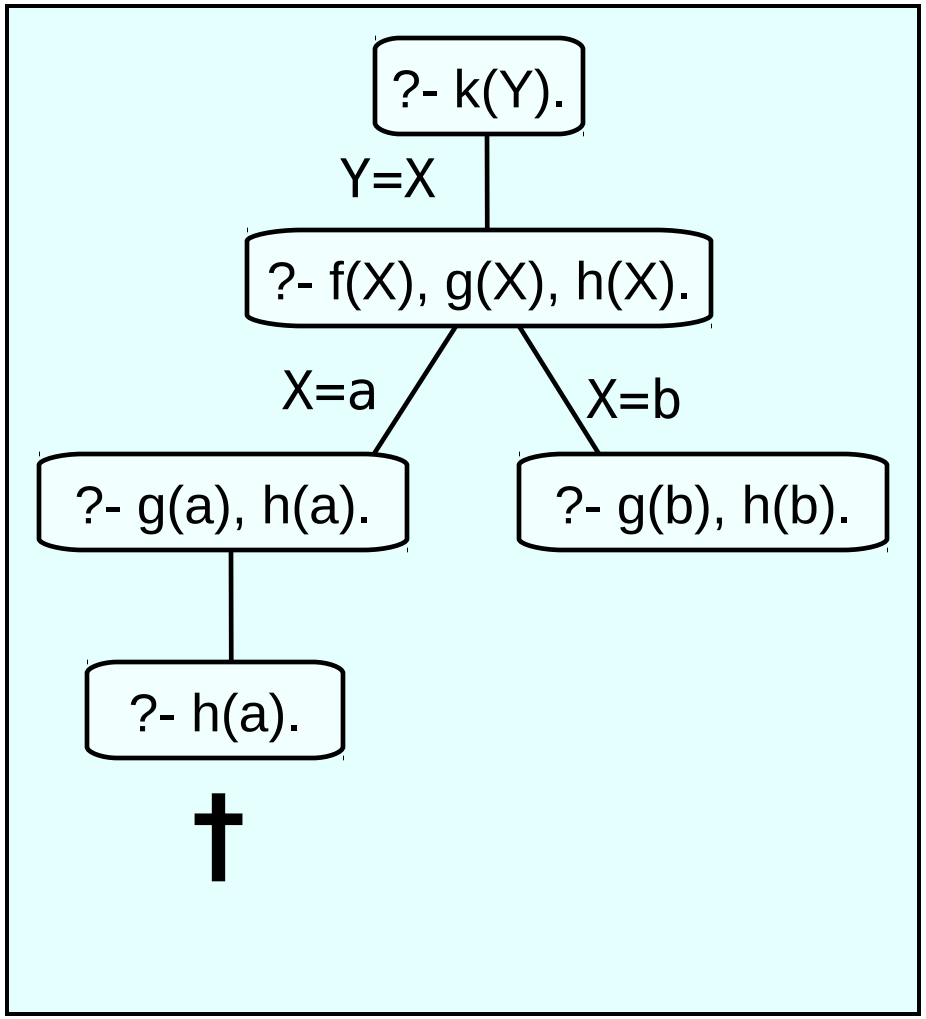
```
?- k(Y).
```



Exemplo: árvore de busca

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

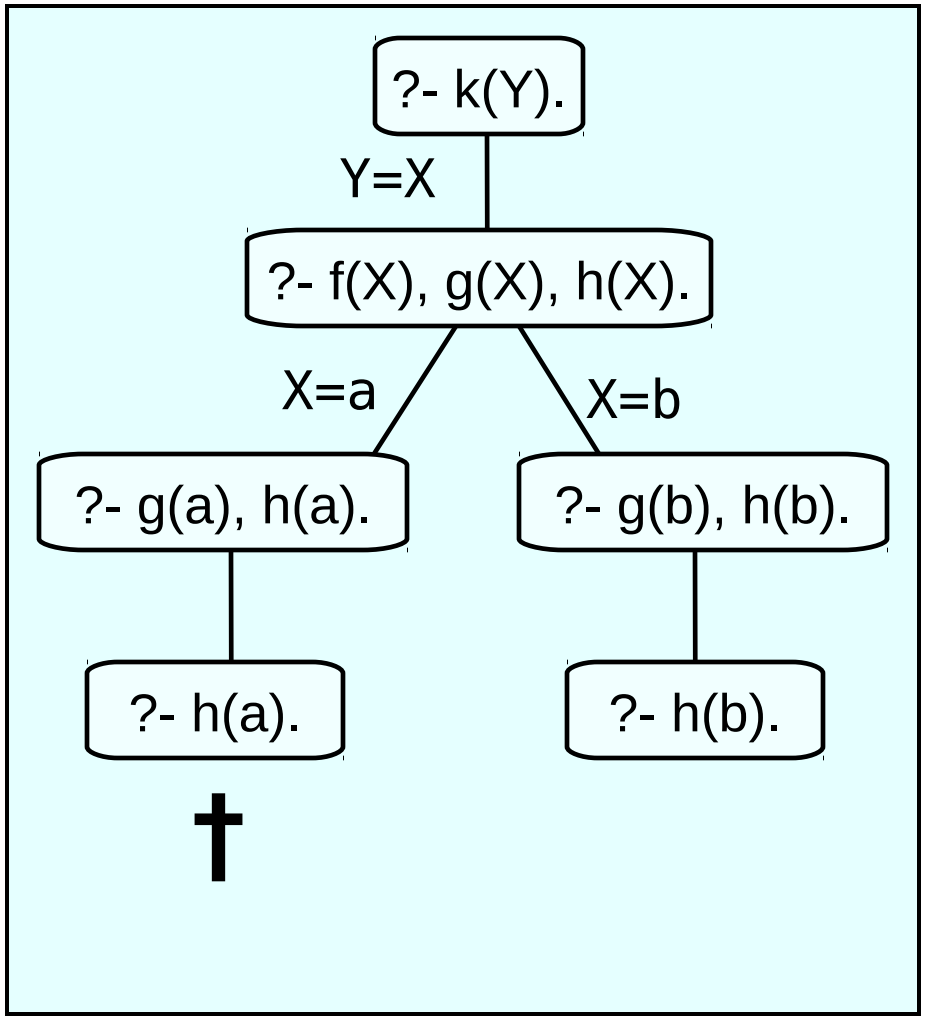
```
?- k(Y).
```



Exemplo: árvore de busca

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

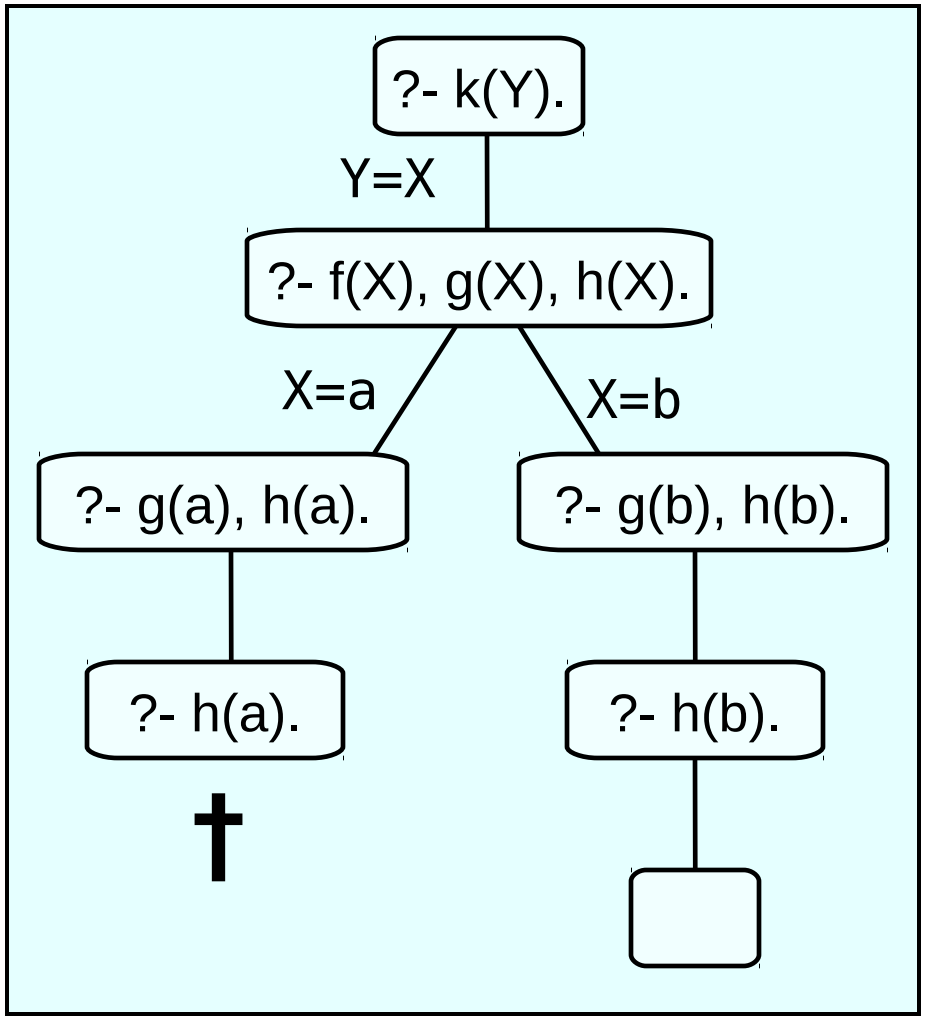
```
?- k(Y).
```



Exemplo: árvore de busca

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

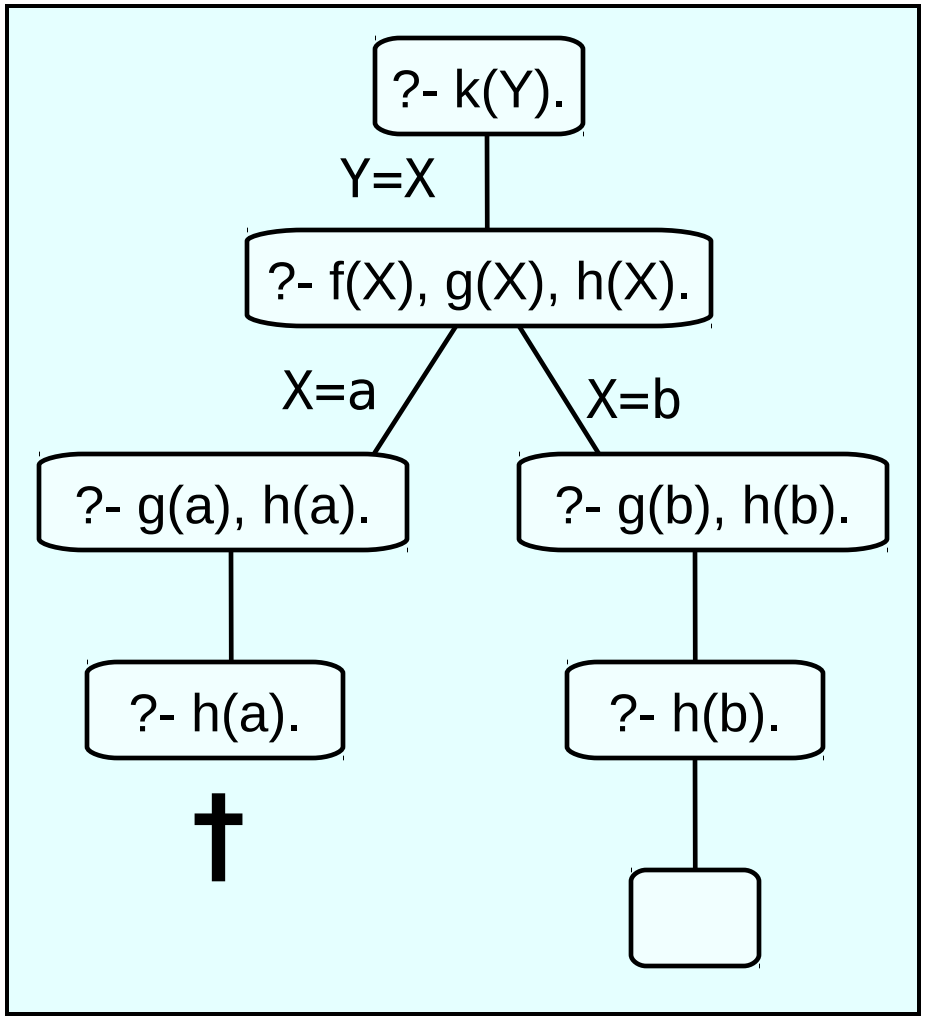
```
?- k(Y).
Y=b
```



Exemplo: árvore de busca

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

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



Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

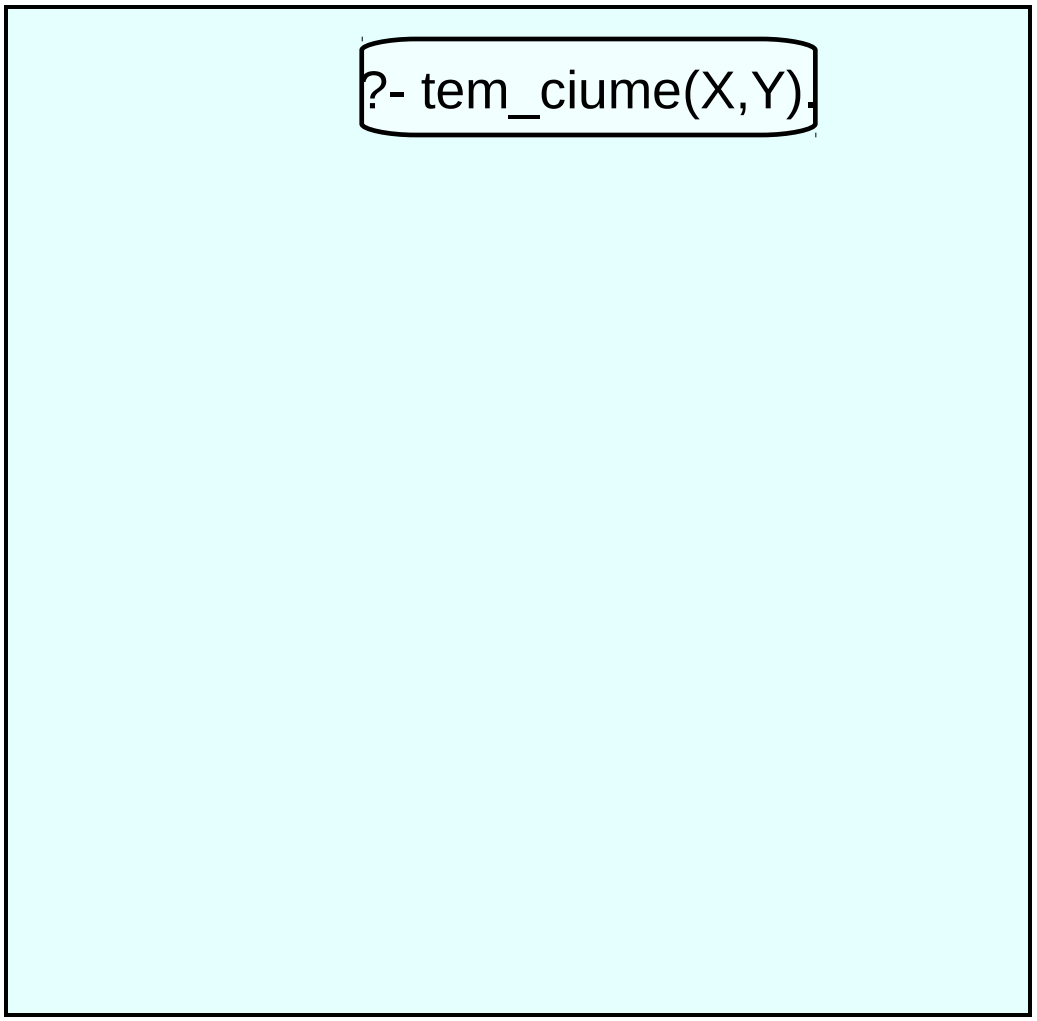
```
?- tem_ciume(X,Y).
```

Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

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

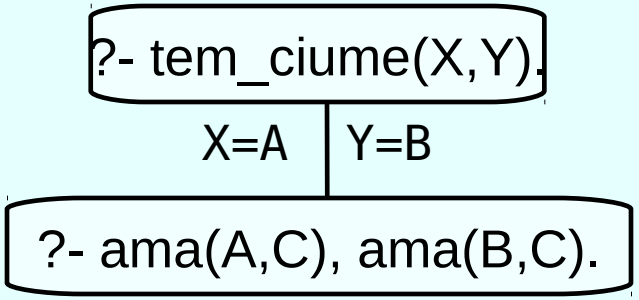


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

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

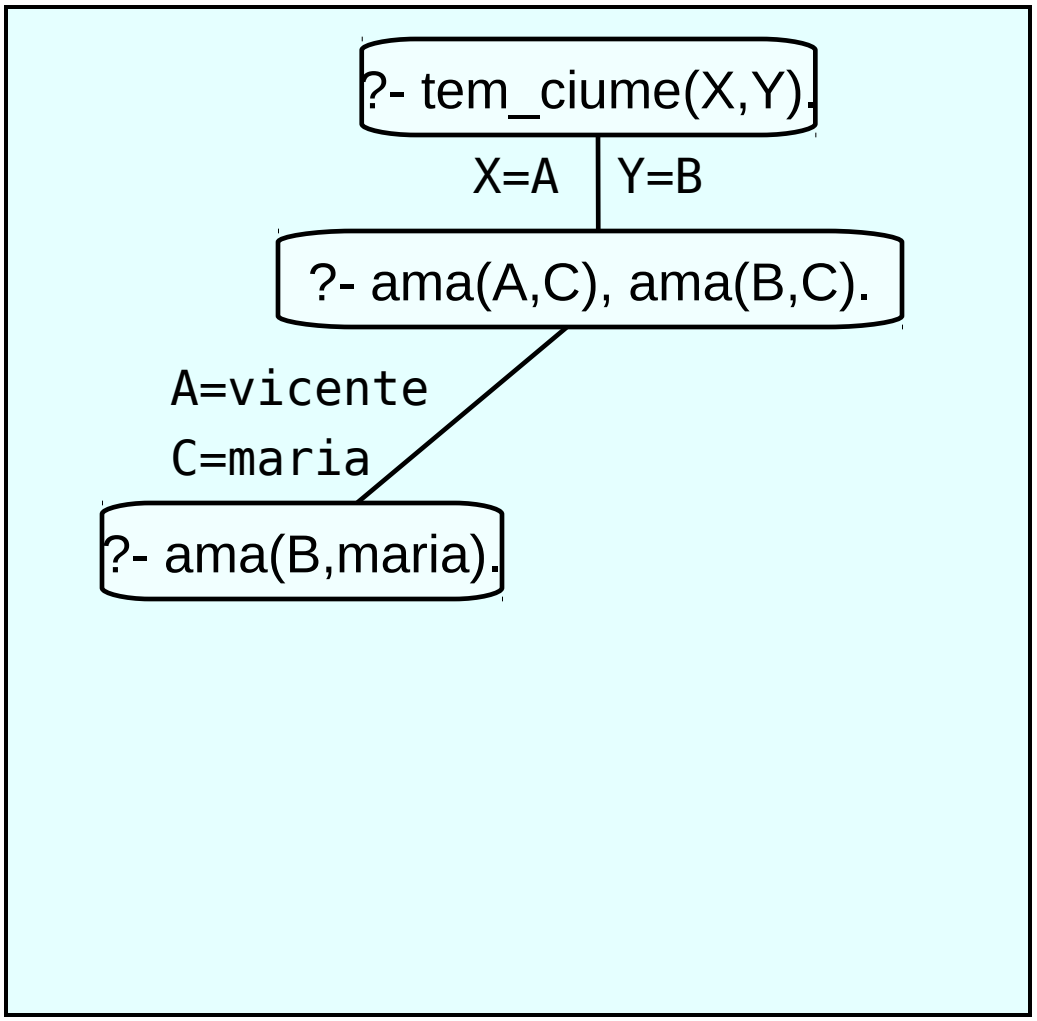


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

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

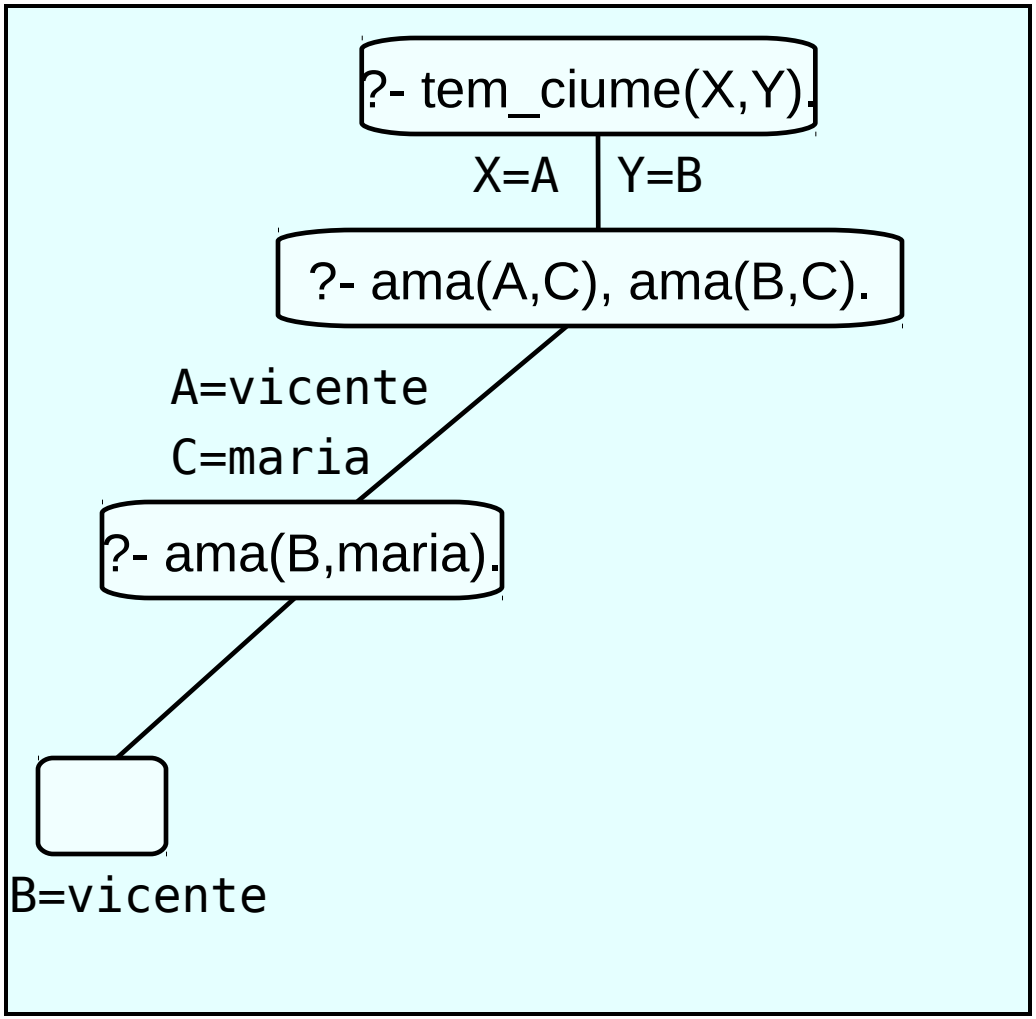


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

```
?- tem_ciume(X,Y).
X=vicente
Y=vicente
```

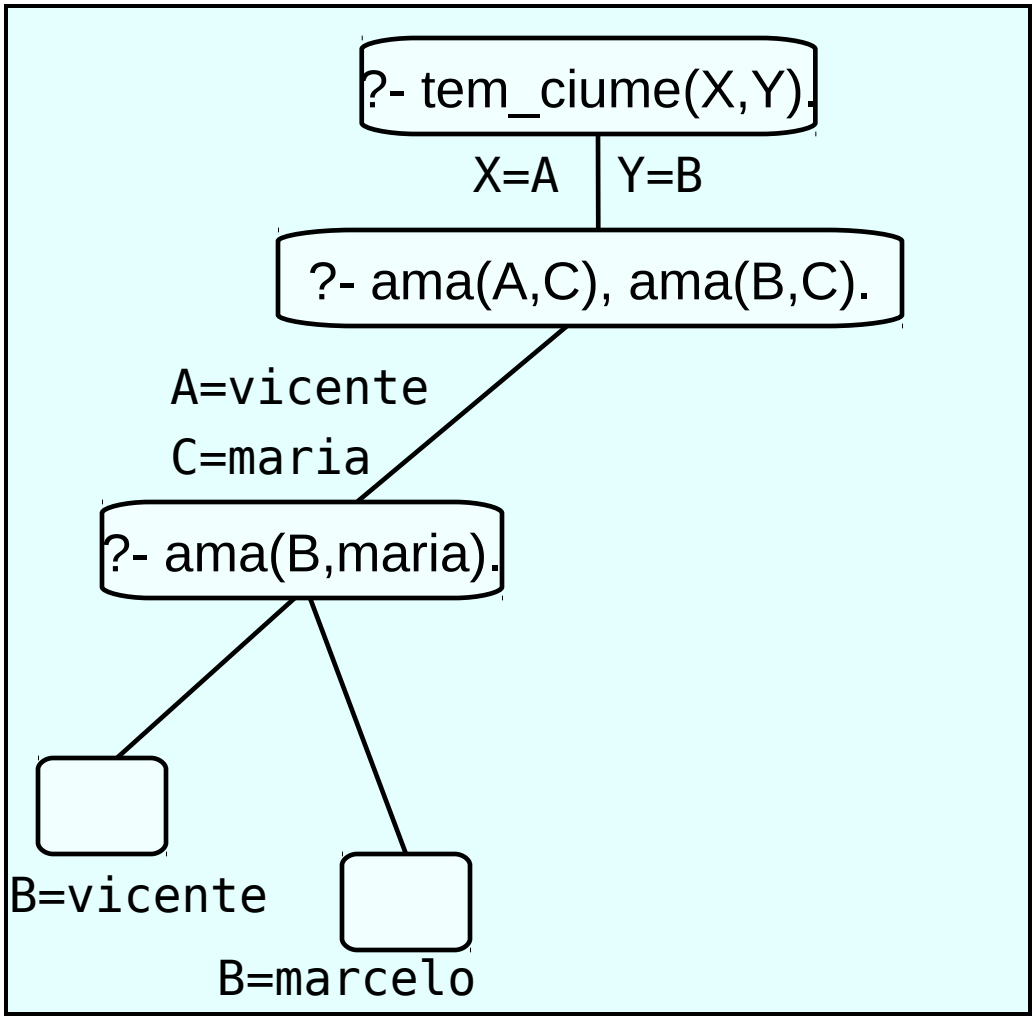


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

```
?- tem_ciume(X,Y).
X=vicente
Y=vicente;
X=vicente
Y=marcelo
```

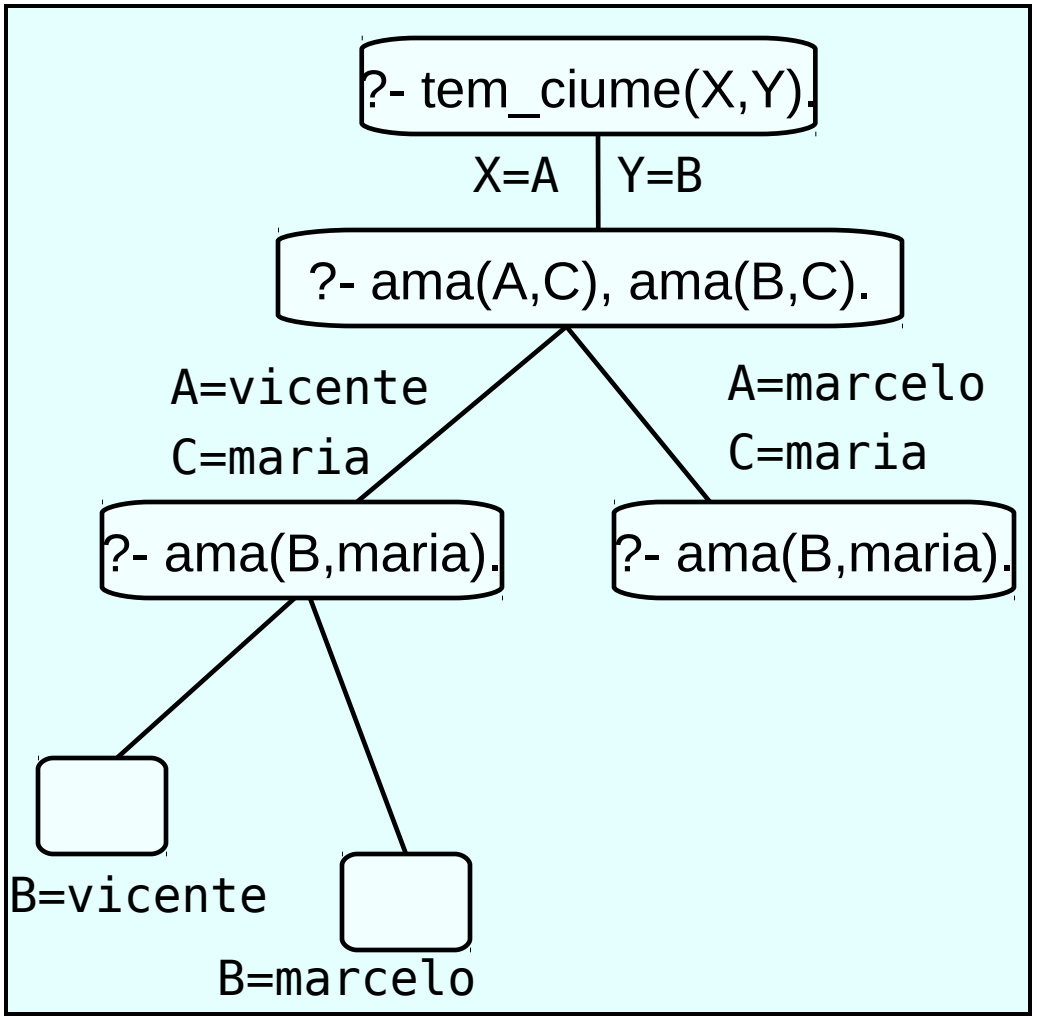


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

```
?- tem_ciume(X,Y).
X=vicente
Y=vicente;
X=vicente
Y=marcelo;
```

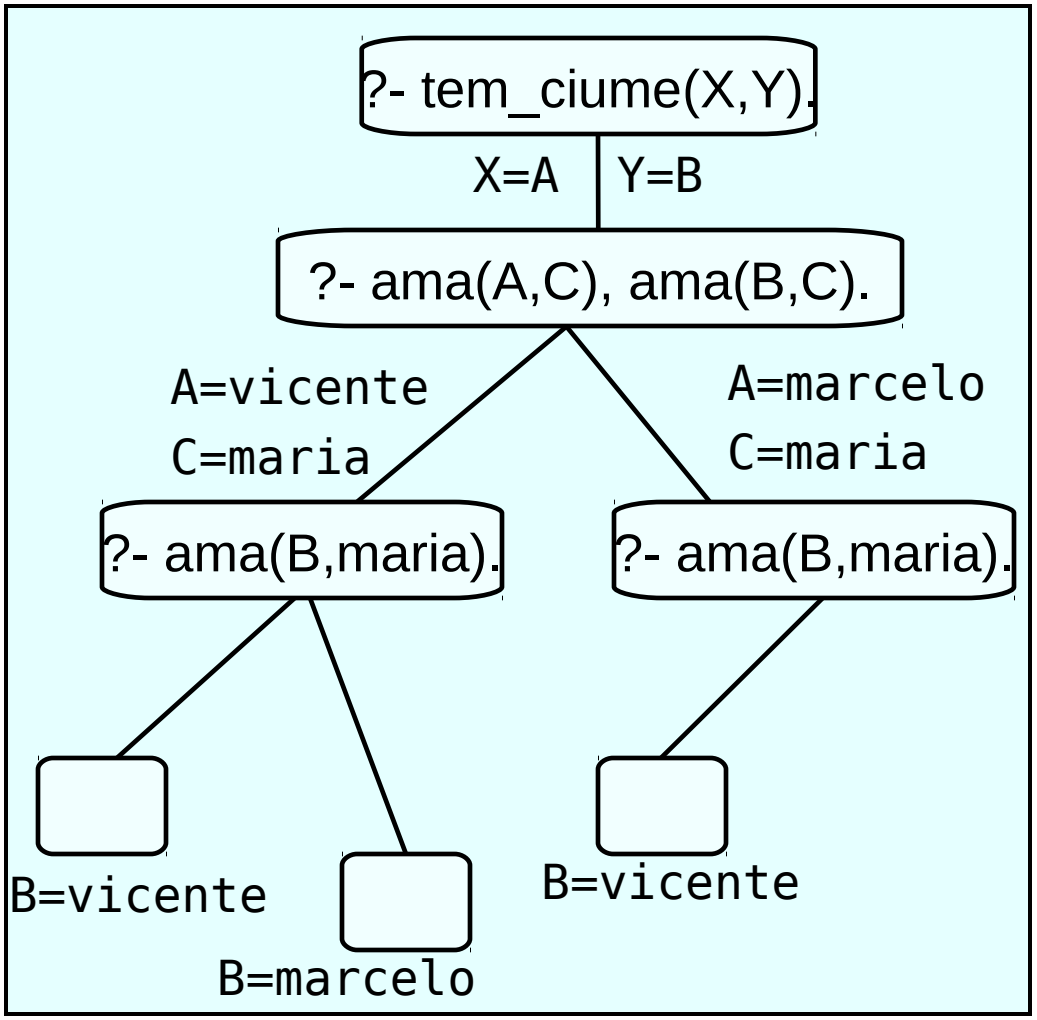


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

```
....
X=vicente
Y=marcelo;
X=marcelo
Y=vicente
```

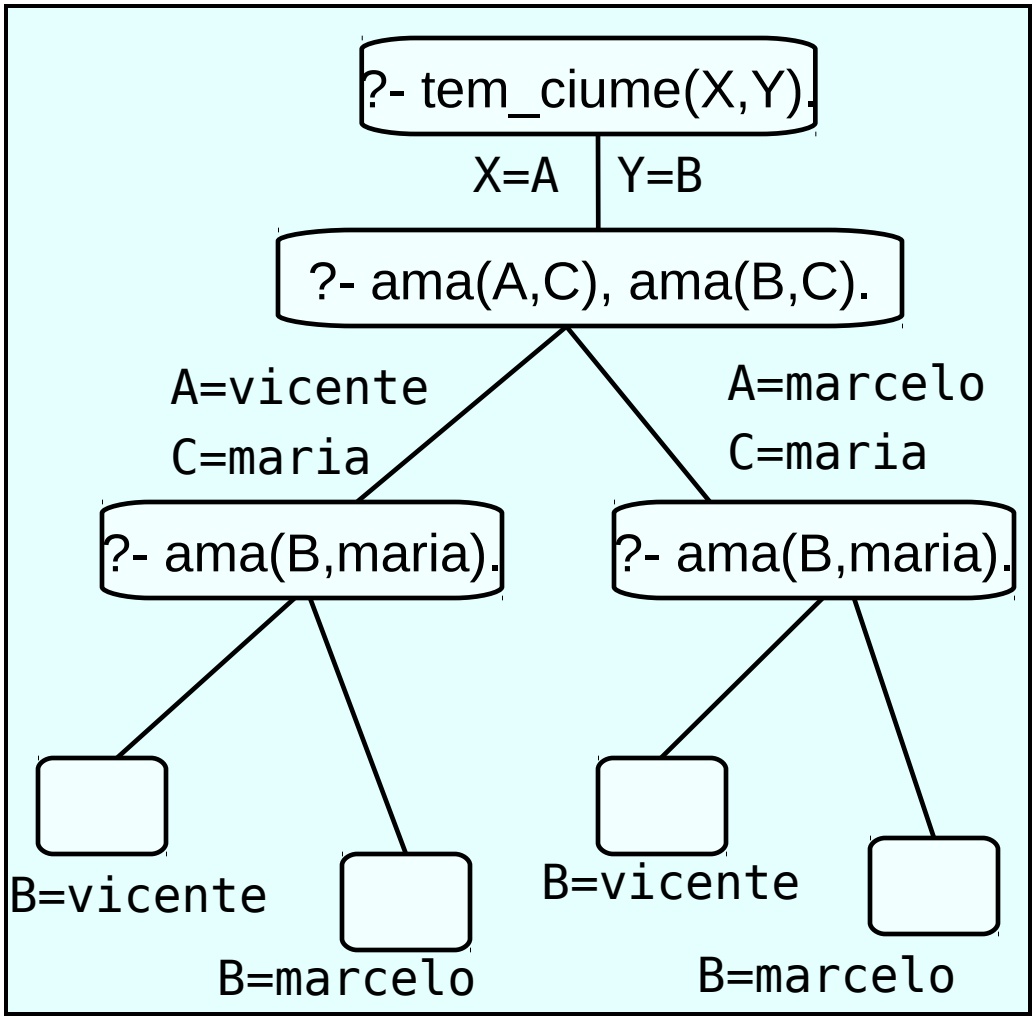


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

```
....
X=marcelo
Y=vicente;
X=marcelo
Y=marcelo
```

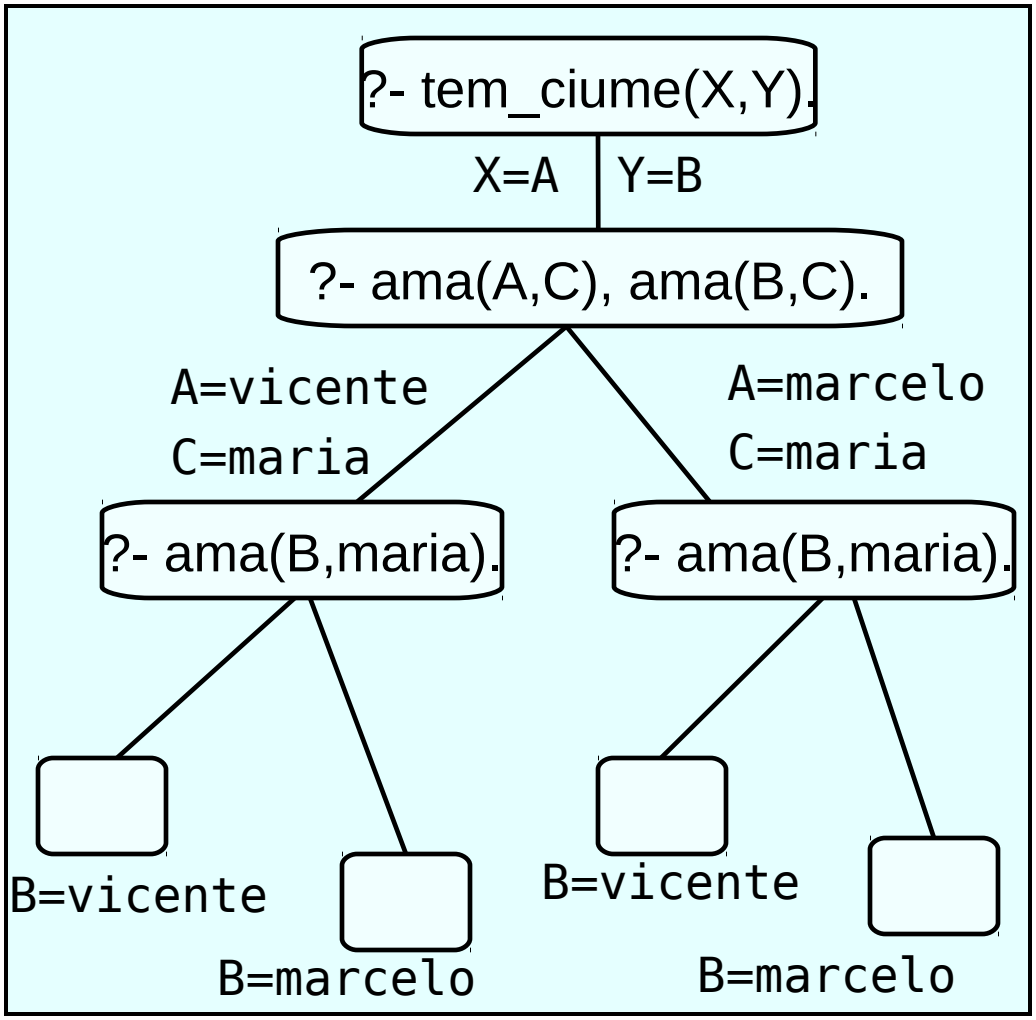


Um outro exemplo

```
ama(vicente,maria).
ama(marcelo,maria).

tem_ciume(A,B):-
    ama(A,C),
    ama(B,C).
```

```
....
X=marcelo
Y=vicente;
X=marcelo
Y=marcelo;
false
```



Resumo desta aula

- Nesta aula nós
  - Definimos unificação
  - Vimos a diferença entre a unificação padrão e a unificação em Prolog
  - Introduzimos árvores de busca

Exercícios

- Fazer a lista de exercícios 1 que se encontra no site do grupo.

Próxima aula

- Discutiremos **recursão** em Prolog
  - Introduziremos definições recursivas em Prolog
  - Mostraremos que podem existir incompatibilidades entre o significado declarativo de um programa Prolog e seu significado procedimental.