Lógica Computacional

Aula Teórica 21: Introdução ao Prolog

Ricardo Gonçalves

Departamento de Informática

24 de novembro de 2023

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

```
amigo(joao, jose).
amigo(jose, maria).
conhecido(X,Z):-amigo(X,Y), amigo(Y,Z).
```

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

```
amigo(joao, jose).
amigo(jose, maria).
conhecido(X,Z):-amigo(X,Y), amigo(Y,Z).
```

- Factos são fórmulas atómicas:
  amigo(joao, jose) e amigo(jose, maria)

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

```
amigo(joao, jose).
amigo(jose, maria).
conhecido(X,Z):-amigo(X,Y), amigo(Y,Z).
```

- Factos são fórmulas atómicas:
  amigo(joao, jose) e amigo(jose, maria)
- As regras podem ser vistas como implicações:
  (amigo(X,Y) $\wedge$ amigo(Y,Z))$\rightarrow$ conhecido(X,Z)

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

```
amigo(joao, jose).
amigo(jose, maria).
conhecido(X,Z):-amigo(X,Y), amigo(Y,Z).
```

- Factos são fórmulas atómicas:
  amigo(joao, jose) e amigo(jose, maria)
- As regras podem ser vistas como implicações:
  (amigo(X,Y) $\wedge$ amigo(Y,Z))$\rightarrow$ conhecido(X,Z)

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

```
amigo(joao, jose).
amigo(jose, maria).
conhecido(X,Z):-amigo(X,Y), amigo(Y,Z).
```

- Factos são fórmulas atómicas:
  amigo(joao, jose) e amigo(jose, maria)
- As regras podem ser vistas como implicações:
  (amigo(X,Y) $\wedge$ amigo(Y,Z))$\rightarrow$ conhecido(X,Z)
  que é equivalente a:

Base de conhecimento

A que corresponde, em termos de Lógica de Primeira Ordem, uma base de conhecimento em Prolog, com factos e regras?

```
amigo(joao, jose).
amigo(jose, maria).
conhecido(X,Z):-amigo(X,Y), amigo(Y,Z).
```

- Factos são fórmulas atómicas:
  amigo(joao, jose) e amigo(jose, maria)
- As regras podem ser vistas como implicações:
  (amigo(X,Y) $\wedge$ amigo(Y,Z))$\rightarrow$ conhecido(X,Z)
  que é equivalente a:
  $\neg$ amigo(X,Y) $\vee$ $\neg$ amigo(Y,Z) $\vee$ conhecido(X,Z)

Base de conhecimento

- Uma base de conhecimento corresponde à conjunção de todas estas fórmulas

Base de conhecimento

- Uma base de conhecimento corresponde à conjunção de todas estas fórmulas
- Logo:

Base de conhecimento

- Uma base de conhecimento corresponde à conjunção de todas estas fórmulas
- Logo:
- Uma base de conhecimento em Prolog (na versão apresentada) corresponde a uma fórmula de Horn!

Base de conhecimento

- Uma base de conhecimento corresponde à conjunção de todas estas fórmulas
- Logo:
- Uma base de conhecimento em Prolog (na versão apresentada) corresponde a uma fórmula de Horn!
- Corresponde à Forma de Implicações que vimos no caso proposicional.

Interrogações

A que correspondem as interrogações?

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

- Interrogações correspondem à conjunção dos objetivos:

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

- Interrogações correspondem à conjunção dos objetivos:

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

- Interrogações correspondem à conjunção dos objetivos: conhecido(joao, X) $\wedge$ conhecido(Y, joao)

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

- Interrogações correspondem à conjunção dos objetivos: conhecido(joao, X) $\wedge$ conhecido(Y, joao)
- O que queremos verificar quando temos uma Base de Conhecimento e uma Interrogação?

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

- Interrogações correspondem à conjunção dos objetivos: conhecido(joao, X) $\wedge$ conhecido(Y, joao)
- O que queremos verificar quando temos uma Base de Conhecimento e uma Interrogação?

Interrogações

A que correspondem as interrogações?

```
?- conhecido(joao, X), conhecido(Y, joao).
```

- Interrogações correspondem à conjunção dos objetivos: conhecido(joao, X) $\wedge$ conhecido(Y, joao)
- O que queremos verificar quando temos uma Base de Conhecimento e uma Interrogação?

Queremos verificar se a Interrogação é consequência da Base de Conhecimento.

Prolog e Resolução

- Como usamos Resolução para verificar se a Interrogação é consequência da Base de Conhecimento?

Prolog e Resolução

- Como usamos Resolução para verificar se a Interrogação é consequência da Base de Conhecimento?
- Verificamos se a Base de Conhecimento juntamente com a negação da Interrogação é contraditória.

Prolog e Resolução

- Como usamos Resolução para verificar se a Interrogação é consequência da Base de Conhecimento?
- Verificamos se a Base de Conhecimento juntamente com a negação da Interrogação é contraditória.
- No exemplo anterior, isto daria origem às seguintes cláusulas:

  C_1={amigo(joao, jose)}
  C_3={amigo(jose, maria)}
  C_3={conhecido(X,Z), ¬ amigo(X,Y), ¬amigo(Y,Z)}
  C_4={¬ conhecido(joao, X), ¬ conhecido(Y, joao)}

Prolog e Resolução

- Como usamos Resolução para verificar se a Interrogação é consequência da Base de Conhecimento?
- Verificamos se a Base de Conhecimento juntamente com a negação da Interrogação é contraditória.
- No exemplo anterior, isto daria origem às seguintes cláusulas:

  C_1={amigo(joao, jose)}
  C_3={amigo(jose, maria)}
  C_3={conhecido(X,Z), ¬ amigo(X,Y), ¬amigo(Y,Z)}
  C_4={¬ conhecido(joao, X), ¬ conhecido(Y, joao)}

- Ainda está na Forna de Horn

Prolog e Resolução

- Como usamos Resolução para verificar se a Interrogação é consequência da Base de Conhecimento?
- Verificamos se a Base de Conhecimento juntamente com a negação da Interrogação é contraditória.
- No exemplo anterior, isto daria origem às seguintes cláusulas:

  C_1={amigo(joao, jose)}
  C_3={amigo(jose, maria)}
  C_3={conhecido(X,Z), ¬ amigo(X,Y), ¬amigo(Y,Z)}
  C_4={¬ conhecido(joao, X), ¬ conhecido(Y, joao)}

- Ainda está na Forna de Horn
- A única cláusula negativa é a que corresponde à Interrogação.

Prolog e Resolução

- Como usamos Resolução para verificar se a Interrogação é consequência da Base de Conhecimento?
- Verificamos se a Base de Conhecimento juntamente com a negação da Interrogação é contraditória.
- No exemplo anterior, isto daria origem às seguintes cláusulas:

  C_1={amigo(joao, jose)}
  C_3={amigo(jose, maria)}
  C_3={conhecido(X,Z), ¬ amigo(X,Y), ¬amigo(Y,Z)}
  C_4={¬ conhecido(joao, X), ¬ conhecido(Y, joao)}

- Ainda está na Forna de Horn — e sempre estará!
- A única cláusula negativa é a que corresponde à Interrogação. — e sempre será!

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda
  - Partindo da cláusulas com os objetivos

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda - Partindo da cláusulas com os objetivos
- Mas como é que se encontra uma refutação?

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda - Partindo da cláusulas com os objetivos
- Mas como é que se encontra uma refutação?
- Procurando(!) a Base de Conhecimento

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda - Partindo da cláusulas com os objetivos
- Mas como é que se encontra uma refutação?
- Procurando(!) a Base de Conhecimento
- Como? Combinando procura, unificação e retrocesso

Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda - Partindo da cláusulas com os objetivos
- Mas como é que se encontra uma refutação?
- Procurando(!) a Base de Conhecimento
- Como? Combinando procura, unificação e retrocesso
- Procura na Base de Conhecimento sempre de cima para baixo

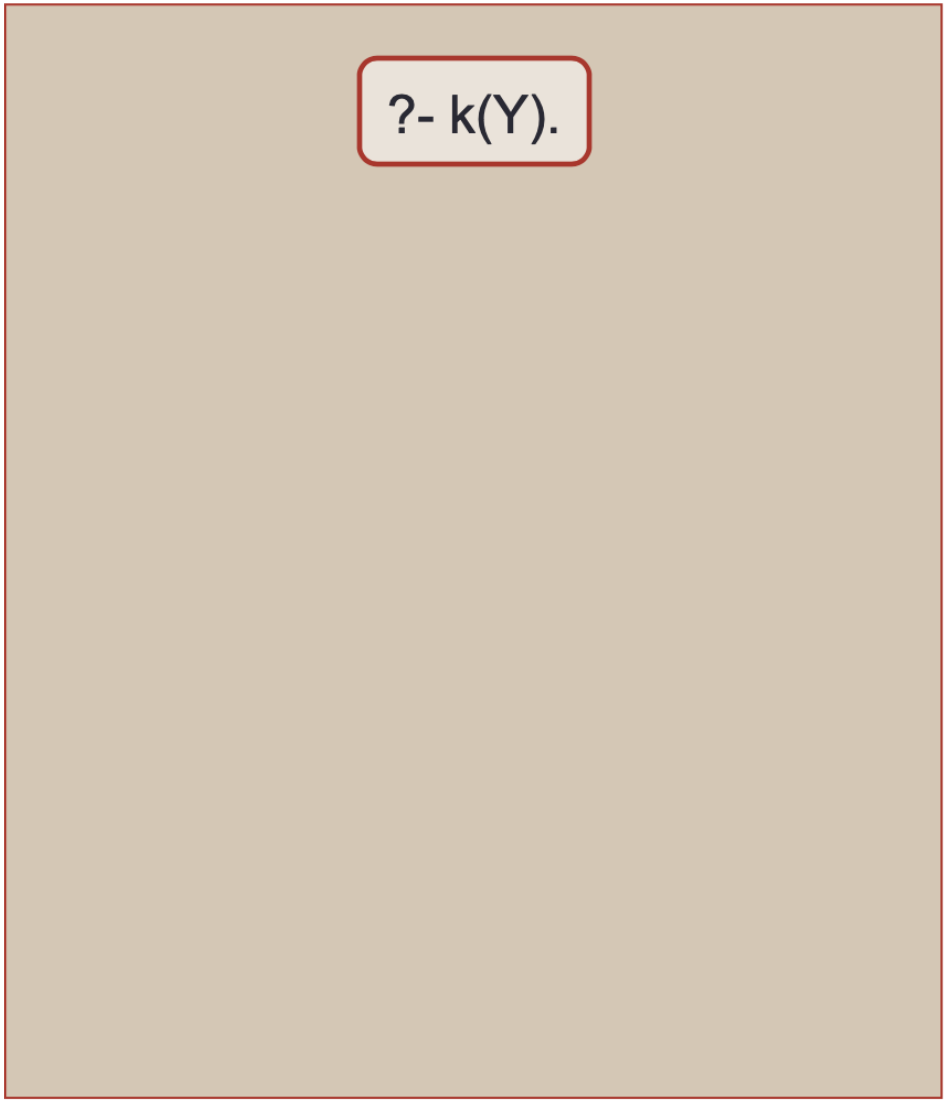
Pesquisa em Prolog

- Prolog usa como base a Resolução-SLD com seletor à esquerda - Partindo da cláusulas com os objetivos
- Mas como é que se encontra uma refutação?
- Procurando(!) a Base de Conhecimento
- Como? Combinando procura, unificação e retrocesso
- Procura na Base de Conhecimento sempre de cima para baixo
- **Árvore de procura:** visualização deste processo de procura

Procura - Exemplo 1

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

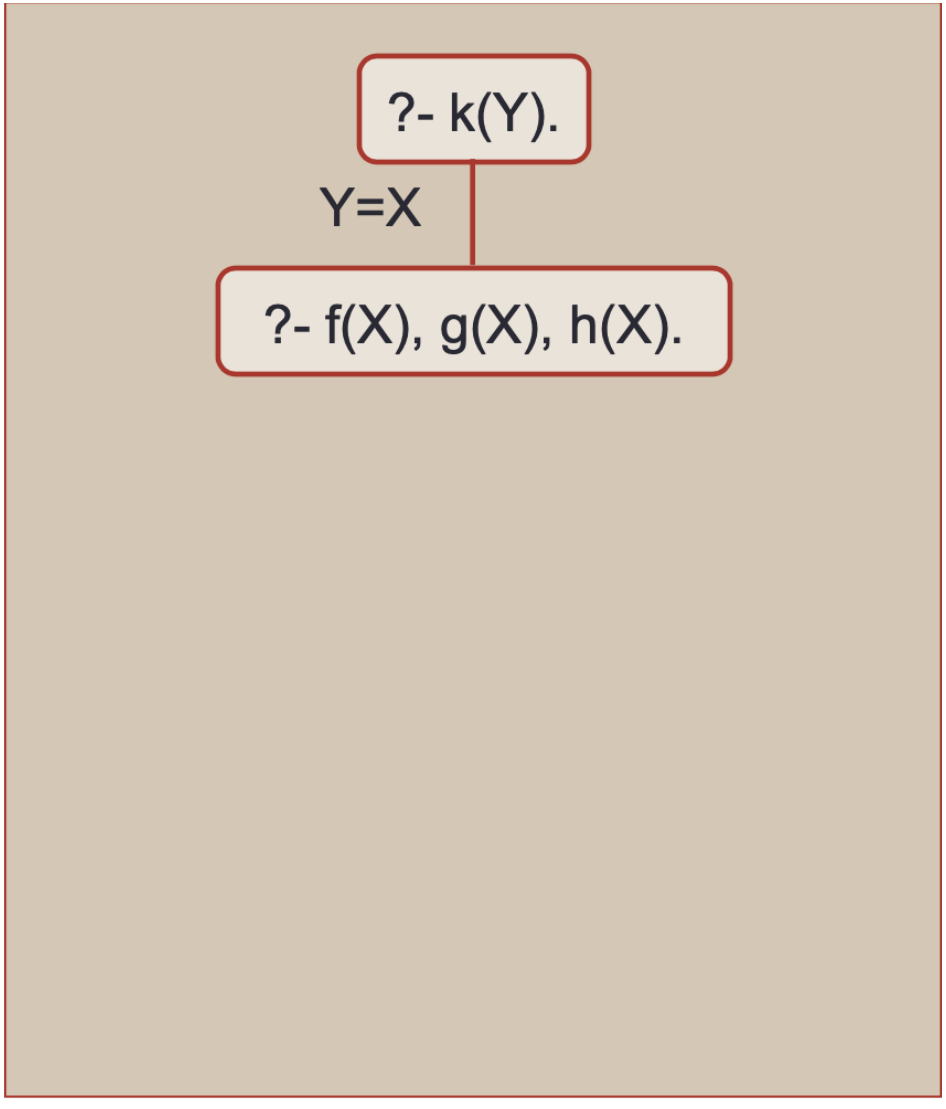
```
?- k(Y).
```



Procura - Exemplo 1

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

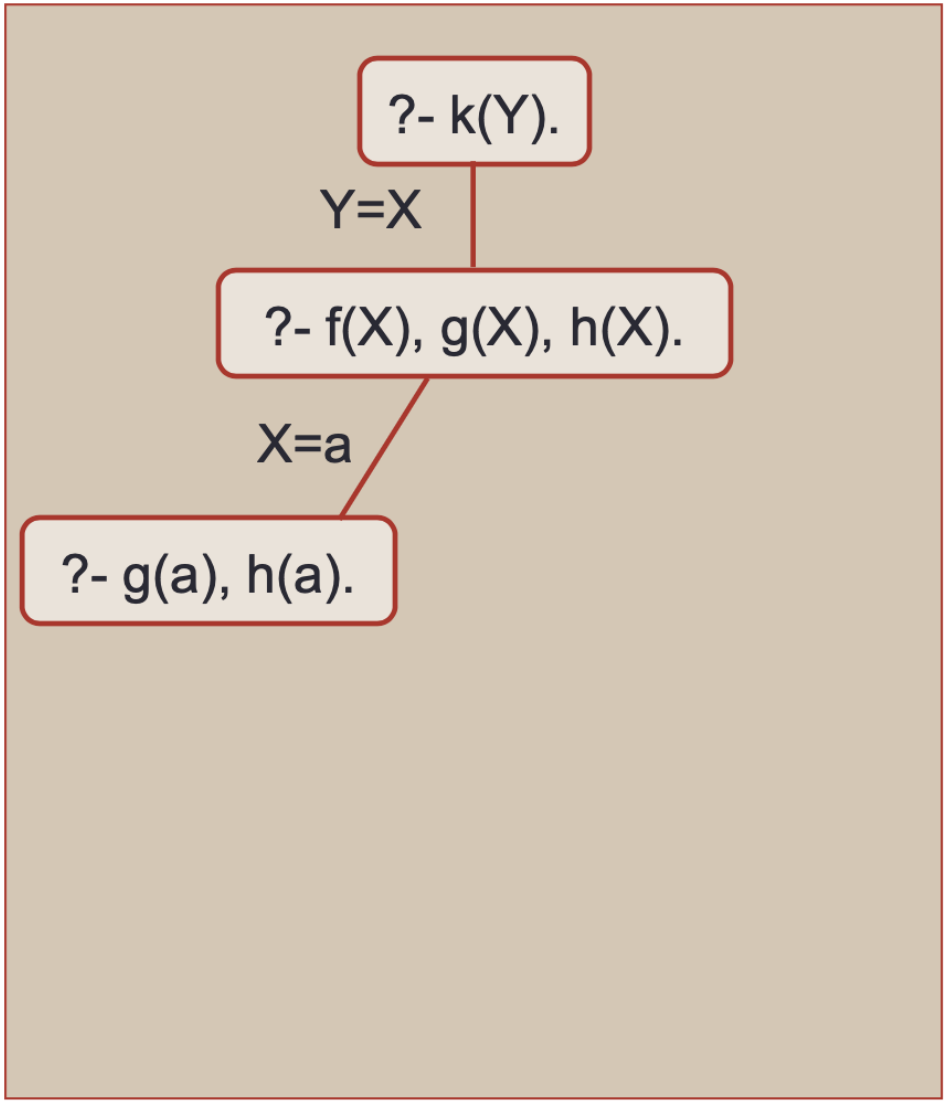
```
?- k(Y).
```



Procura - Exemplo 1

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

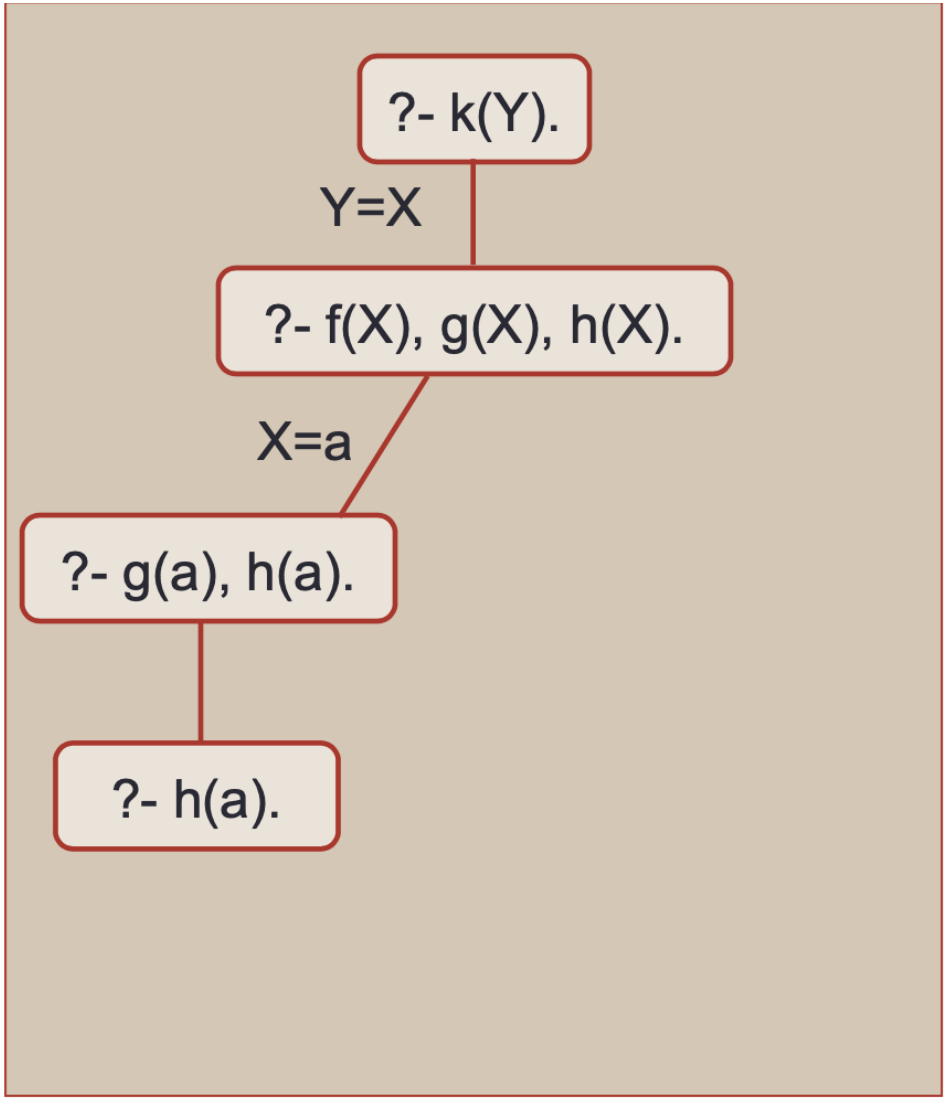
```
?- k(Y).
```



Procura - Exemplo 1

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

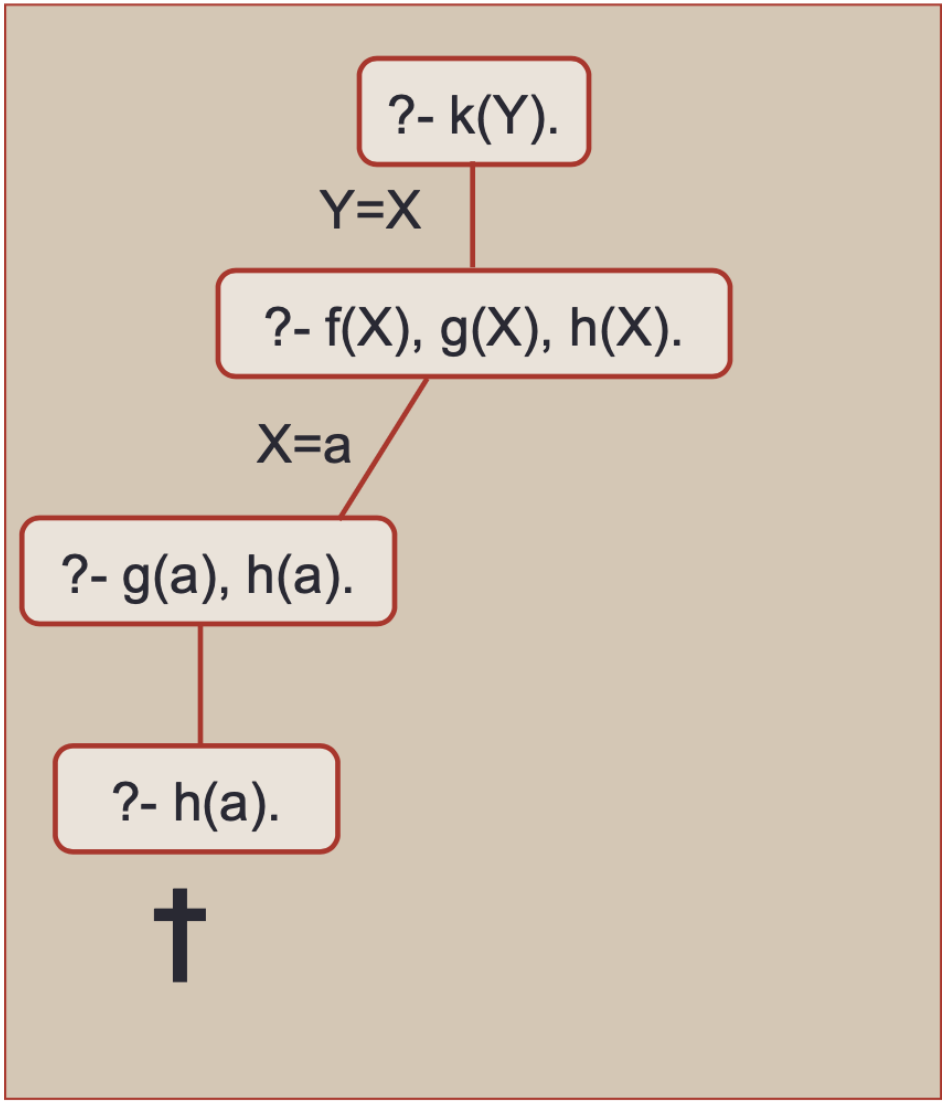
```
?- k(Y).
```



Procura - Exemplo 1

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

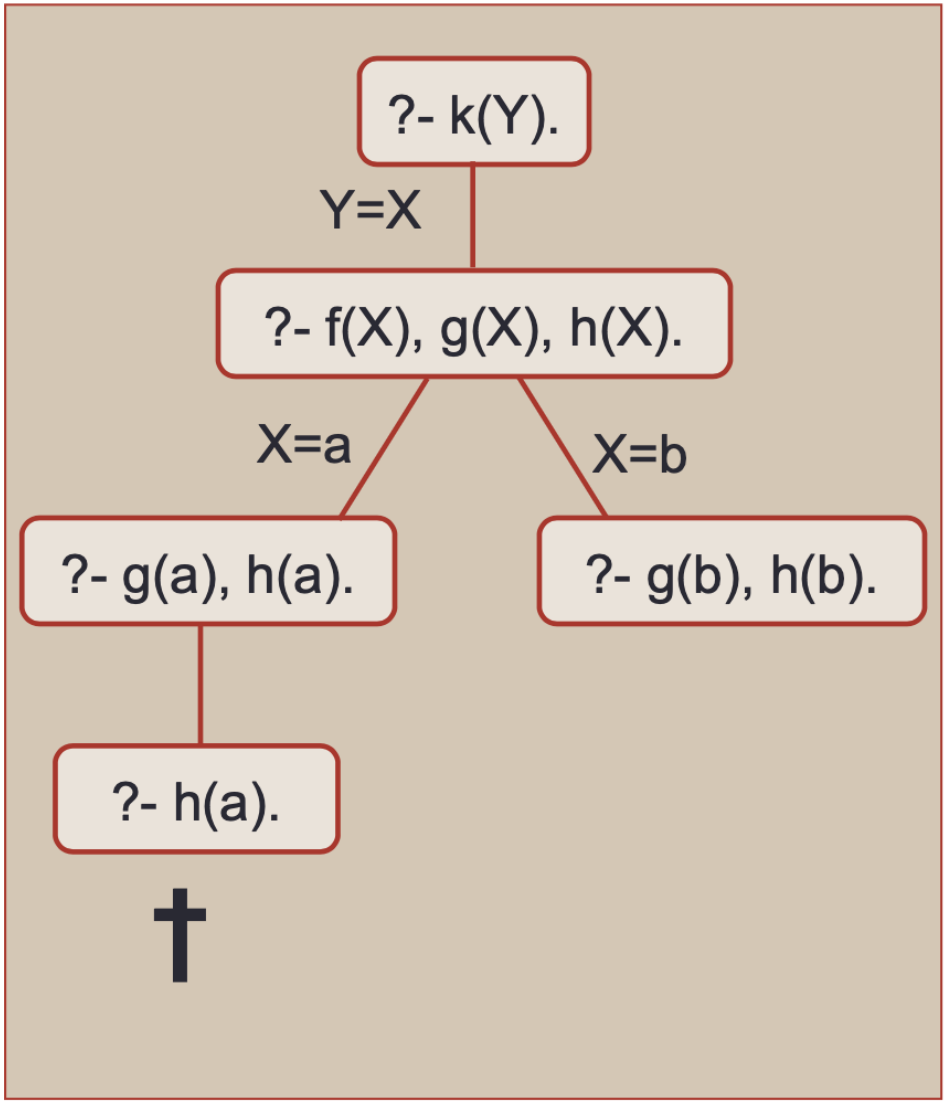
```
?- k(Y).
```



Procura - Exemplo 1

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

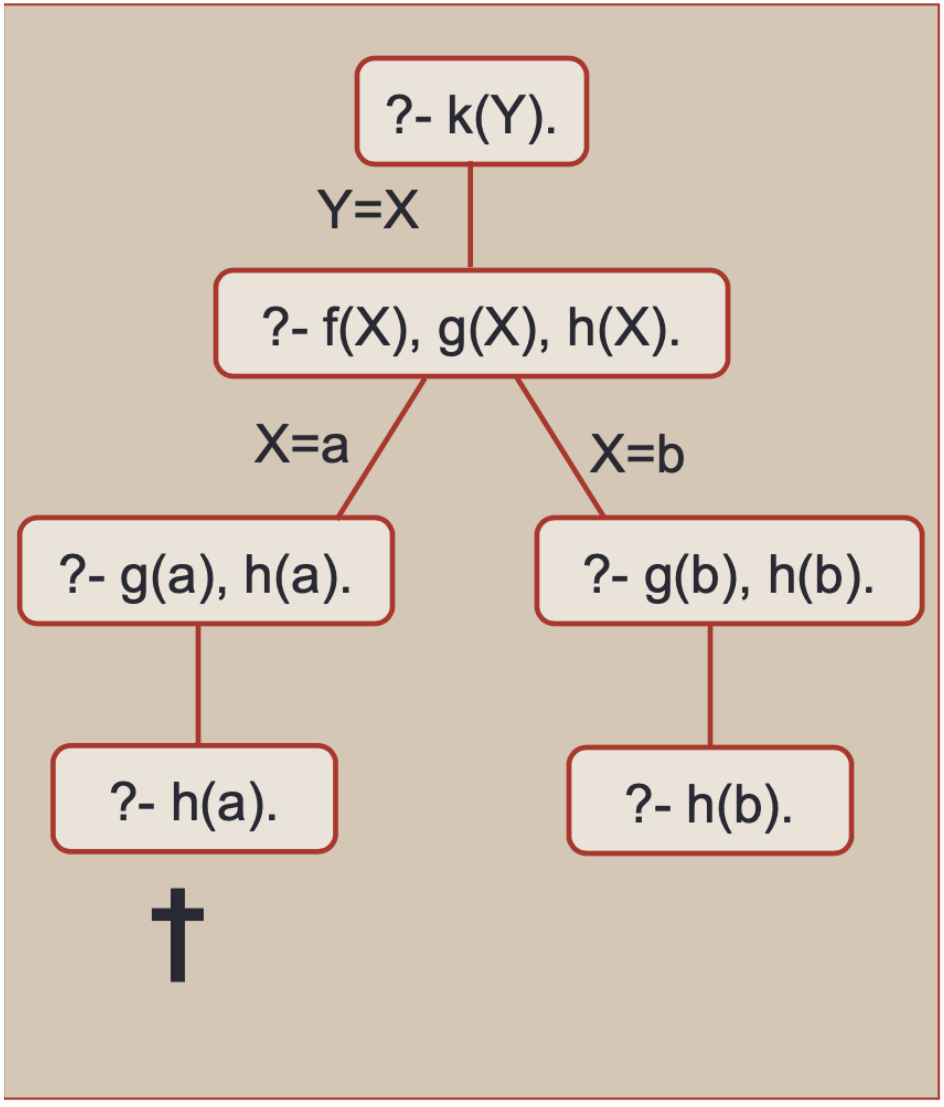
```
?- k(Y).
```



Procura - Exemplo 1

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

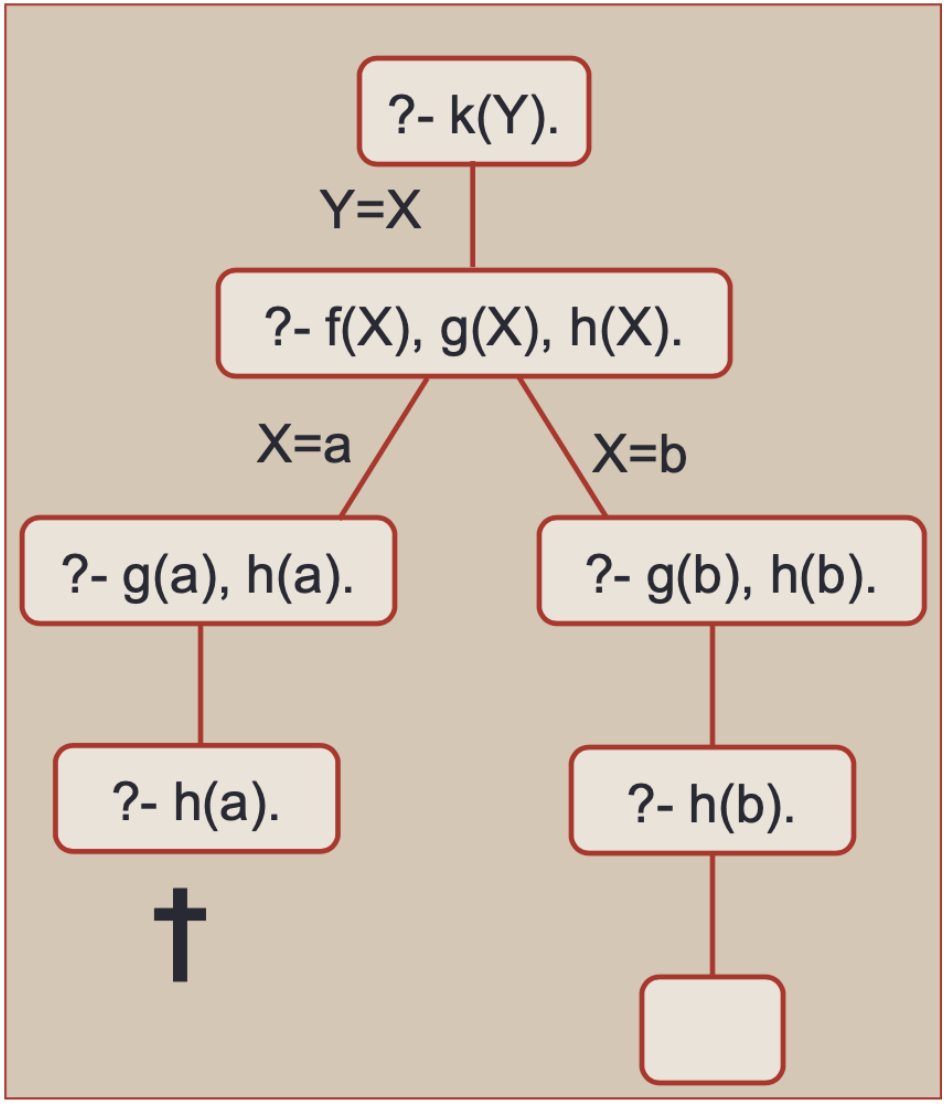
```
?- k(Y).
```



Procura - Exemplo 1

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

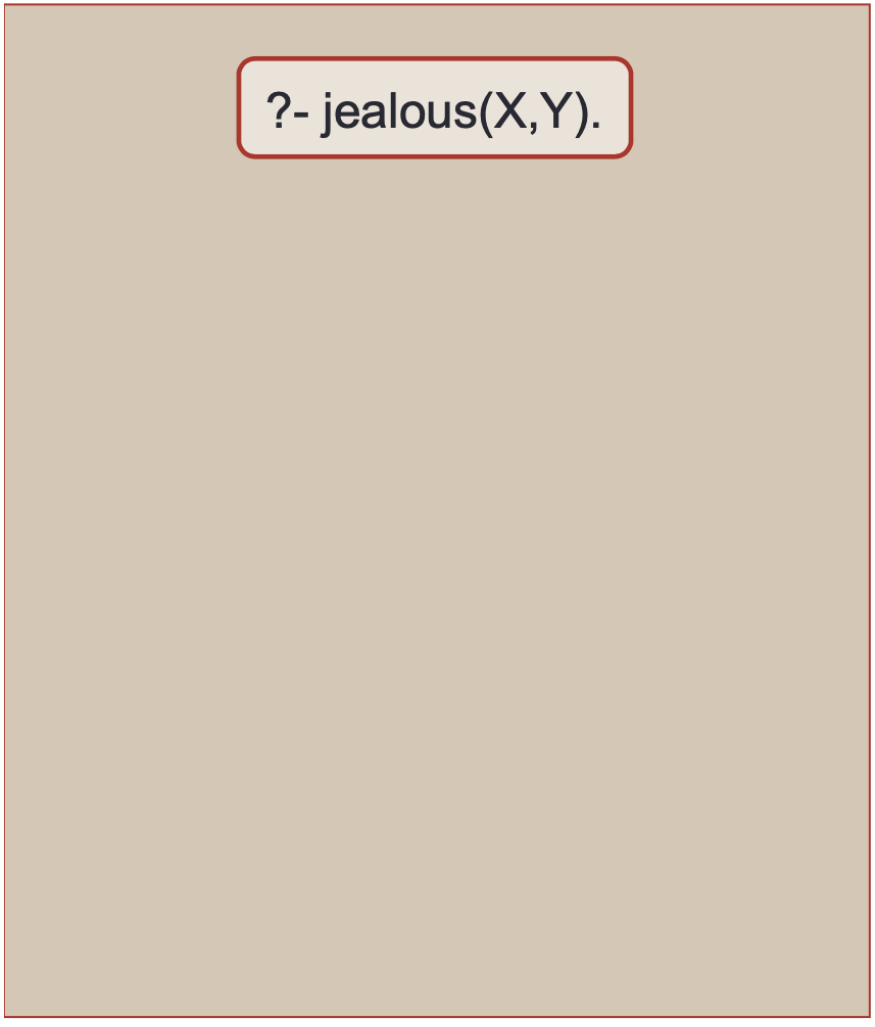
```
?- k(Y).
Y=b.
```



Procura - Exemplo 2

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

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



Procura - Exemplo 2

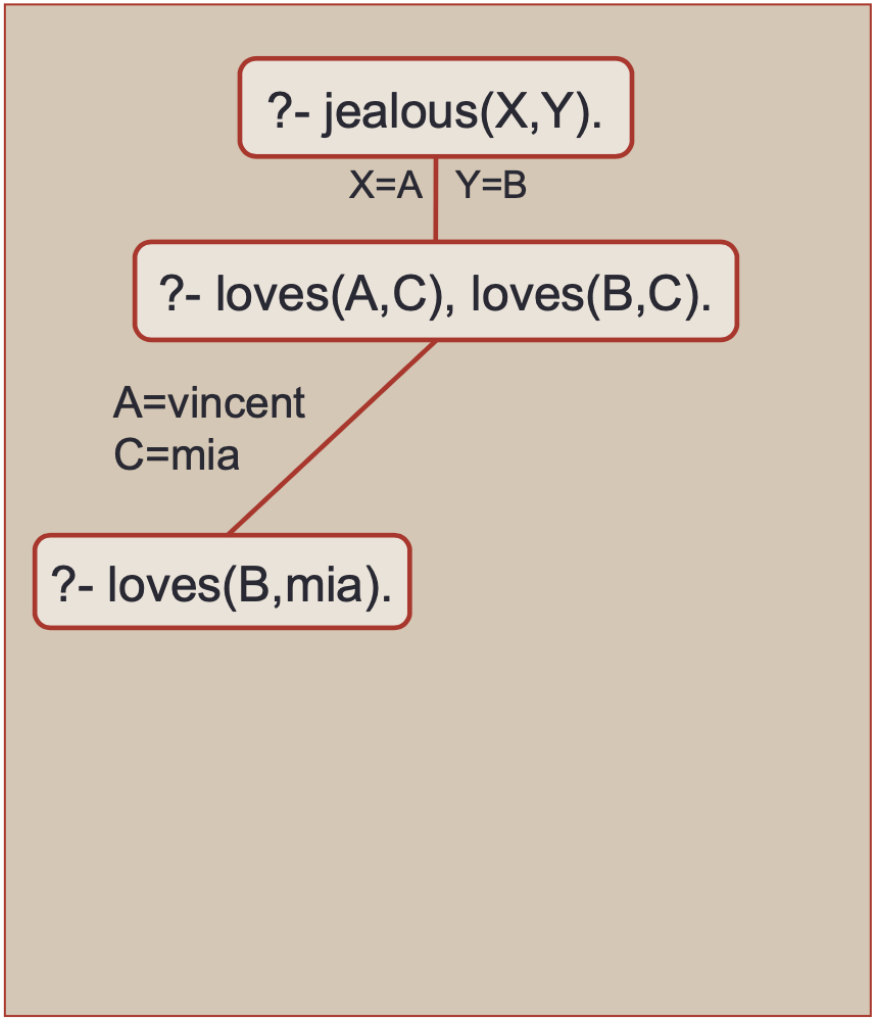
```
loves(vincent,mia).
loves(marsellus,mia).
jealous(A,B):- loves(A,C), loves(B,C).
```

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

Procura - Exemplo 2

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

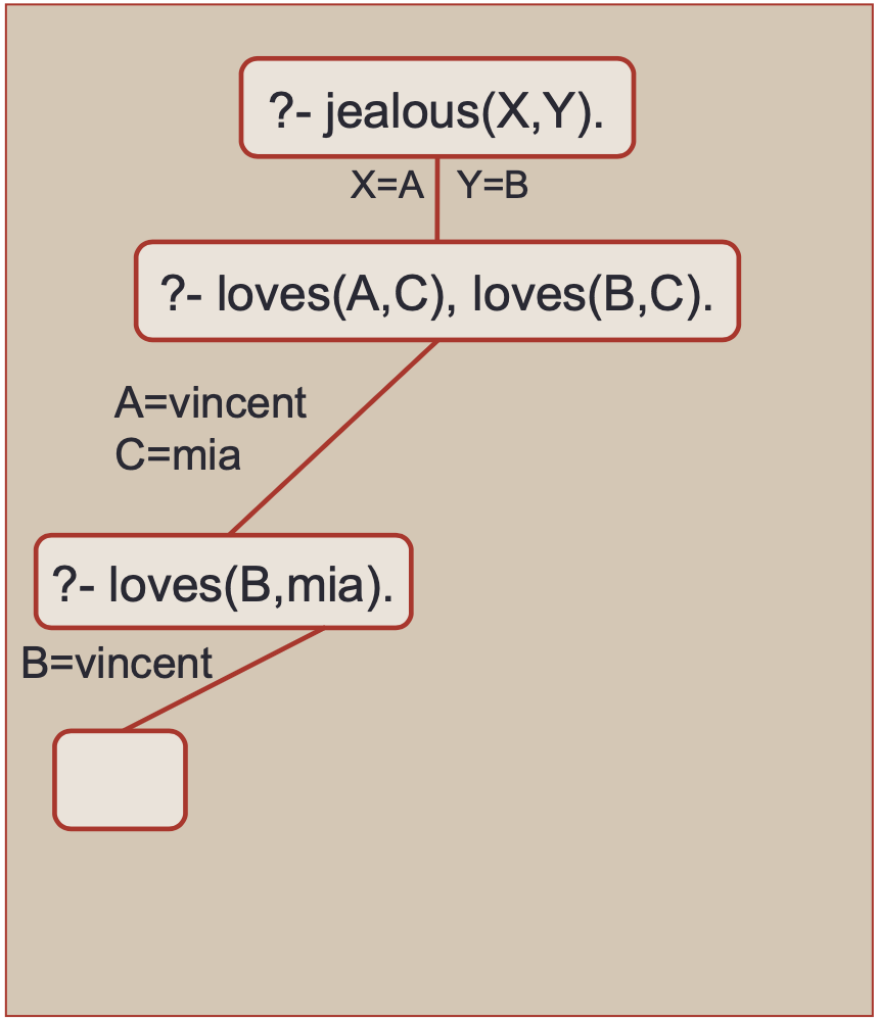
```
?- jealous(X,Y).
```



Procura - Exemplo 2

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

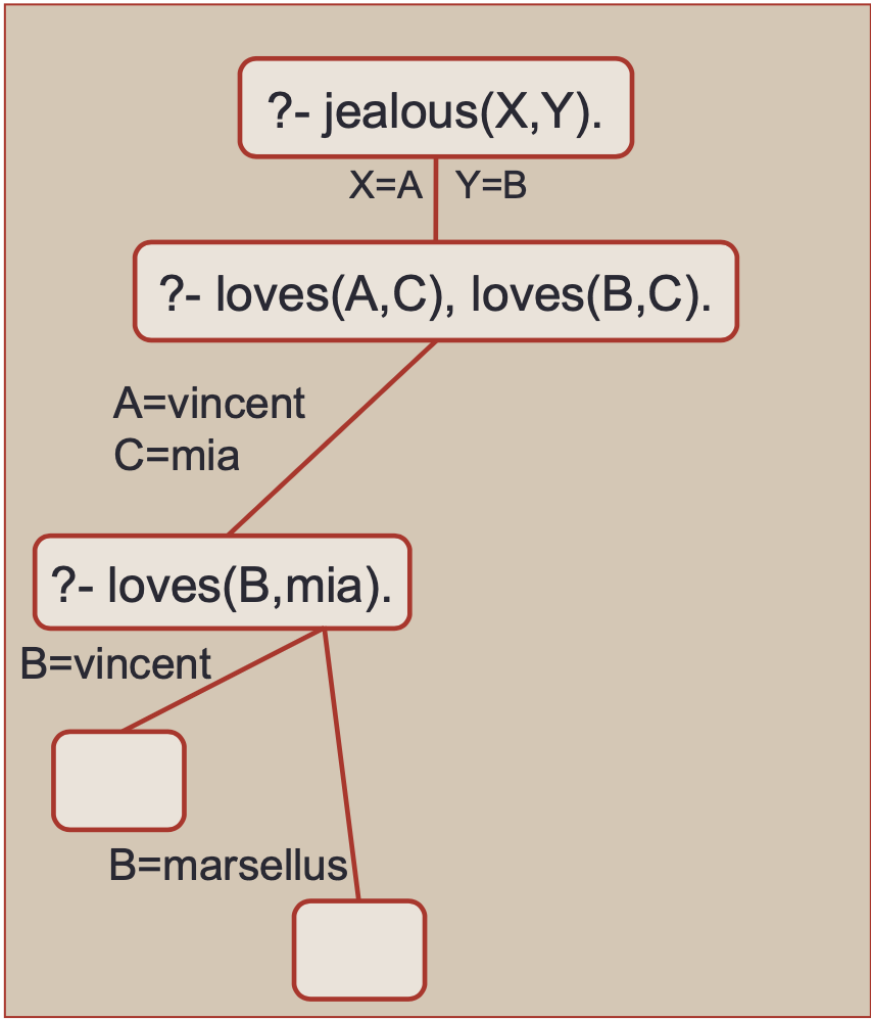
```
?- jealous(X,Y).
X=vincent,
Y=vincent;
```



Procura - Exemplo 2

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

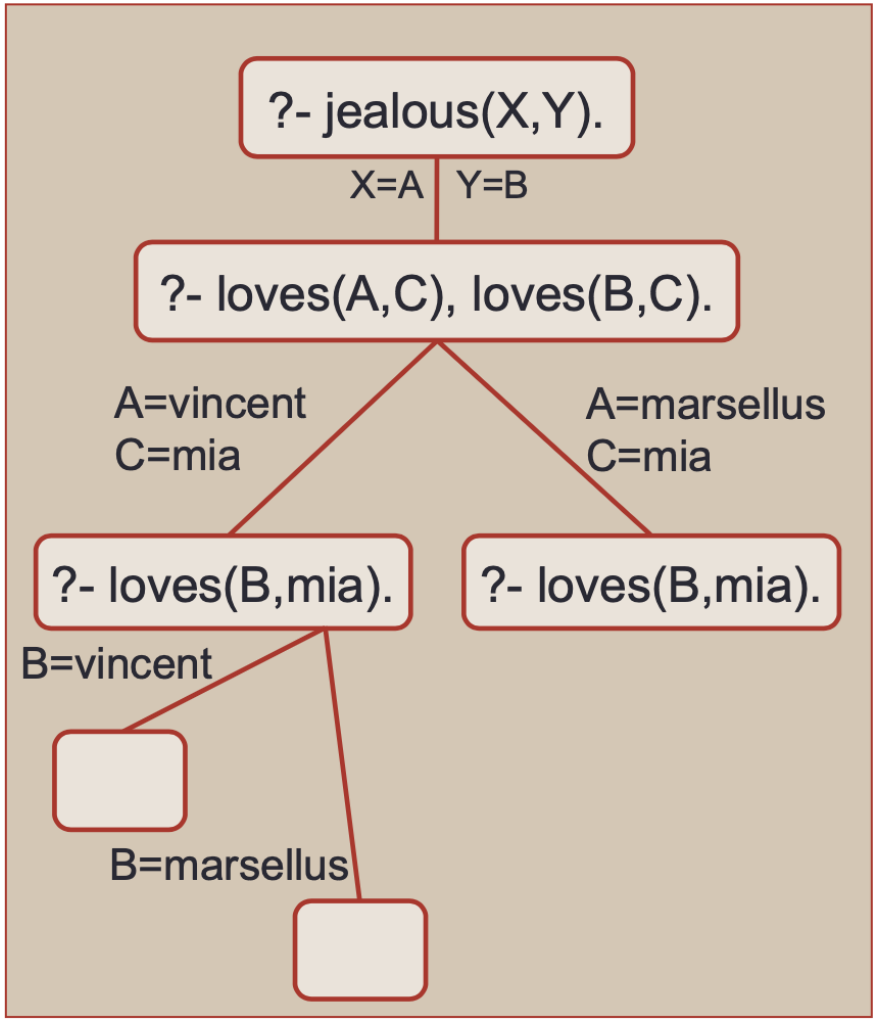
```
?- jealous(X,Y).
X=vincent,
Y=vincent;
X=vincent,
Y=marsellus;
```



Procura - Exemplo 2

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

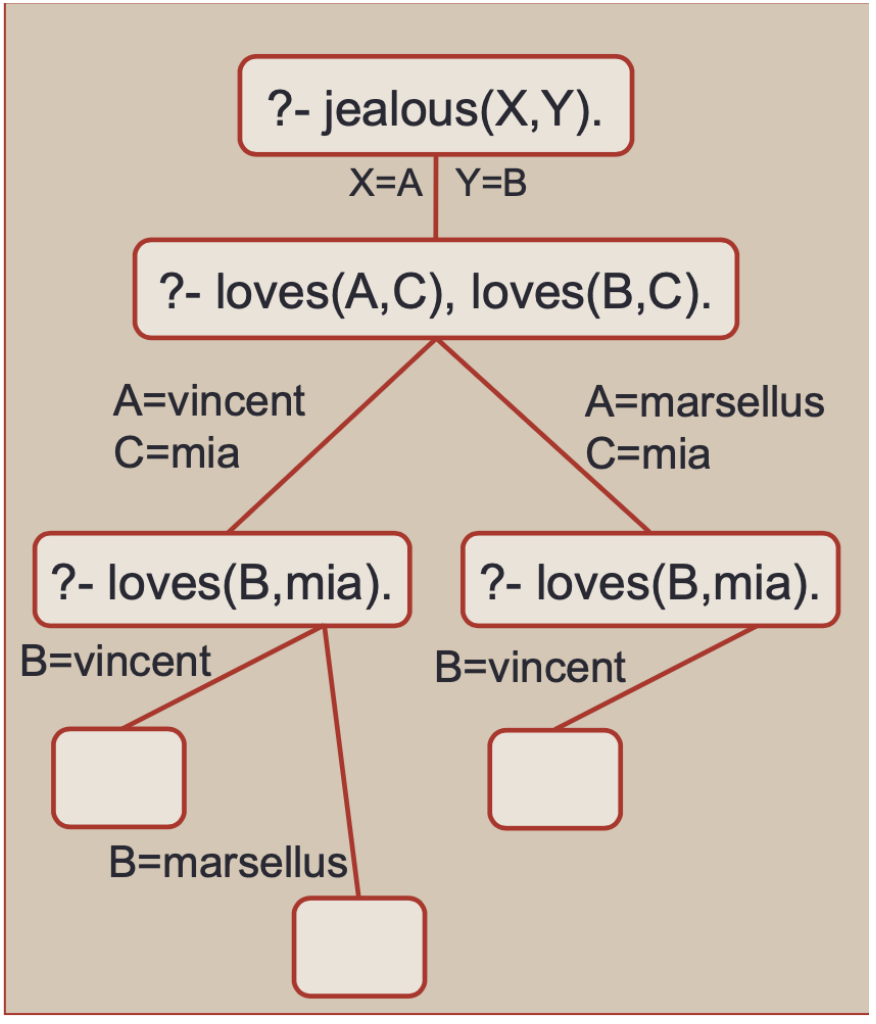
```
?- jealous(X,Y).
X=vincent,
Y=vincent;
X=vincent,
Y=marsellus;
```



Procura - Exemplo 2

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

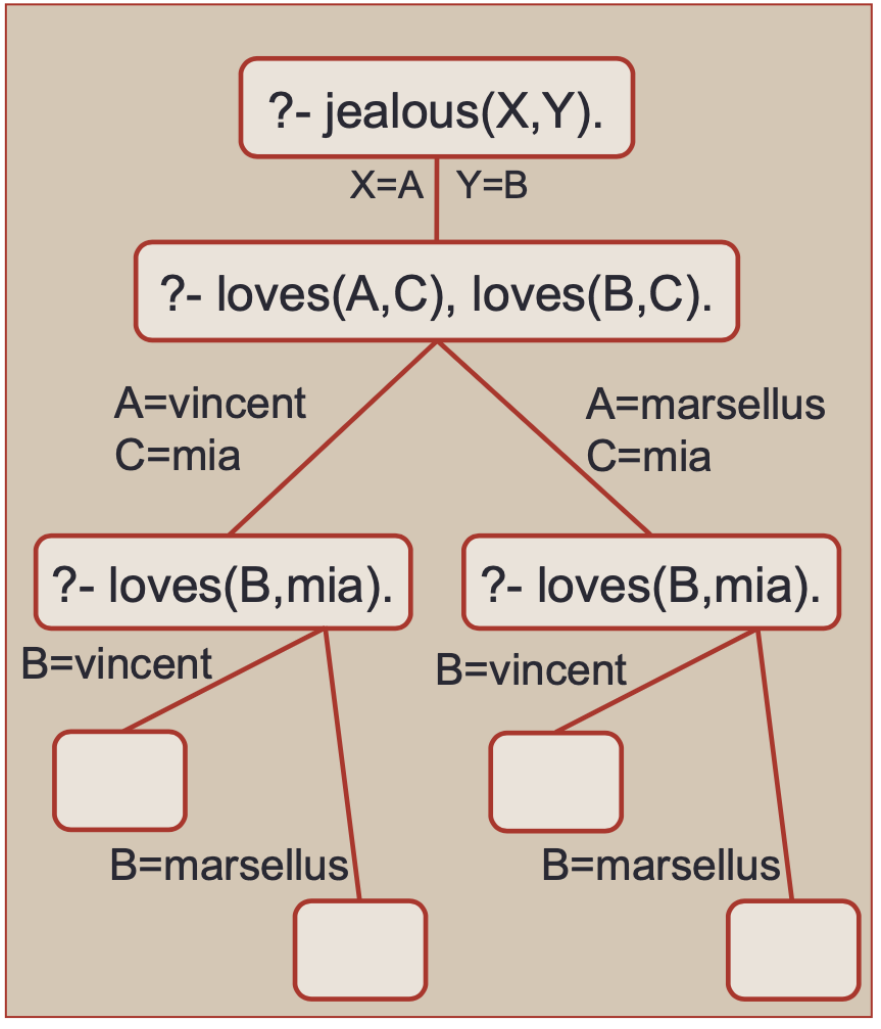
```
?- jealous(X,Y).
X=vincent,
Y=vincent;
X=vincent,
Y=marsellus;
X=marsellus,
Y=vincent;
```



Procura - Exemplo 2

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



Definições Recursivas

- Em Prolog os predicados podem ser definidos recursivamente

Definições Recursivas

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

Definições Recursivas

- Em Prolog os predicados podem ser definidos recursivamente
- Um predicado é definido recursivamente se uma ou mais regras da sua definição se referem a si próprias
- É o caso de definições indutivas de conjuntos e funções

Definição Recursiva - motivação

```
filho(jose,maria).
filho(maria,antonio).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
true
```

Definição Recursiva - motivação

```
filho(jose,maria).
filho(maria,antonio).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
?-descend(jose,antonio).
```

Definição Recursiva - motivação

```
filho(jose,maria).
filho(maria,antonio).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
?-descend(jose,antonio).
true
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
false
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
?-descend(jose,lurdes).
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
?-descend(jose,lurdes).
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,Y).
```

```
?-descend(jose,lurdes).
false
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
true
?-descend(jose,antonio).
false
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
```

Definição Recursiva

filho(**jose**,maria).
filho(maria,antonio).
filho(antonio,**lurdes**).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).

?-descend(jose,lurdes).

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
true
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
false
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
false
```

Como definir o predicado "descend" em geral?

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), filho(Z,W), filho(W,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
false
```

Como definir o predicado "descend" em geral?

Definição Recursiva de "descend".

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
true
?-descend(jose,antonio).
true
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).— caso base
descend(X,Y):- filho(X,Z), descend(Z,Y).
— passo
```

```
true
?-descend(jose,antonio).
true
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(jose,lurdes).
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(jose,lurdes).
true
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
true
```

Definição Recursiva

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(jose,lurdes).
true
?-descend(jose,antonio).
true
```

Exercício: fazer a árvore de procura para cada interrogações.

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"

true

Definição Recursiva - um exemplo já visto

1. “Pedro I é um ascendente directo de João I"
2. “Joao I é um ascendente directo de Duarte I"

true

Definição Recursiva - um exemplo já visto

1. “Pedro I é um ascendente directo de João I"
2. “Joao I é um ascendente directo de Duarte I"
3. “Um ascendente directo de uma pessoa é seu antepassado"

true

Definição Recursiva - um exemplo já visto

1. “Pedro I é um ascendente directo de João I"
2. “Joao I é um ascendente directo de Duarte I"
3. “Um ascendente directo de uma pessoa é seu antepassado"
4. “Um ascendente directo de um antepassado de uma pessoa é seu antepassado"

true

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"
2. "Joao I é um ascendente directo de Duarte I"
3. "Um ascendente directo de uma pessoa é seu antepassado"
4. "Um ascendente directo de um antepassado de uma pessoa é seu antepassado"
5. Será que "Pedro I é um antepassado de Duarte I"?

true

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"
2. "Joao I é um ascendente directo de Duarte I"
3. "Um ascendente directo de uma pessoa é seu antepassado"
4. "Um ascendente directo de um antepassado de uma pessoa é seu antepassado"
5. Será que "Pedro I é um antepassado de Duarte I"?

true

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"
2. "Joao I é um ascendente directo de Duarte I"
3. "Um ascendente directo de uma pessoa é seu antepassado"
4. "Um ascendente directo de um antepassado de uma pessoa é seu antepassado"
5. Será que "Pedro I é um antepassado de Duarte I"?

Queremos verificar que a Interrogação (5) é consequência da Base de Conhecimento (1)-(4).

true

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"
2. "Joao I é um ascendente directo de Duarte I"
3. "Um ascendente directo de uma pessoa é seu antepassado"
4. "Um ascendente directo de um antepassado de uma pessoa é seu antepassado"
5. Será que "Pedro I é um antepassado de Duarte I"?

Queremos verificar que a Interrogação (5) é consequência da Base de Conhecimento (1)-(4).
Escrevemos os factos e as regras da Base de Conhecimento.

Base de Conhecimento

```
ad(pedroI,joaoI).
ad(joaoI,duarteI).
ant(X,Y):- ad(X,Y).
ant(X,Z):- ad(X,Y), ant(Y,Z).
```

```
true
```

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"
2. "Joao I é um ascendente directo de Duarte I"
3. "Um ascendente directo de uma pessoa é seu antepassado"
4. "Um ascendente directo de um antepassado de uma pessoa é seu antepassado"
5. Será que "Pedro I é um antepassado de Duarte I"?

Queremos verificar que a Interrogação (5) é consequência da Base de Conhecimento (1)-(4).
Escrevemos os factos e as regras da Base de Conhecimento.

Base de Conhecimento

```
ad(pedroI,joaoI).
ad(joaoI,duarteI).
ant(X,Y):- ad(X,Y).
ant(X,Z):- ad(X,Y), ant(Y,Z).
```

Interrogação

```
?- ant(pedroI,duarteI).
```

Definição Recursiva - um exemplo já visto

1. "Pedro I é um ascendente directo de João I"
2. "Joao I é um ascendente directo de Duarte I"
3. "Um ascendente directo de uma pessoa é seu antepassado"
4. "Um ascendente directo de um antepassado de uma pessoa é seu antepassado"
5. Será que "Pedro I é um antepassado de Duarte I"?

Queremos verificar que a Interrogação (5) é consequência da Base de Conhecimento (1)-(4).
Escrevemos os factos e as regras da Base de Conhecimento.

Base de Conhecimento

```
ad(pedrol,joaol).
ad(joaol,duartel).
ant(X,Y):- ad(X,Y).
ant(X,Z):- ad(X,Y), ant(Y,Z).
```

Interrogação

```
?- ant(pedrol,duartel).
true
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural

```
true
?- nat(X).
X=0;
X=s(0);
X=s(s(0));
...
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
true
?- nat(X).
X=0;
X=s(0);
X=s(s(0));
...
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
true
?- nat(X).
X=0;
X=s(0);
X=s(s(0));
...
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
nat(0).
nat(s(N)):- nat(N).
```

```
true
?- nat(X).
X=0;
X=s(0);
X=s(s(0));
...
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
nat(0).
nat(s(N)):- nat(N).
```

```
?- nat(s(s(s(0)))).
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
nat(0).
nat(s(N)):- nat(N).
```

```
?- nat(s(s(s(0)))).
true
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
nat(0).
nat(s(N)):- nat(N).
```

```
?- nat(s(s(s(0)))).
true
?- nat(X).
```

Definição Recursiva - Números naturais

Recordar definição indutiva dos números naturais:

- 0 é um natural
- se N é um natural, então s(N) também é.

```
nat(0).
nat(s(N)):- nat(N).
```

```
?- nat(s(s(s(0)))).
true
?- nat(X).
X=0;
X=s(0);
X=s(s(0));
...
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(s(N),M,):- soma(N,M,Z).
```

```
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
X=Y, Y=s(0);
X=s(s(0))
Y=0.
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
```

```
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
X=Y, Y=s(0);
X=s(s(0))
Y=0.
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,?):- soma(N,M,Z).
```

```
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
X=Y, Y=s(0);
X=s(s(0))
Y=0.
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
X=Y, Y=s(0);
X=s(s(0))
Y=0.
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
?- soma(s(0),s(s(0)),X).
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
?- soma(s(0),s(s(0)),X).
X=s(s(s(0)))
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
?- soma(s(0),s(s(0)),X).
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
?- soma(s(0),s(s(0)),X).
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
?- soma(s(0),s(s(0)),X).
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
X=Y, Y=s(0);
```

Definição Recursiva - Soma de números naturais

Assumindo esta representação dos números naturais, como se define a soma de dois números?

```
soma(0,N,N).
soma(s(N),M,s(Z)):- soma(N,M,Z).
```

```
?- soma(s(0),s(s(0)),X).
X=s(s(s(0)))
?- soma(X,Y,s(s(0))).
X=0,
Y=s(s(0));
X=Y, Y=s(0);
X=s(s(0))
Y=0.
```

Prolog - Linguagem declarativa?

- O Prolog foi a primeira tentativa de criar uma linguagem de programação em lógica

Prolog - Linguagem declarativa?

- 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

Prolog - Linguagem declarativa?

- 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

Prolog - Linguagem declarativa?

- 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 - Linguagem declarativa?

- O Prolog dá alguns passos importantes nesta direcção, mas...

Prolog - Linguagem declarativa?

- O Prolog dá alguns passos importantes nesta direcção, mas...

Prolog - Linguagem declarativa?

- O Prolog dá alguns passos importantes nesta direcção, mas... não é uma pura linguagem de programação em lógica!

Prolog - Linguagem declarativa?

- 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:

Prolog - Linguagem declarativa?

- 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

Prolog - Linguagem declarativa?

- 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

Prolog - Linguagem declarativa?

- 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 fazer escolhas alternativas

Prolog - Linguagem declarativa?

Descendentes - Versão 1

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago.
```

Prolog - Linguagem declarativa?

Descendentes - Versão 1

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago.
```

Prolog - Linguagem declarativa?

Descendentes - Versão 1

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

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

Prolog - Linguagem declarativa?

Descendentes - Versão 1

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(A,B).
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago.
```

Prolog - Linguagem declarativa?

Descendentes - Versão 1

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- filho(X,Z), descend(Z,Y).
```

```
?-descend(A,B).
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago.
```

O que acontece se trocarmos a ordem das regras?

Prolog - Linguagem declarativa?

Descendentes - Versão 2

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Z), descend(Z,Y).
descend(X,Y):- filho(X,Y).
```

A=jose, B=tiago;
A=jose, B=lurdes;
A=jose, B=antonio;
...
A=lurdes, B=tiago.

Prolog - Linguagem declarativa?

Descendentes - Versão 2

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Z), descend(Z,Y).
descend(X,Y):- filho(X,Y).
```

A=jose, B=tiago;
A=jose, B=lurdes;
A=jose, B=antonio;
...
A=lurdes, B=tiago.

Prolog - Linguagem declarativa?

Descendentes - Versão 2

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Z), descend(Z,Y).
descend(X,Y):- filho(X,Y).
```

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

Prolog - Linguagem declarativa?

Descendentes - Versão 2

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Z), descend(Z,Y).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
A=jose, B=tiago;
A=jose, B=lurdes;
A=jose, B=antonio;
...
A=lurdes, B=tiago.
```

Prolog - Linguagem declarativa?

Descendentes - Versão 2

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Z), descend(Z,Y).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
A=jose, B=tiago;
A=jose, B=lurdes;
A=jose, B=antonio;
...
A=lurdes, B=tiago.
```

Trocámos a ordem das duas regras e obtivemos os mesmos resultados, mas por ordem diferente.

Prolog - Linguagem declarativa?

Descendentes - Versão 2

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Z), descend(Z,Y).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
A=jose, B=tiago;
A=jose, B=lurdes;
A=jose, B=antonio;
...
A=lurdes, B=tiago.
```

Trocámos a ordem das duas regras e obtivemos os mesmos resultados, mas por ordem diferente.

E se trocarmos a ordem no corpo das regras?

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

Stack limit (..Gb) exceeded

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

Stack limit (..Gb) exceeded

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

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

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
Stack limit (..Gb) exceeded
```

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
Stack limit (..Gb) exceeded
```

A procura entra num ciclo infinito. Porquê?

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
Stack limit (..Gb) exceeded
```

A procura entra num ciclo infinito. Porquê?
Para responder a **?-descend(A,B).** o Prolog vai usar a primeira regra.

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
Stack limit (..Gb) exceeded
```

A procura entra num ciclo infinito. Porquê?
Para responder a **?-descend(A,B).** o Prolog vai usar a primeira regra.
Logo, o próximo objetivo é satisfazer **?-descend(Z1,B).**

Prolog - Linguagem declarativa?

Descendentes - Versão 3

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- descend(Z,Y), filho(X,Z).
descend(X,Y):- filho(X,Y).
```

```
?-descend(A,B).
Stack limit (..Gb) exceeded
```

A procura entra num ciclo infinito. Porquê?
Para responder a **?-descend(A,B).** o Prolog vai usar a primeira regra.
Logo, o próximo objetivo é satisfazer **?-descend(Z1,B).**
Logo, o próximo objetivo é satisfazer **?-descend(Z2,B).** ....

Prolog - Linguagem declarativa?

Descendentes - Versão 4

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- descend(Z,Y), filho(X,Z).
```

```
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago;
Stack limit (..Gb) exceeded
```

Prolog - Linguagem declarativa?

Descendentes - Versão 4

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- descend(Z,Y), filho(X,Z).
```

```
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago;
Stack limit (..Gb) exceeded
```

Prolog - Linguagem declarativa?

Descendentes - Versão 4

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- descend(Z,Y), filho(X,Z).
```

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

Prolog - Linguagem declarativa?

Descendentes - Versão 4

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- descend(Z,Y), filho(X,Z).
```

```
?-descend(A,B).
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago;
Stack limit (..Gb) exceeded
```

Prolog - Linguagem declarativa?

Descendentes - Versão 4

```
filho(jose,maria).
filho(maria,antonio).
filho(antonio,lurdes).
filho(lurdes,tiago).
descend(X,Y):- filho(X,Y).
descend(X,Y):- descend(Z,Y), filho(X,Z).
```

```
?-descend(A,B).
A=jose, B=maria;
A=maria, B=antonio;
A=antonio, B=lurdes;
...
A=antonio, B=tiago;
Stack limit (..Gb) exceeded
```

A troca na ordem das regras permite obter algumas (neste caso todas) respostas antes de a procura entrar num ciclo infinito.

Prolog - Linguagem declarativa?

Prolog é uma linguagem puramente declarativa?

Prolog - Linguagem declarativa?

Prolog é uma linguagem puramente declarativa?

Não! A ordem interessa.

Prolog - Linguagem declarativa?

Prolog é uma linguagem puramente declarativa?

Não! A ordem interessa.

As quatro versões dos Descendentes são equivalentes em termos lógicos, mas...

Prolog - Linguagem declarativa?

Prolog é uma linguagem puramente declarativa?

Não! A ordem interessa.

As quatro versões dos Descendentes são equivalentes em termos lógicos, mas...

O Prolog tem um comportamento diferente em cada um deles.

Prolog - Linguagem declarativa?

Prolog é uma linguagem puramente declarativa?

Não! A ordem interessa.

As quatro versões dos Descendentes são equivalentes em termos lógicos, mas...

O Prolog tem um comportamento diferente em cada um deles.

A diferença pode mesmo ser grande: ciclos infinitos!

Prolog e Recursividade

Em regras recursivas, a ordem dos elementos no corpo pode fazer toda a diferença.

Prolog e Recursividade

Em regras recursivas, a ordem dos elementos no corpo pode fazer toda a diferença.

É preciso entender o processo de procura do Prolog.

Prolog e Recursividade

Em regras recursivas, a ordem dos elementos no corpo pode fazer toda a diferença.

É preciso entender o processo de procura do Prolog.

Temos de evitar as **regras recursivas à esquerda**:

Prolog e Recursividade

Em regras recursivas, a ordem dos elementos no corpo pode fazer toda a diferença.

É preciso entender o processo de procura do Prolog.

Temos de evitar as **regras recursivas à esquerda**:

aquelas em que o elemento mais à esquerda do seu corpo é igual à cabeça da regra (a menos da escolha das variáveis).

Prolog e Recursividade

Em regras recursivas, a ordem dos elementos no corpo pode fazer toda a diferença.

É preciso entender o processo de procura do Prolog.

Temos de evitar as **regras recursivas à esquerda**:

aquelas em que o elemento mais à esquerda do seu corpo é igual à cabeça da regra (a menos da escolha das variáveis).

descend(X,Y):- **descend(Z,Y)**, filho(X,Z)

Prolog e Recursividade

Em regras recursivas, a ordem dos elementos no corpo pode fazer toda a diferença.

É preciso entender o processo de procura do Prolog.

Temos de evitar as **regras recursivas à esquerda**:

aquelas em que o elemento mais à esquerda do seu corpo é igual à cabeça da regra (a menos da escolha das variáveis).

```
descend(X,Y):- descend(Z,Y), filho(X,Z)
```

Para tentar evitar ciclos infinitos gerados por regras recursivas, devemos colocar as chamadas recursivas o mais à direita possível.

```
descend(X,Y):- filho(X,Z), descend(Z,Y)
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

```
membro(X,[H|T]):- membro(X,T). — passo recursivo
```

```
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
X = lurdes; X = miguel; X = pai(pedro); X = rodrigo.
?- membro(pai(X),[lurdes,miguel,pai(pedro),rodrigo]).
X=pedro.
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples

```
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
X = lurdes; X = miguel; X = pai(pedro); X = rodrigo.
?- membro(pai(X),[lurdes,miguel,pai(pedro),rodrigo]).
X=pedro.
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

```
membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo
```

```
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
X = lurdes; X = miguel; X = pai(pedro); X = rodrigo.
?- membro(pai(X),[lurdes,miguel,pai(pedro),rodrigo]).
X=pedro.
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo

?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

```
membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo
```

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo

?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo

?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

```
membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo
```

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

```
membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo
```

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
X = lurdes; X = miguel; X = pai(pedro); X = rodrigo.
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

```
membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo
```

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
X = lurdes; X = miguel; X = pai(pedro); X = rodrigo.
?- membro(pai(X),[lurdes,miguel,pai(pedro),rodrigo]).
```

Membro de uma lista

Como podemos definir um predicado **membro**/2 que recebe um termo e uma lista e indica se o termo pertence à lista?

membro(X,[X|T]). — caso mais simples
membro(X,[H|T]):- membro(X,T). — passo recursivo

```
?- membro(lurdes,[lurdes,miguel,pai(pedro),rodrigo]).
true
?- membro(pedro,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(jose,[lurdes,miguel,pai(pedro),rodrigo]).
false
?- membro(X,[lurdes,miguel,pai(pedro),rodrigo]).
X = lurdes; X = miguel; X = pai(pedro); X = rodrigo.
?- membro(pai(X),[lurdes,miguel,pai(pedro),rodrigo]).
X=pedro.
```

Listas

- Uma lista é uma sequência finita de elementos

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []
- [lurdes, [max, jose], [rodrigo, triste(rodrigo)]]

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []
- [lurdes, [max, jose], [rodrigo, triste(rodrigo)]]
- [[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []
- [lurdes, [max, jose], [rodrigo, triste(rodrigo)]]
- [[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

- O comprimento de uma lista é o seu número de elementos

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []
- [lurdes, [max, jose], [rodrigo, triste(rodrigo)]]
- [[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

- O comprimento de uma lista é o seu número de elementos
- Qualquer termo Prolog pode ser um elemento de uma lista

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []
- [lurdes, [max, jose], [rodrigo, triste(rodrigo)]]
- [[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

- O comprimento de uma lista é o seu número de elementos
- Qualquer termo Prolog pode ser um elemento de uma lista
- Existe uma lista especial: a lista vazia [ ]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes

```
  Head: lurdes
  Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
  Head: [ ]
  Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
  Head: triste(z)
  Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça

```
  Head: lurdes
  Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
  Head: [ ]
  Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
  Head: triste(z)
  Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto

```
  Head: lurdes
  Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
  Head: [ ]
  Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
  Head: triste(z)
  Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista

```
  Head: lurdes
  Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
  Head: [ ]
  Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
  Head: triste(z)
  Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento

```
  Head: lurdes
  Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
  Head: [ ]
  Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
  Head: triste(z)
  Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

```
Head: lurdes
Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]
Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
Head: triste(z)
Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

```
Head: lurdes
Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]
Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]
Head: triste(z)
Tail: [ ]
```

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]

Head: lurdes

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]

[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]

[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]
Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]
[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]
Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
[triste(z)]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]

[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]
Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

[triste(z)]
Head: triste(z)
Tail: [ ]

Cabeça e resto da lista vazia

- A lista vazia não tem nem cabeça nem resto

Cabeça e resto da lista vazia

- A lista vazia não tem nem cabeça nem resto
- Em Prolog, [ ] é uma lista simples especial sem nenhuma estrutura interna

Cabeça e resto da lista vazia

- A lista vazia não tem nem cabeça nem resto
- Em Prolog, [ ] é uma lista simples especial sem nenhuma estrutura interna
- A lista vazia tem um papel importante nos predicados recursivos para o processamento de listas em Prolog

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto

```
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false

?- [X,Y|Tail] = [[ ], triste(z), [2, [b,c]], [], Z, [2,[b,c]]]
X=[ ],
Y = triste(z),
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]].
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false

?- [X,Y|Tail] = [[ ], triste(z), [2, [b,c]], [], Z, [2,[b,c]]]
X=[ ],
Y = triste(z),
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]].
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false

?- [X,Y|Tail] = [[ ], triste(z), [2, [b,c]], [], Z, [2,[b,c]]]
X=[ ],
Y = triste(z),
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]].
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
X = lurdes,
Y = [max, jose, maria].
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false

?- [X,Y|Tail] = [[ ], triste(z), [2, [b,c]], [], Z, [2,[b,c]]]
```

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false

?- [X,Y|Tail] = [[ ], triste(z), [2, [b,c]], [], Z, [2,[b,c]]]
X=[ ],
Y = triste(z),
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]].
```

Membro de uma lista

Na verdade, na implementação de Prolog que vamos usar, ao usarmos a Base de Conhecimento tal como indicada no slide anterior, teremos um aviso do sistema.

membro(X,[_|T]):- membro(X,T). — passo recursivo

Membro de uma lista

Na verdade, na implementação de Prolog que vamos usar, ao usarmos a Base de Conhecimento tal como indicada no slide anterior, teremos um aviso do sistema.
A questão é que a variável T da primeira regra e a variável H da segunda regra não são usadas para a unificação, isto é, só ocorrem uma vez.

membro(X,[_|T]):- membro(X,T). — passo recursivo

Membro de uma lista

Na verdade, na implementação de Prolog que vamos usar, ao usarmos a Base de Conhecimento tal como indicada no slide anterior, teremos um aviso do sistema.
A questão é que a variável T da primeira regra e a variável H da segunda regra não são usadas para a unificação, isto é, só ocorrem uma vez.
O sistema indica então o aviso: Singleton variables: [T]

membro(X,[_|T]):- membro(X,T). — passo recursivo

Membro de uma lista

Na verdade, na implementação de Prolog que vamos usar, ao usarmos a Base de Conhecimento tal como indicada no slide anterior, teremos um aviso do sistema.
A questão é que a variável T da primeira regra e a variável H da segunda regra não são usadas para a unificação, isto é, só ocorrem uma vez.
O sistema indica então o aviso: Singleton variables: [T]
Estas variáveis deveriam ser anónimas.

membro(X,[_|T]):- membro(X,T). — passo recursivo

Membro de uma lista

Na verdade, na implementação de Prolog que vamos usar, ao usarmos a Base de Conhecimento tal como indicada no slide anterior, teremos um aviso do sistema.
A questão é que a variável T da primeira regra e a variável H da segunda regra não são usadas para a unificação, isto é, só ocorrem uma vez.
O sistema indica então o aviso: Singleton variables: [T]
Estas variáveis deveriam ser anónimas.
Reescrevendo a Base de Conhecimento, obtemos:

membro(X,[X|_]). — caso mais simples

Membro de uma lista

Na verdade, na implementação de Prolog que vamos usar, ao usarmos a Base de Conhecimento tal como indicada no slide anterior, teremos um aviso do sistema.
A questão é que a variável T da primeira regra e a variável H da segunda regra não são usadas para a unificação, isto é, só ocorrem uma vez.
O sistema indica então o aviso: Singleton variables: [T]
Estas variáveis deveriam ser anónimas.
Reescrevendo a Base de Conhecimento, obtemos:

```
membro(X,[X|_]). — caso mais simples
membro(X,[_|T]):- membro(X,T). — passo recursivo
```

Recursividade em listas

- O predicado membro/2 acede recursivamente aos elementos da lista

Recursividade em listas

- O predicado membro/2 acede recursivamente aos elementos da lista
  - faz alguma coisa na cabeça, e em seguida,

Recursividade em listas

- O predicado membro/2 acede recursivamente aos elementos da lista
  - faz alguma coisa na cabeça, e em seguida,
  - recursivamente faz a mesma coisa na cauda

Recursividade em listas

- O predicado membro/2 acede recursivamente aos elementos da lista
  - faz alguma coisa na cabeça, e em seguida,
  - recursivamente faz a mesma coisa na cauda
- Esta técnica é muito comum em Prolog!

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:

???

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
false
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e

???

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
false
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
false
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
false
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
false
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
?- a2b([a,a,a,a],[b,b,b,b]).
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
?- a2b([a,a,a,a],[b,b,b,b]).
true
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
?- a2b([a,a,a,a],[b,b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
?- a2b([a,a,a,a],[b,b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
?- a2b([a,a,a,a],[b,b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
```

Recursividade em listas

- Definir o predicado **a2b**/2 que recebe duas listas como argumentos e sucede se:
  - o primeiro argumento é uma lista de a's, e
  - o segundo argumento é uma lista de b's exactamente do mesmo tamanho

???

```
?- a2b([a,a,a,a],[b,b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,c,a,a],[b,b,b,t]).
false
```

Recursividade em listas

- Qual o caso base?

```
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base?

```
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!

```
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
```

Recursividade em listas

- Qual o caso base? A lista vazia!
- Como é que, à custa de um par de listas em que o predicado é satisfeito construímos outro par de listas, um pouco mais complexo, em que o predicado também é satisfeito?

```
a2b([],[]). — caso base
a2b([a|L1],[b|L2]):- a2b(L1,L2).
— passo recursivo
```

```
?- a2b([a,a,a],[b,b,b]).
true
?- a2b([a,a,a,a],[b,b,b]).
false
?- a2b([a,t,a,a],[b,b,b,c]).
false
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b] ?-
a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
```

Artimética e Listas

- Qual o tamanho de uma lista?

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia:

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia:

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:1 mais o tamanho do seu resto.

```
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:1 mais o tamanho do seu resto.

len([],0). — caso base

X=7.

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:1 mais o tamanho do seu resto.

```
len([],0). — caso base
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:1 mais o tamanho do seu resto.

```
len([],0). — caso base
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
X=7.
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:1 mais o tamanho do seu resto.

```
len([],0). — caso base
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
?- len([a,b,c,d,e,[a,x],t],X).
```

Artimética e Listas

- Qual o tamanho de uma lista?
  - Tamanho da lista vazia: 0
  - Tamanho de uma lista não-vazia:1 mais o tamanho do seu resto.

```
len([],0). — caso base
len([_|L],N) :- len(L,X), N is X + 1.
— passo recursivo
```

```
?- len([a,b,c,d,e,[a,x],t],X).
X=7.
```