

Linguagens de Programação

Programação Lógica (Prolog)

Andrei Rimsa Álvares

andrei@decom.cefetmg.br

Sumário

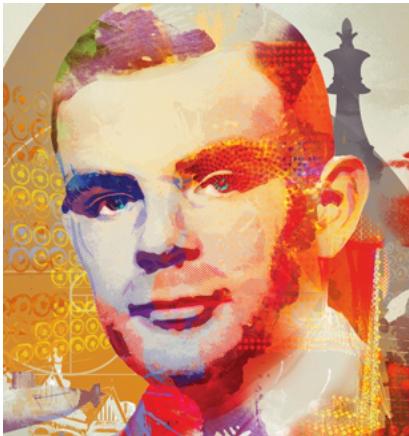
- Introdução
- Prolog
- Unificação
- Mais sobre Prolog
- Resumo

INTRODUÇÃO



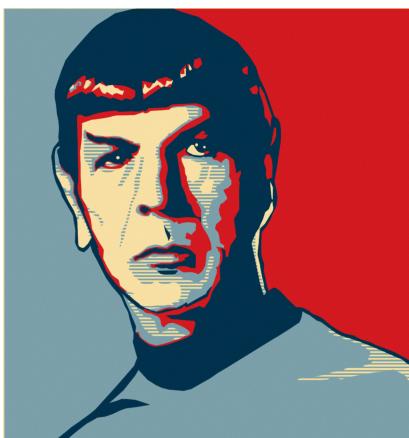
Linguagens de Programação

Programação Imperativa vs. Lógica



Imperativa

- **Programas:** mapeamento de entradas em saídas
- **Modelo computacional:** von Neumann (variáveis, atribuição, comandos, ...)
- **Estilo procedural:** "como fazer"
- **Programação:** instruções que executam comandos
- **Computação:** ações que alteram estado



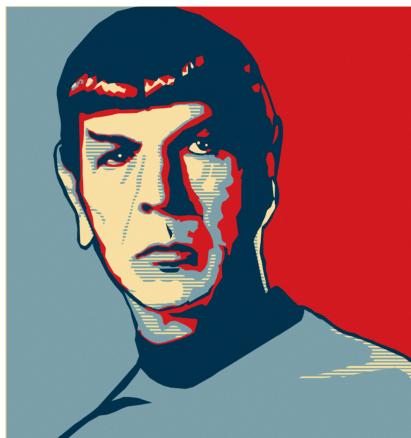
Lógica

- **Programas:** conhecimentos sobre um problema e não uma sequência de passos
- **Modelo computacional:** lógica matemática
- **Estilo declarativo:** "o que fazer"
- **Programação:** simbólica (não numérica)
- **Computação:** se uma consulta pode ser deduzida de relações do programa

Programação Funcional vs. Lógica



Funcional



Lógica

- **Programas:** com mapeamentos
 - Dado a , determine o valor de $m(a)$
(sempre resultará em uma única resposta)

- **Programas:** com relações
 - Dados a e b , verificar se $R(a, b)$ é verdadeiro
 - Dado a , encontrar todos os valores para y tal que $R(a, y)$ é verdadeiro
 - Dado b , encontrar todos os valores para x tal que $R(x, b)$ é verdadeiro
 - Encontre todos os valores para x e y , tal que $R(x, y)$ é verdadeiro

Programação Lógica

- Nenhuma linguagem de programação em lógica pode explorar totalmente o potencial da lógica matemática
 - Pois o formalismo matemático não é implementável
- Exemplo:
 - O *Último Teorema de Fermat* afirma que não existe nenhum conjunto de inteiros positivos x, y, z e n com n maior que 2 que satisfaça a seguinte equação:

$$x^n + y^n = z^n$$

Exemplo

- Hierarquia de uma família

```
avo(X, Y) :- pai(X, Z), pai(Z, Y).  
avo(X, Y) :- pai(X, Z), mae(Z, Y).
```

```
pai(carlos, joao).  
pai(joao, jose).
```

```
mae(maria, joao).
```

```
?- avo(carlos, jose).  
true .
```

```
?- avo(carlos, joao).  
false.
```

```
?- avo(manoel, carlos).  
false.
```

```
?- avo(X, jose).  
X = carlos .
```

PROLOG



Linguagens de Programação

Prolog

- Prolog é uma linguagem de programação lógica de propósito geral associada com inteligência artificial e linguística que possui suas raízes na lógica de primeira ordem, uma lógica formal
- Prolog é declarativa, a lógica do programa é expressa em termos de **relações**, representada como fatos e regras; uma computação é iniciada executando uma pesquisa sobre essas relações
- A linguagem foi inicialmente concebida por um grupo coordenado por Alain Colmerauer em Marseille (França) no começo de 1970
 - Foi uma das primeiras linguagens de programação lógica e se mantém como a mais popular até hoje
- A linguagem é usada para prova de teoremas, sistemas especialistas, processamento de linguagens naturais, ...

Elementos

- Valores
 - Constantes: letras (começa com minúscula), números, texto
 - Ex.: maria, 'maria', 123, "Texto"
 - Estruturas (tuplas rotuladas)
 - Ex.: ponto(5, 6), data(01, 01, 2000)
 - Listas
 - Ex.: [1,2,3], [maria, jose]
- Variáveis (começam com letra maiúscula)
 - Ex.: X, Y, Estudante

Cláusulas de *Horn*

- Cláusulas de *Horn* são um subconjunto das cláusulas da lógica de predicados
 - Forma geral
$$B :- A_1, A_2, \dots, A_n.$$
, onde cada A_i é uma relação da forma $R_i(\dots)$
- Interpretação
 - se as condições A_1, A_2, \dots e A_n forem verdadeiras, então a conclusão B é verdadeira (na forma procedural: **se** condições **então** conclusão)
 - Se algum A_i for falso, não se pode deduzir que B é falso, apenas que não se pode inferir que B é verdadeiro

Cláusulas de *Horn*

- Nomenclatura



- Se $n = 0$
a cláusula é chamada de **fato**



Toda cláusula termina
com . (ponto).

- Se $n \geq 1$
a cláusula é chamada de **regra**

Exemplo

- Exemplo somente com fatos

- Quem gosta de peixe?
?- `gosta(X, peixe).`
`X = maria.`

- Pedro e Maria se gostam?
?- `gosta(maria, pedro),`
`gosta(pedro, maria).`
false.

- Existe algo que Pedro e Maria (ambos) gostem?
?- `gosta(pedro, X), gosta(maria, X).`
`X = vinho .`

```
gosta(maria, peixe).  
gosta(pedro, vinho).  
gosta(maria, vinho).  
gosta(pedro, maria).
```

Outro Exemplo

- Exemplo com fatos e regras

```
estrela(sol).  
estrela(sirius).
```

```
orbita(venus, sol).  
orbita(terra, sol).  
orbita(marte, sol).  
orbita(lua, terra).  
orbita(phobos, marte).  
orbita(deimos, marte).
```

```
planeta(B) :- orbita(B, sol).
```

```
satelite(B) :- orbita(B, P),  
             planeta(P).
```

```
?- orbita(venus, sol).  
true.
```

```
?- orbita(B, marte).  
B = phobos ; B = deimos.
```

```
?- orbita(B, venus).  
false.
```

```
?- planeta(mercurio).  
false.
```

```
?- planeta(X).  
X = venus ; X = terra ; X = marte.
```

```
?- satelite(X).  
X = lua ; X = phobos ; X = deimos.
```

UNIFICAÇÃO



Linguagens de Programação

Unificação

- Unificação é o nome do algoritmo usado por Prolog para determinar se existe uma maneira de instanciar as variáveis de dois predicados de modo a torná-los iguais.

Predicado 1	Predicado 2	Unificação
$p(X, Y)$	$q(X, Y)$	false
$p(X, Y)$	$p(joao, jose)$	$X = joao, Y = jose$
$p(X, Y)$	$p(joao, Z)$	$X = joao, Y = Z$
$p(X, X)$	$p(1, 1)$	$X = 1$
$p(X, X)$	$p(1, W)$	$X = 1, W = 1$
$p(X, X)$	$p(1, 2)$	false
$p(X, X, 2)$	$p(1, W, W)$	false
$p(G, jose)$	$p(X, Y)$	$G = X, Y = jose$

Algoritmo

- O algoritmo de execução
 - Faz uma busca em profundidade em uma árvore de pesquisa, com resolução de um objetivo, onde a lista de cláusulas é pesquisada de cima para baixo
 - Se a unificação com o lado esquerdo de uma cláusula **for bem sucedida**, então tenta-se resolver os subobjetivos do lado direito dessa cláusula (da esquerda para a direita)
 - Pode haver retrocessos (backtracking) caso uma unificação **não seja bem sucedida**, tentando explorar outras unificações alternativas
 - Se todas as alternativas forem exploradas sem sucesso, então a resolução falha

A resolução é sujeita à ordem em que as cláusulas foram escritas.

Exemplo

- Verificar se João e Mario são irmãos

```
homem(mario).
```

```
mae(joao, roberta).  
mae(mario, roberta).
```

```
pai(joao, paulo).  
pai(mario, paulo).
```

```
pais(X, M, P) :- mae(X, M),  
                  pai(X, P).
```

```
irmao(X, Y) :- homem(Y),  
                  pais(X, M, P),  
                  pais(Y, M, P).
```

```
?- irmao(joao, mario).
```

Qual a resposta
dessa consulta?

Exemplo

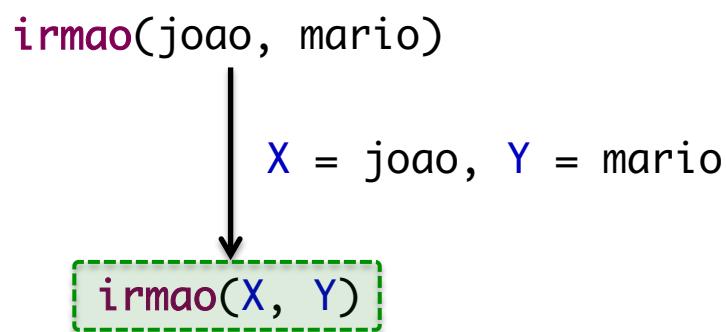
- Execução do algoritmo

irmao(joao, mario)

Com qual regra essa consulta pode ser unificada?

Exemplo

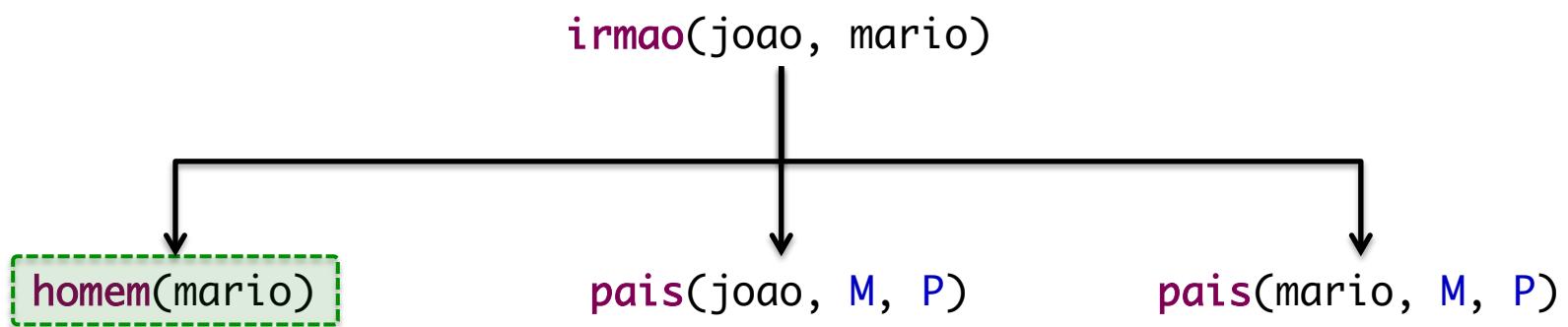
- Execução do algoritmo



Expandir a regra $\text{irmao}(X, Y)$ com suas cláusulas à direita.

Exemplo

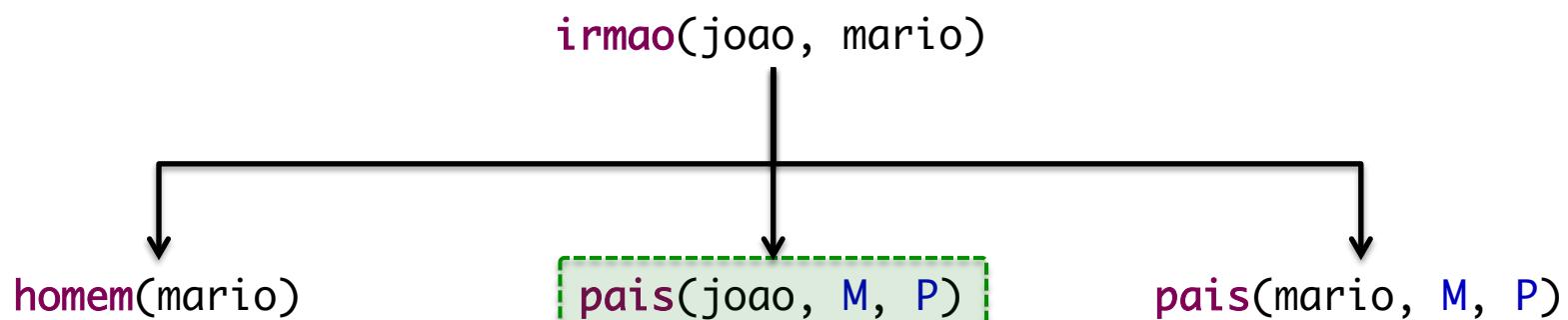
- Execução do algoritmo



A cláusula `homem(mario)` é um fato.

Exemplo

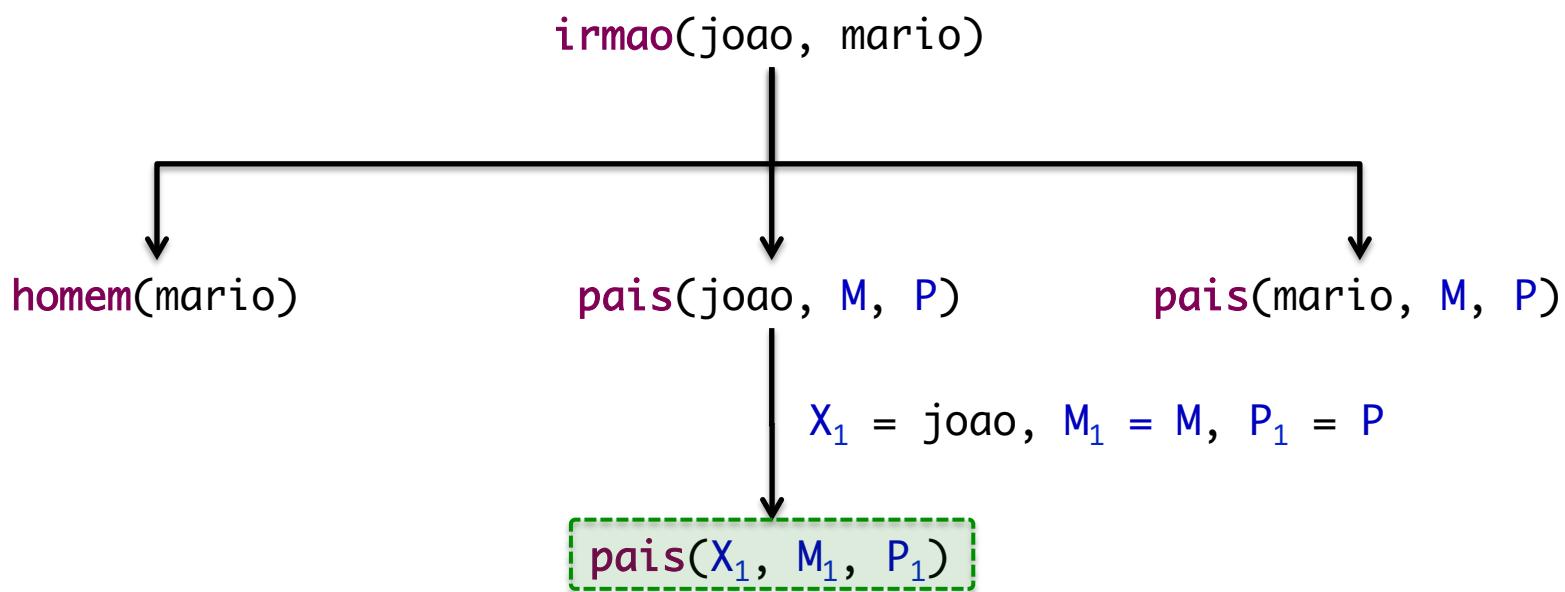
- Execução do algoritmo



Qual regra pode ser unificada
nesse momento?

Exemplo

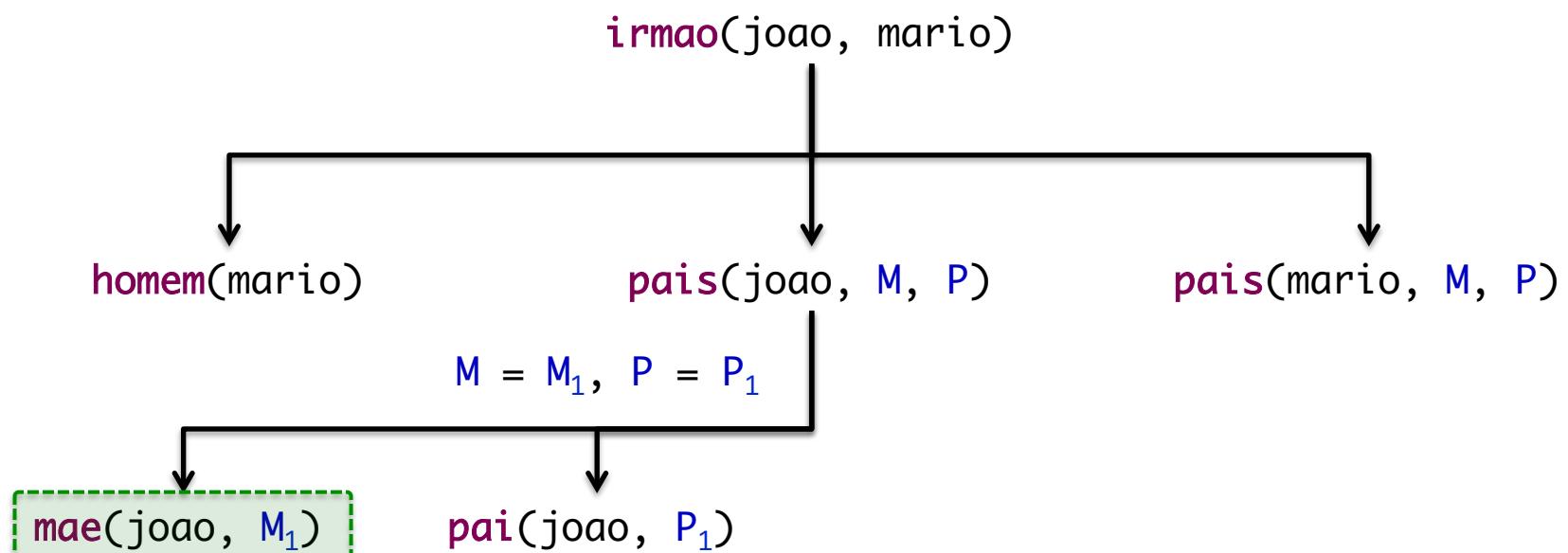
- Execução do algoritmo



Expandir a regra $\text{pais}(X_1, M_1, P_1)$ com suas cláusulas à direita.

Exemplo

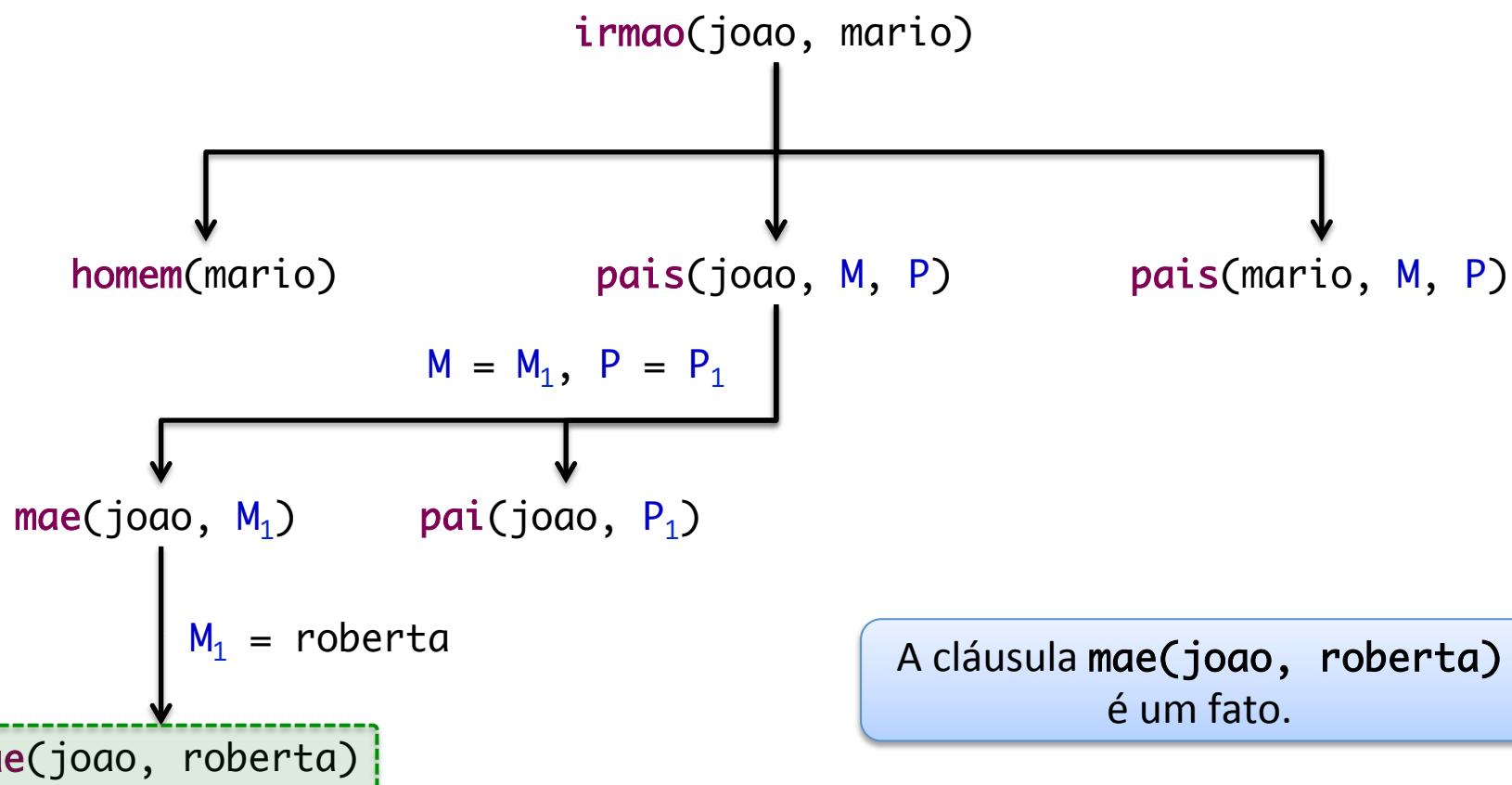
- Execução do algoritmo



Qual regra pode ser unificada
nesse momento?

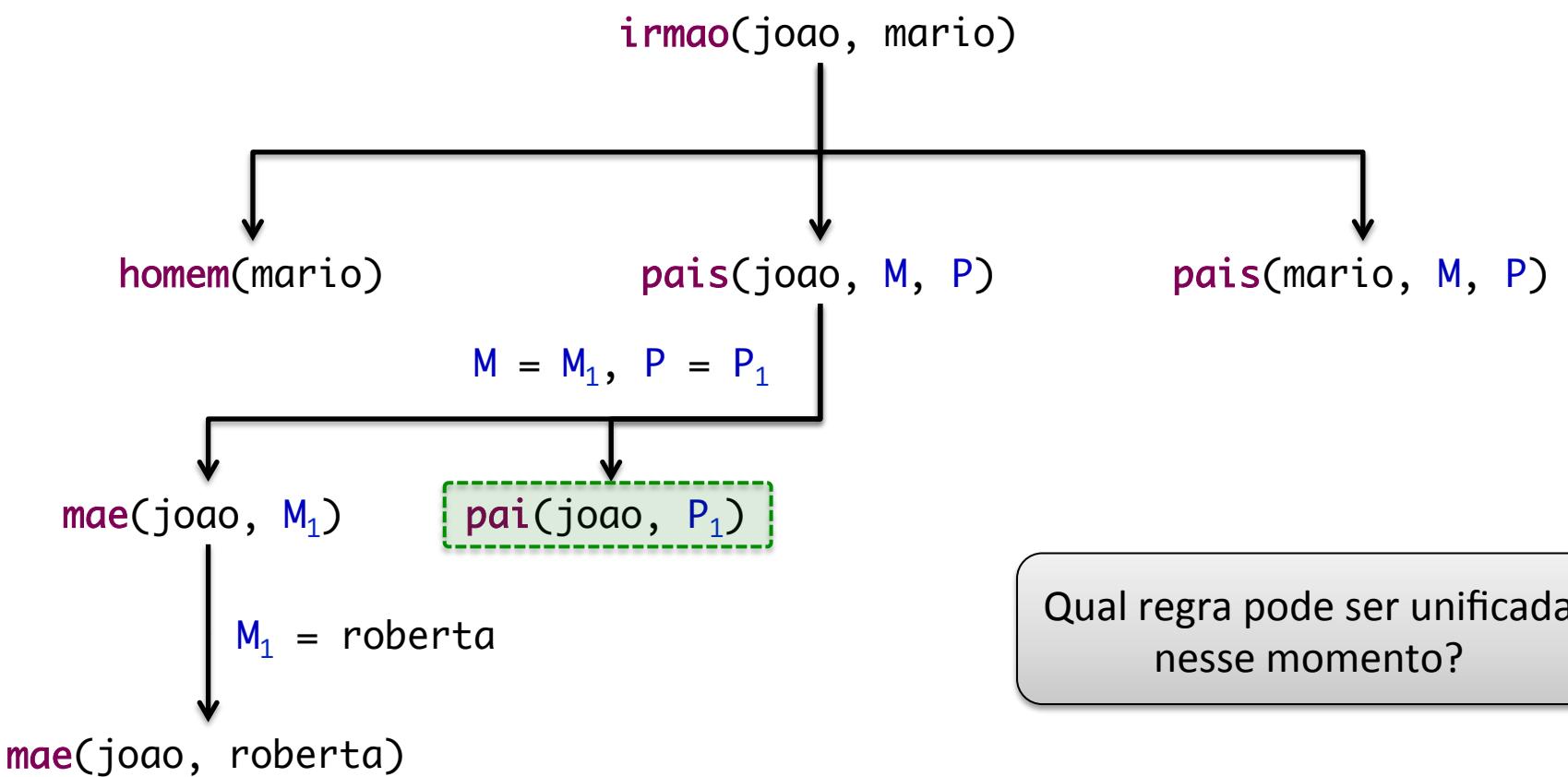
Exemplo

- Execução do algoritmo



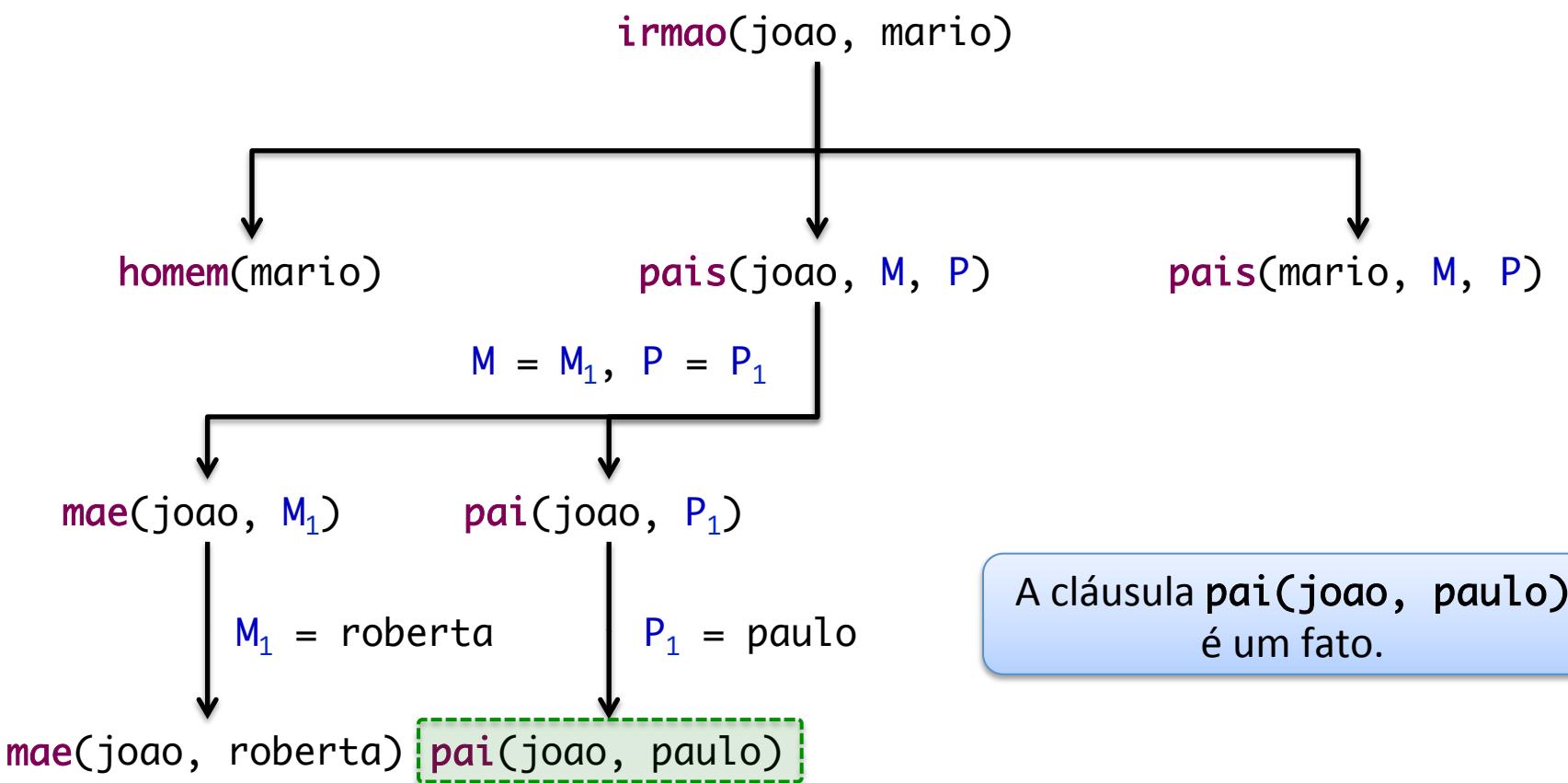
Exemplo

- Execução do algoritmo



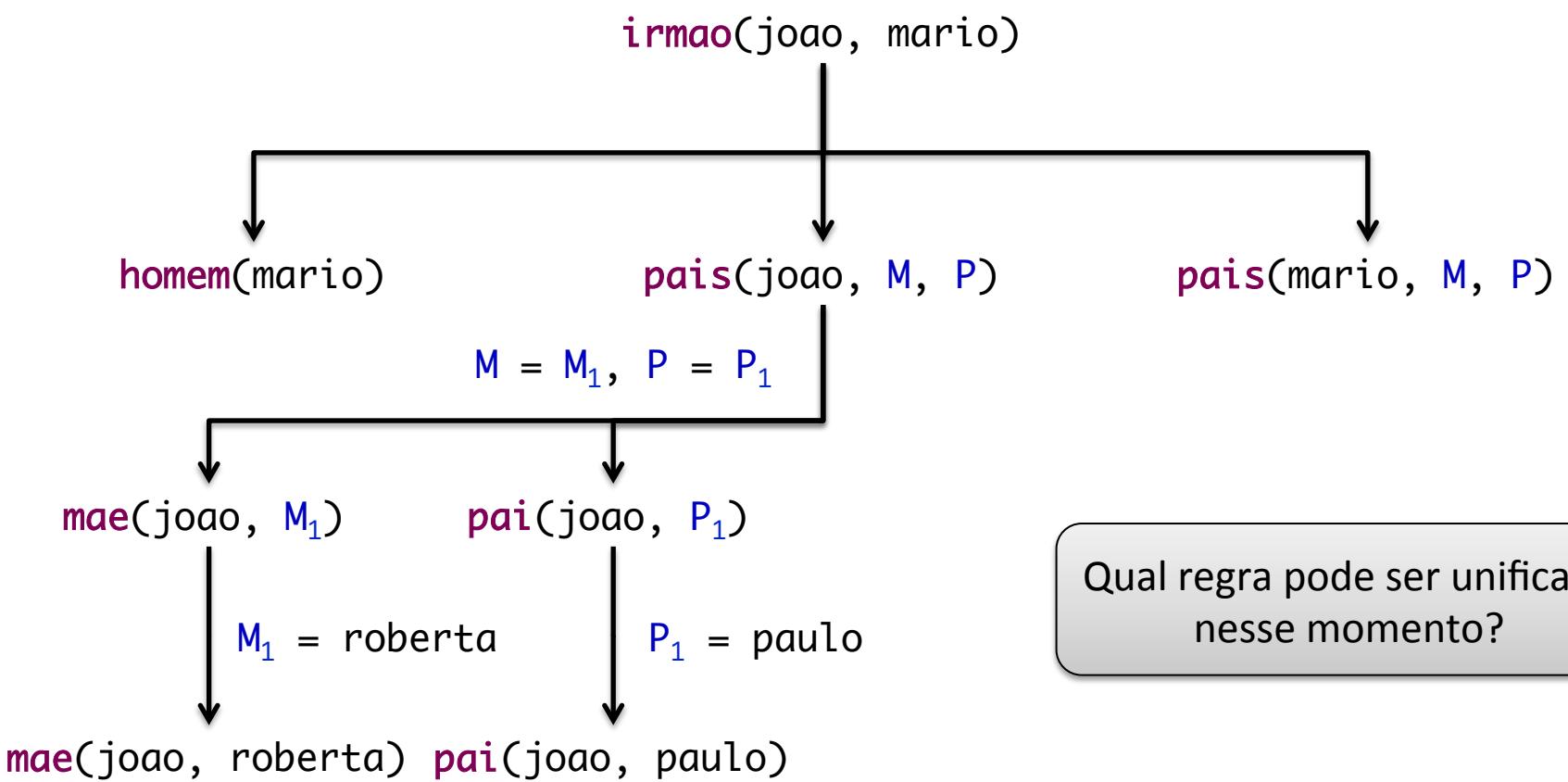
Exemplo

- Execução do algoritmo



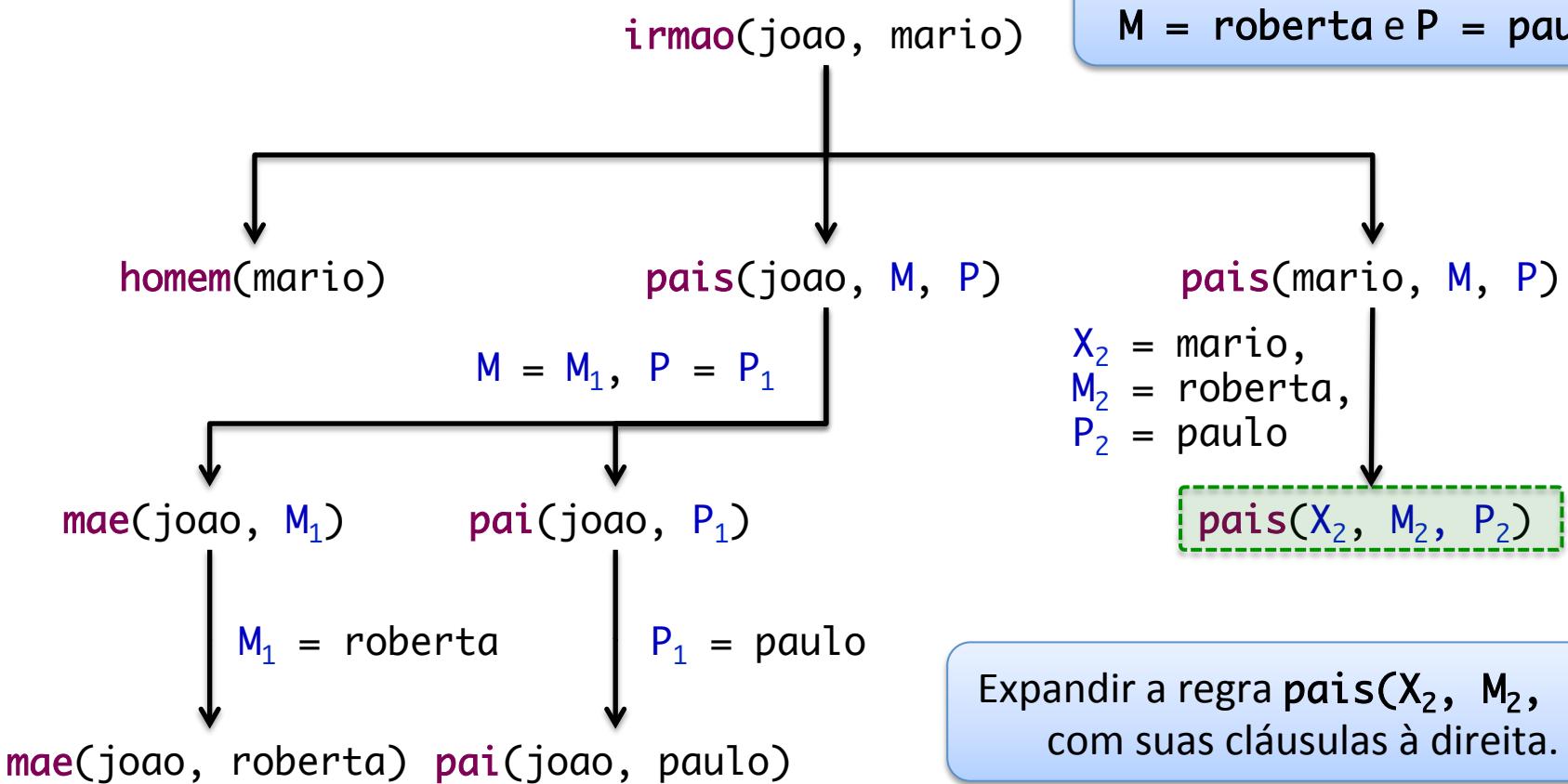
Exemplo

- Execução do algoritmo



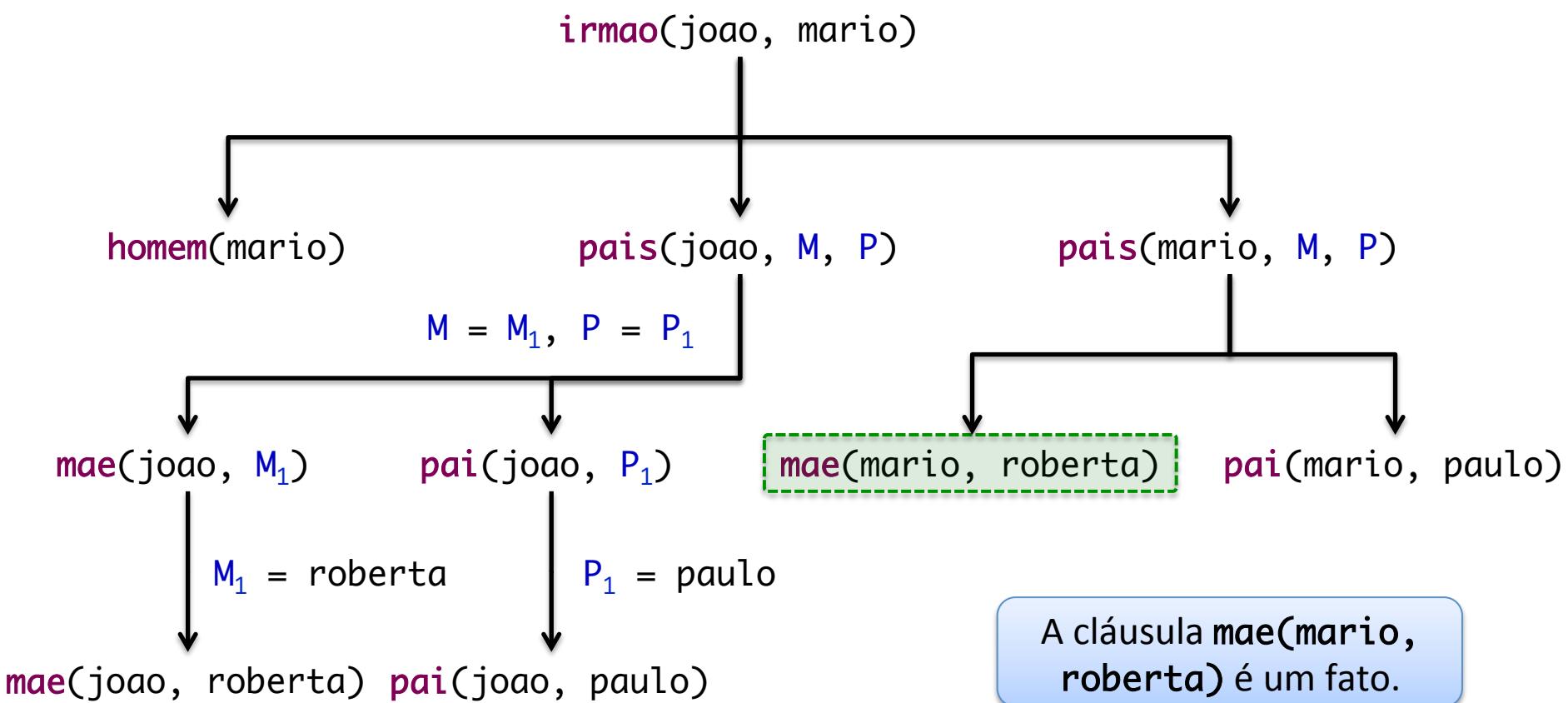
Exemplo

- Execução do algoritmo



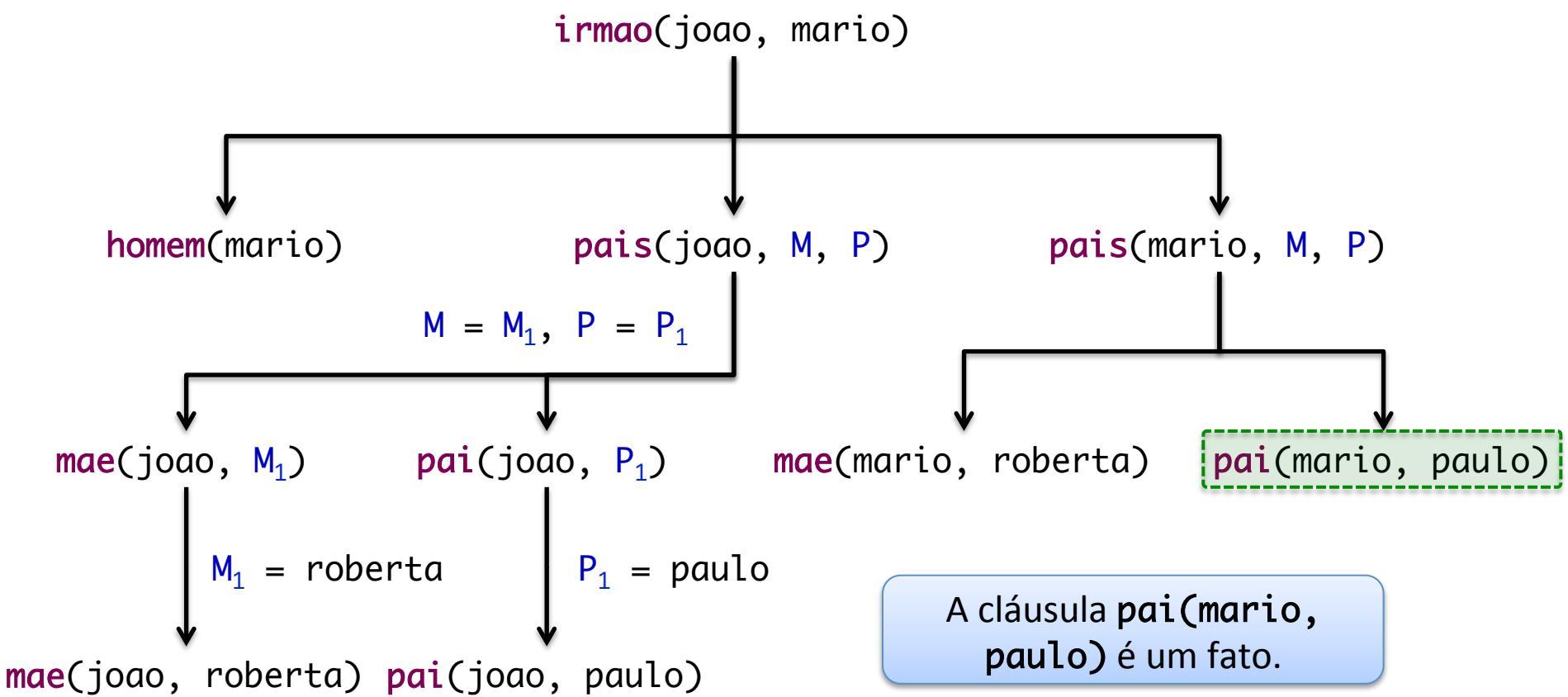
Exemplo

- Execução do algoritmo



Exemplo

- Execução do algoritmo

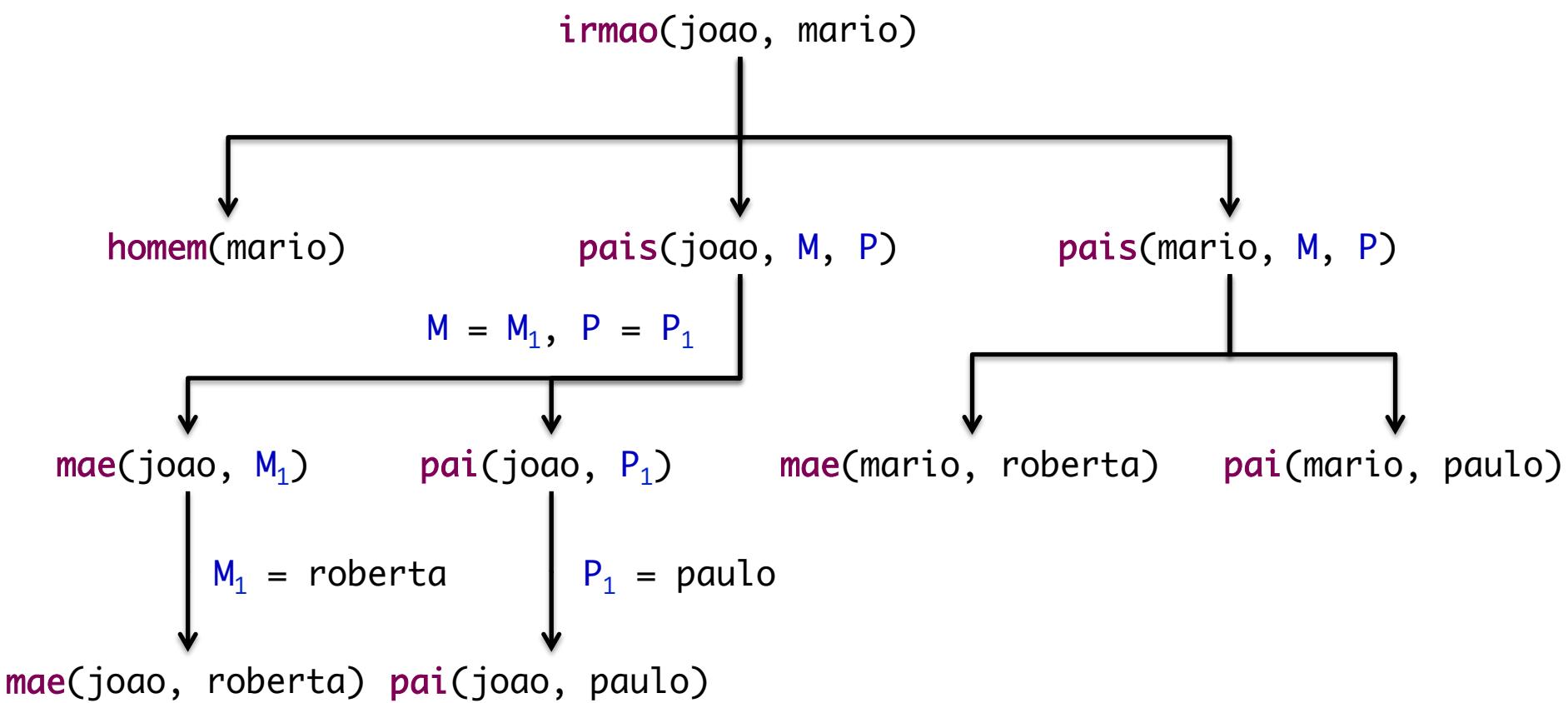




Exemplo

- Execução do algoritmo

Todas as cláusulas foram unificadas, portanto a resposta é true.



Outro Exemplo

- Consultar quem é muito inteligente

```
muito_inteligente(X) :- inteligente(X),  
                     simpatico(X).
```

```
inteligente(X) :- vivo(X), humano(X).
```

```
inteligente(X) :- golfinho(X).
```

```
humano(X) :- homem(X).
```

```
humano(X) :- mulher(X).
```

```
vivo(chico).
```

```
golfinho(flipper).
```

```
mulher(maria).
```

```
homem(chico).
```

```
simpatico(flipper).
```

```
?- muito_inteligente(X).
```

Qual a resposta
dessa consulta?

Outro Exemplo

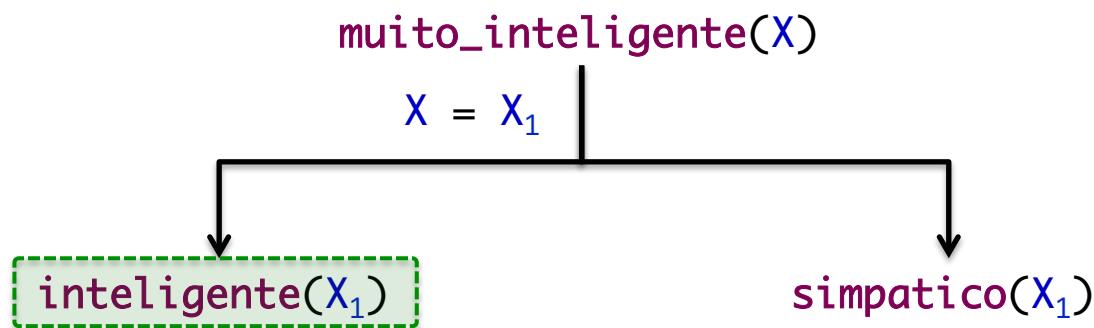
- Execução do algoritmo

`muito_inteligente(X)`

Com qual regra essa consulta pode ser unificada?

Outro Exemplo

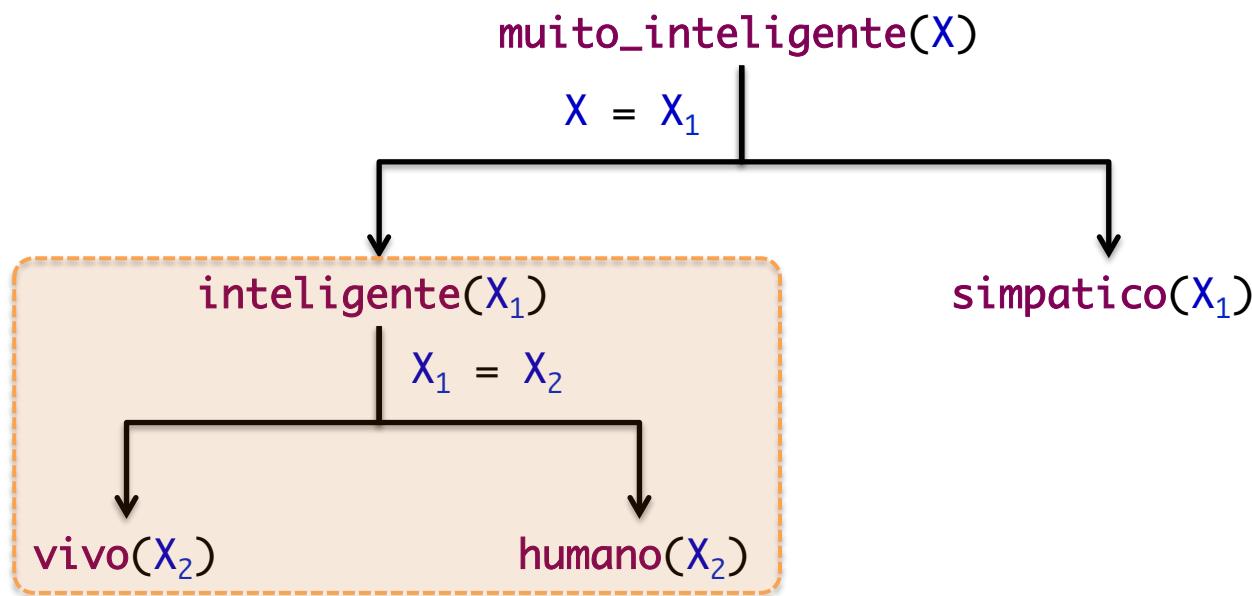
- Execução do algoritmo



Quais regras podem ser
unificadas agora?

Outro Exemplo

- Execução do algoritmo

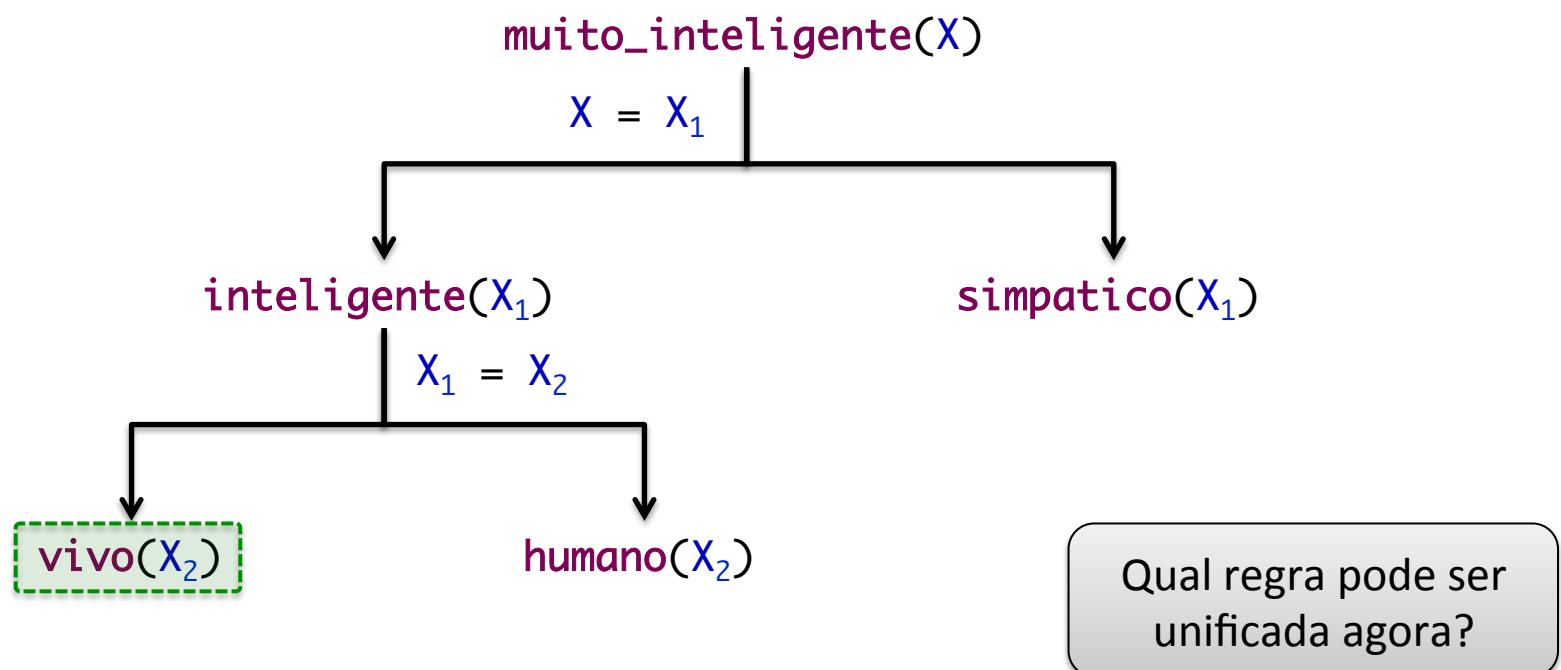


Obs: existem duas regras que podem ser unificadas para $\text{inteligente}(X_1)$, expandiu-se a primeira (na ordem do programa)



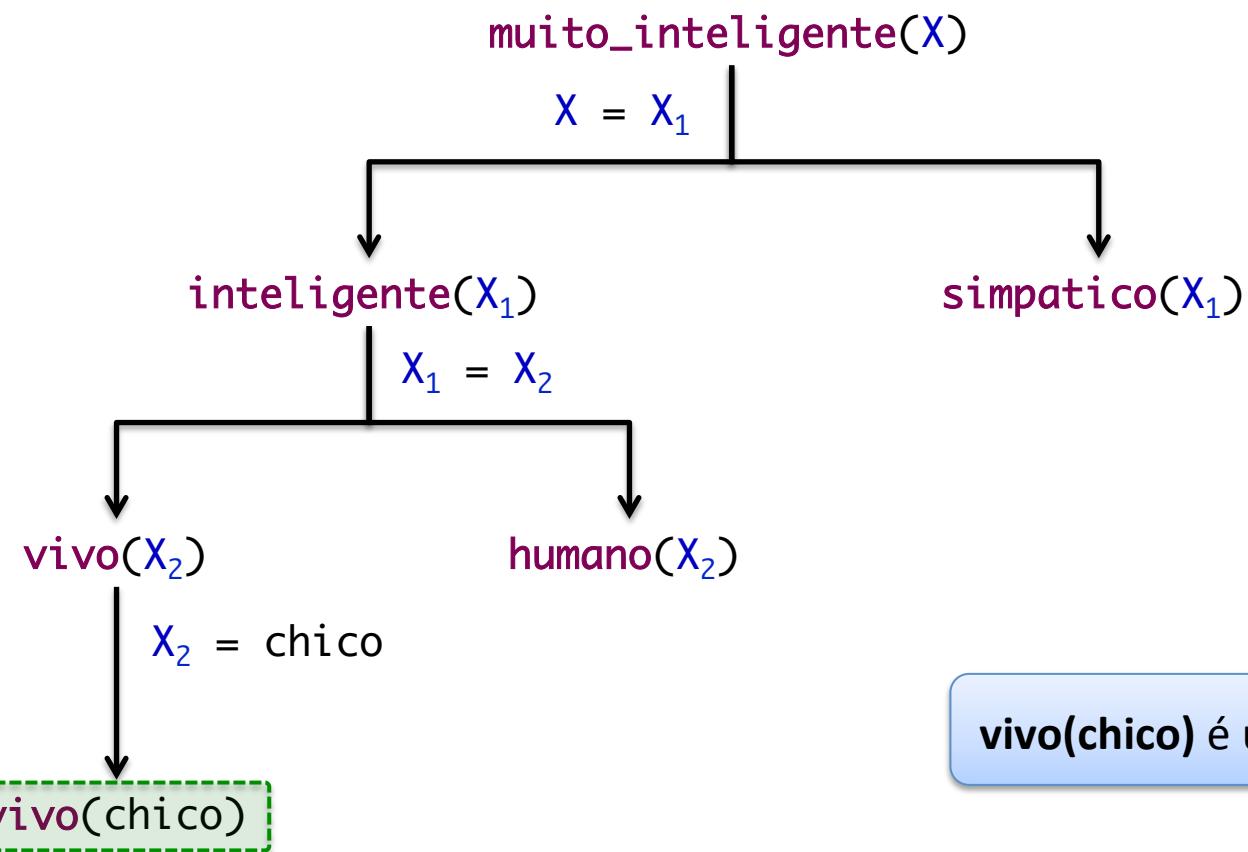
Outro Exemplo

- Execução do algoritmo



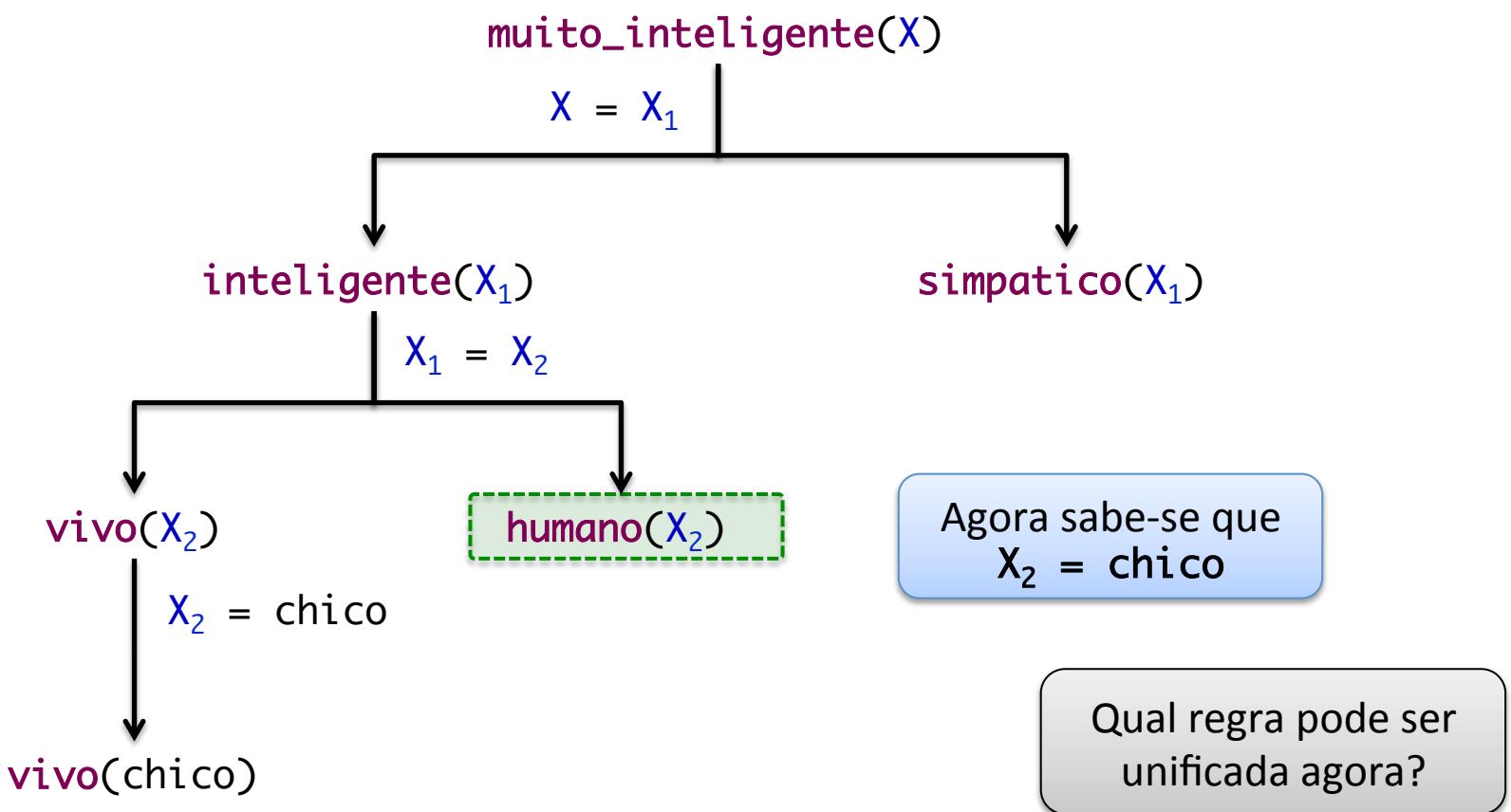
Outro Exemplo

- Execução do algoritmo



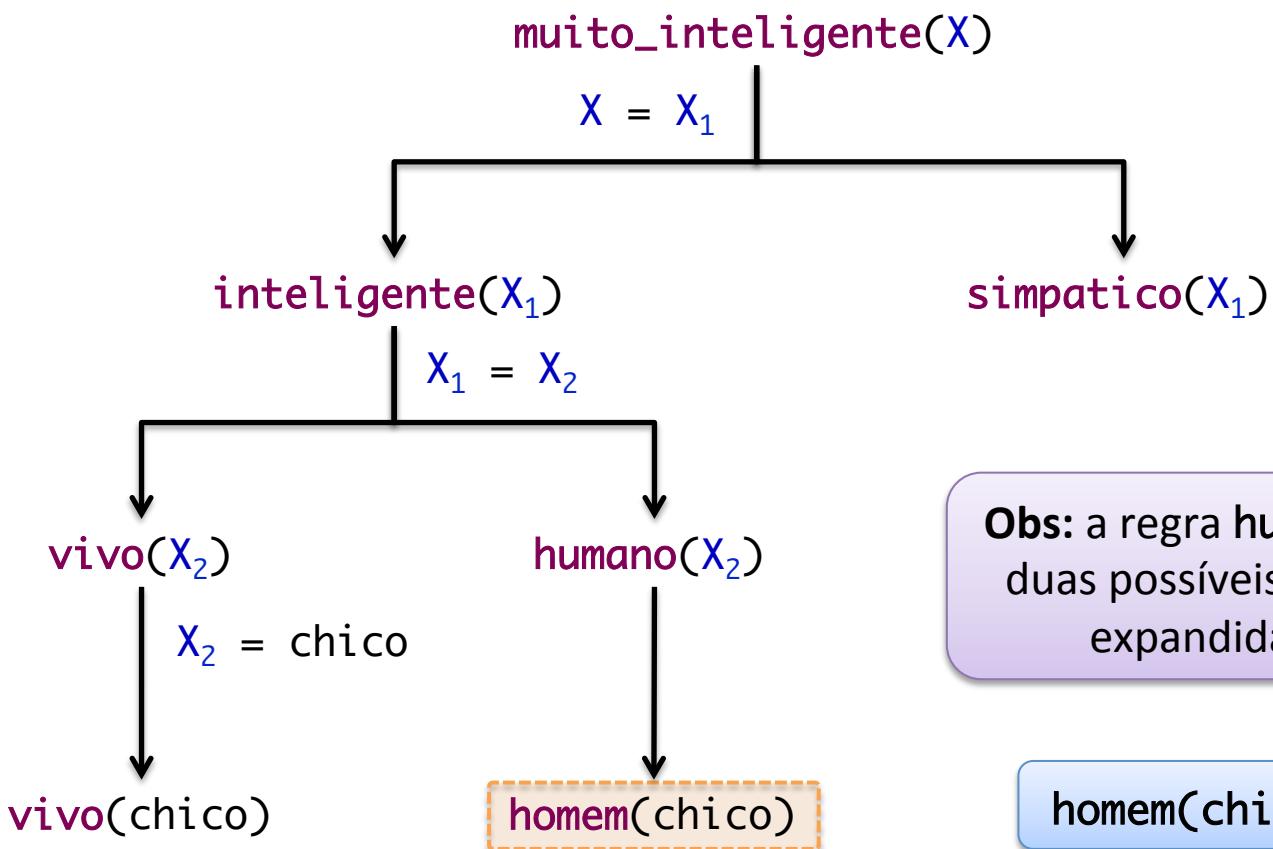
Outro Exemplo

- Execução do algoritmo



Outro Exemplo

- Execução do algoritmo

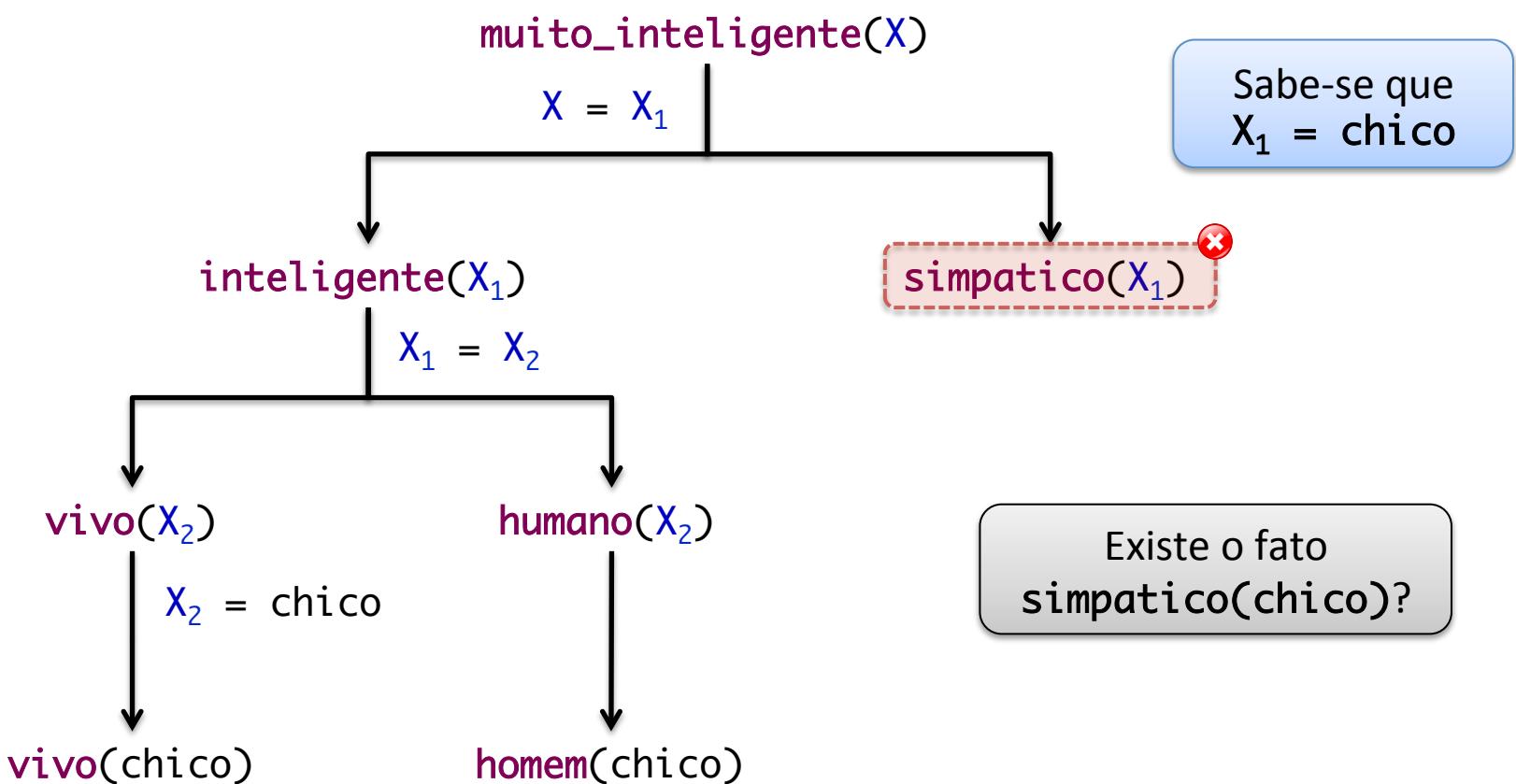


Obs: a regra `humano(X2)` possui duas possíveis unificações; foi expandida a primeira

`homem(chico)` é um fato

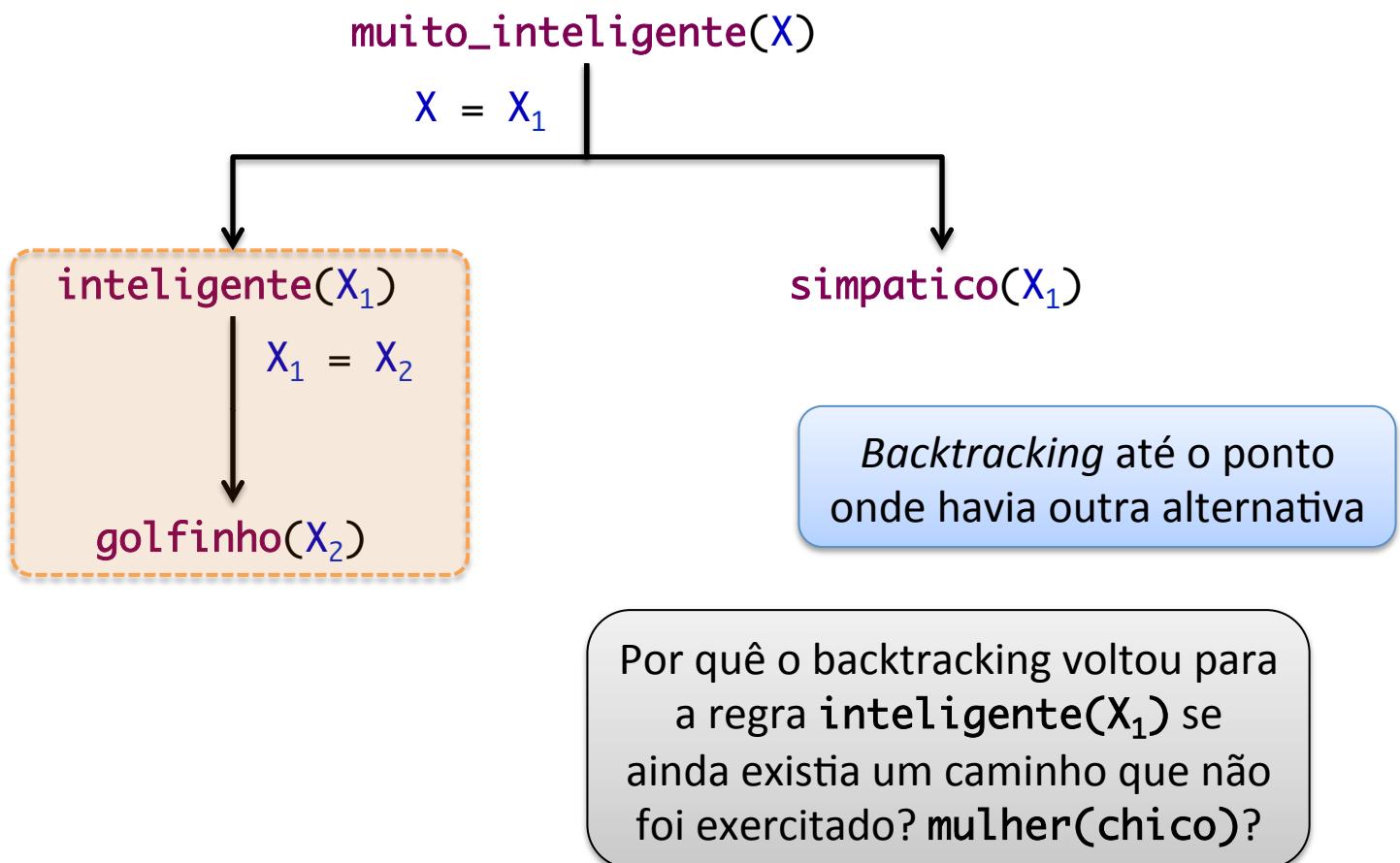
Outro Exemplo

- Execução do algoritmo



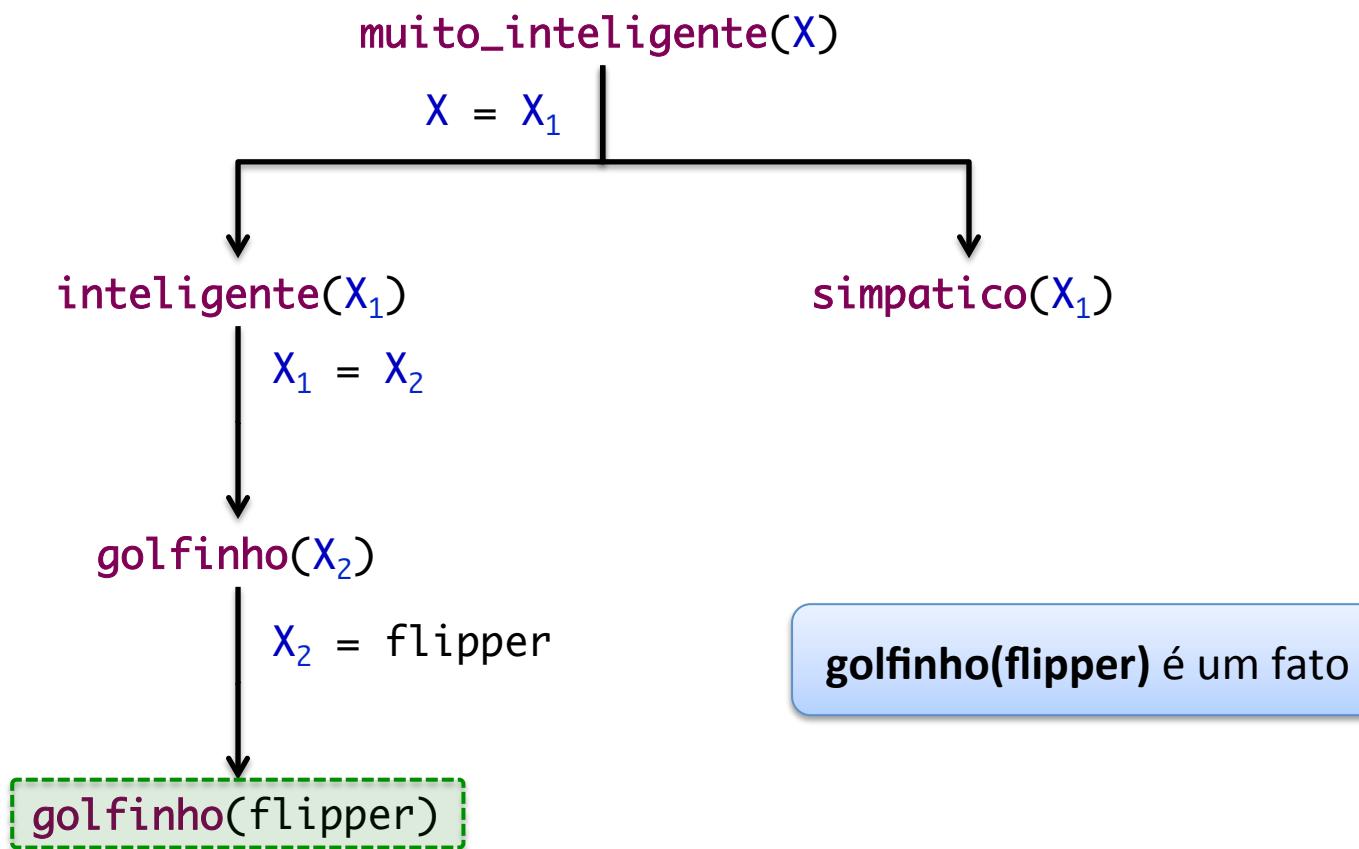
Outro Exemplo

- Execução do algoritmo



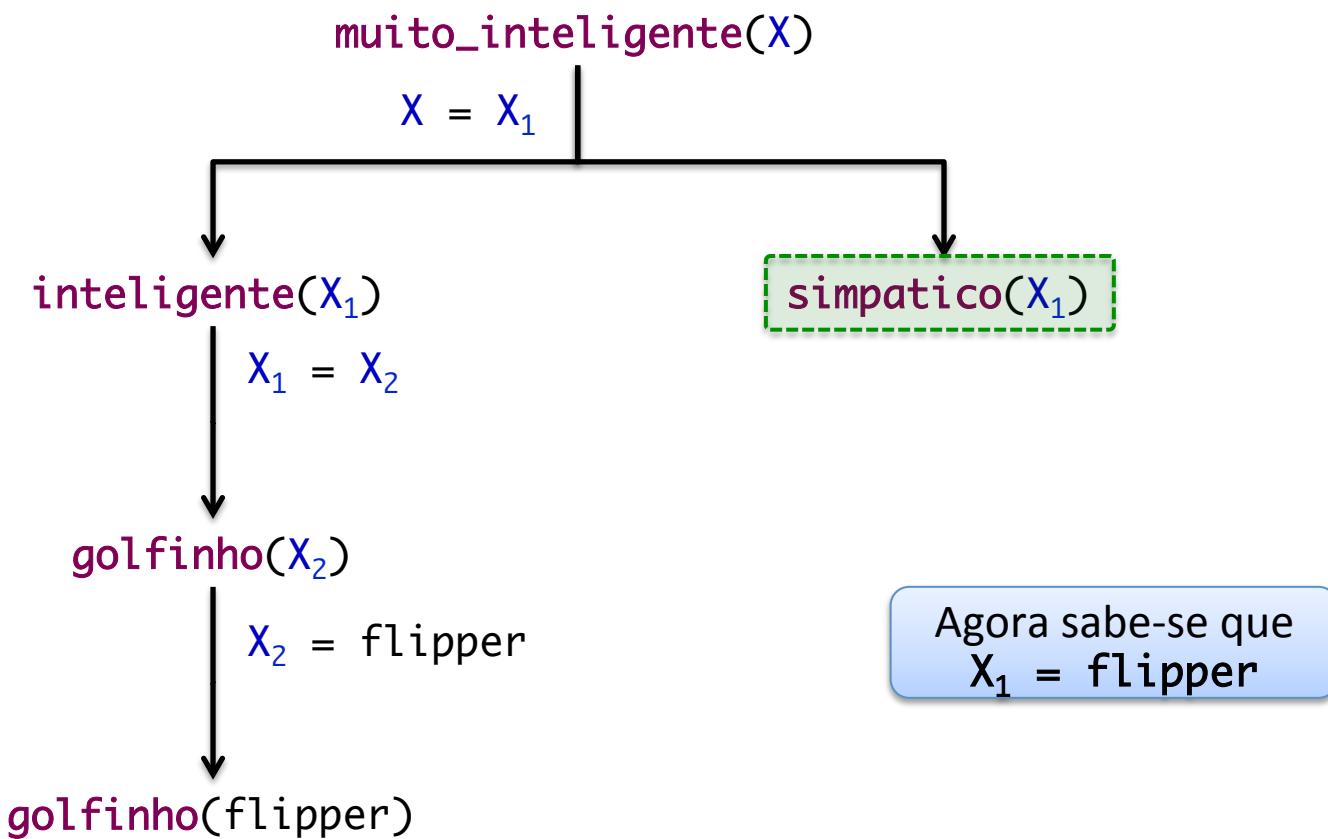
Outro Exemplo

- Execução do algoritmo



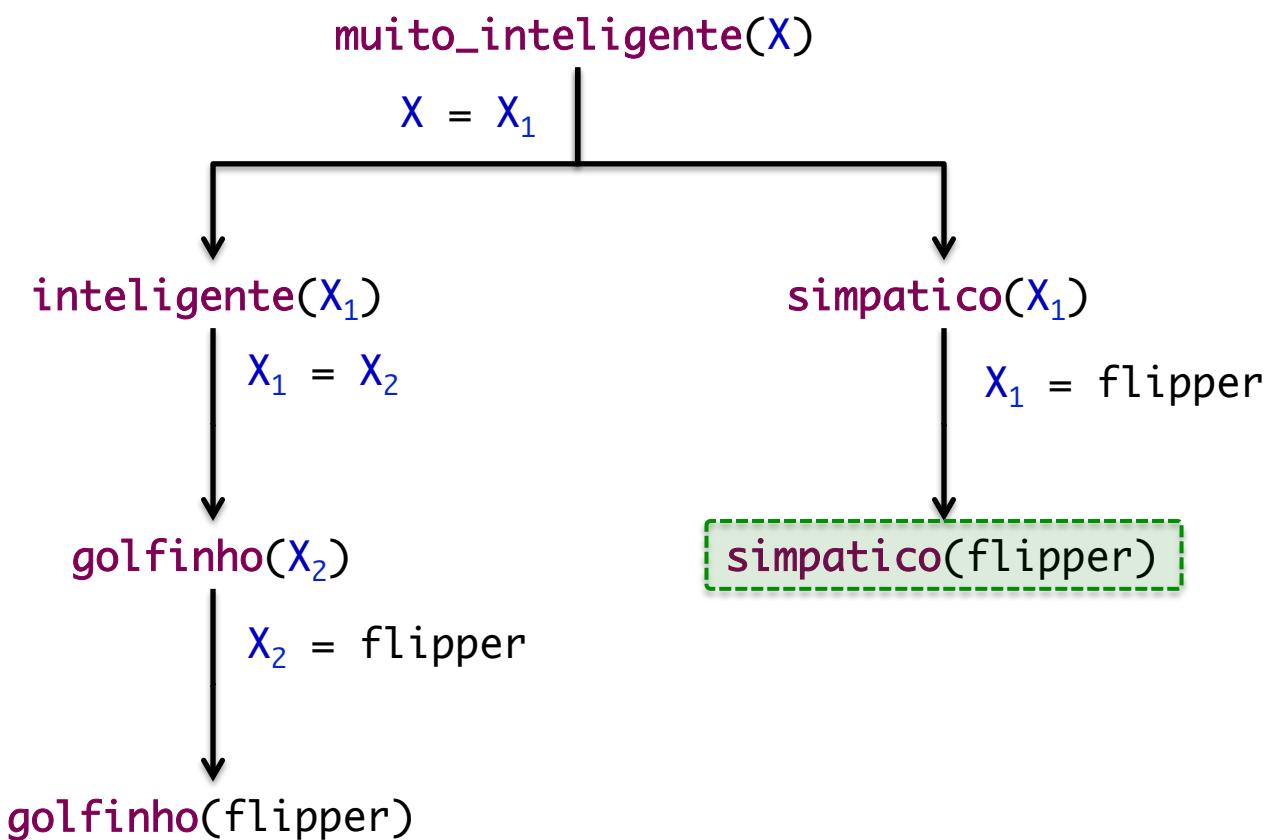
Outro Exemplo

- Execução do algoritmo



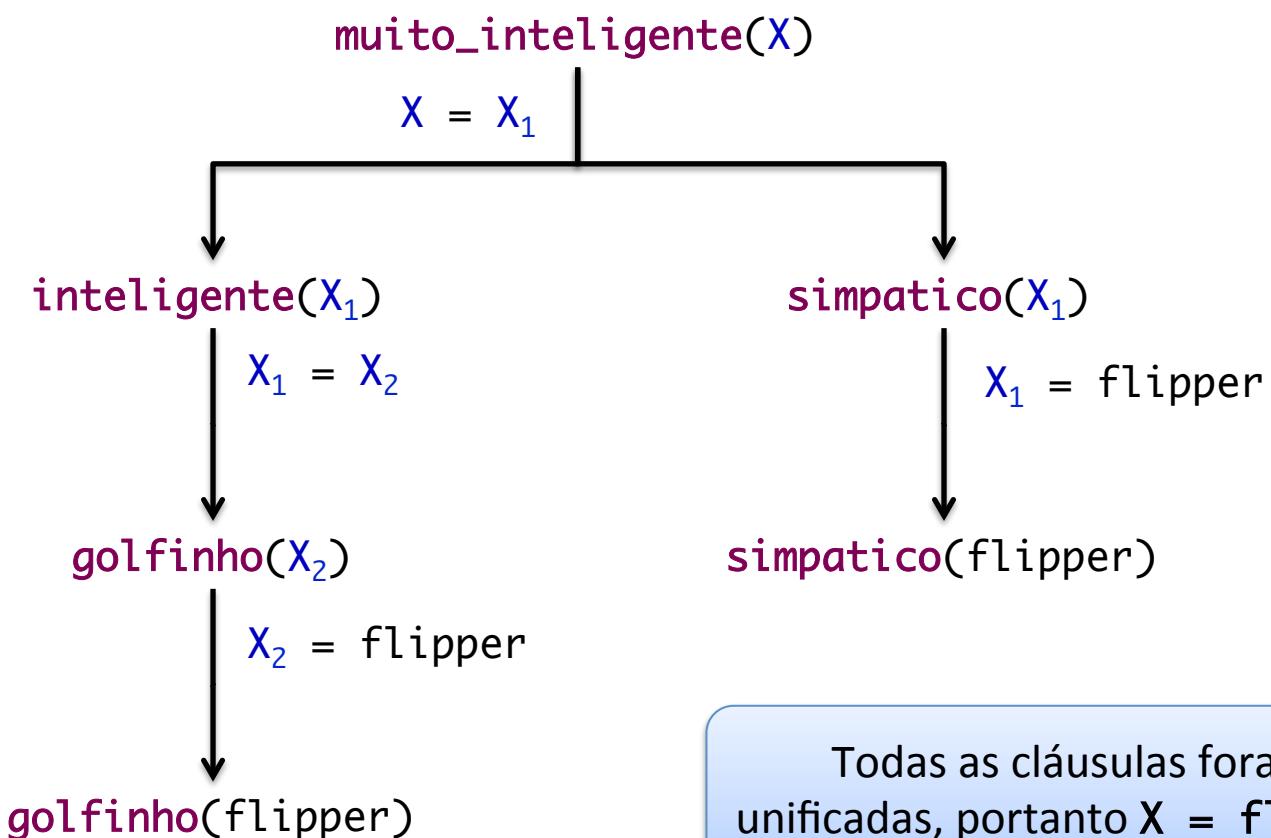
Outro Exemplo

- Execução do algoritmo



Outro Exemplo

- Execução do algoritmo



MAIS SOBRE PROLOG



Linguagens de Programação

Cláusulas Recursivas

- Pode-se definir cláusulas com recursão para executar laços
- Exemplo: consultar os descendentes
?- descendente(manoel, madalena).
true .
?- descendente(manoel, carlos).
true .

```
pai(carlos, madalena).  
pai(manoel, felipe).
```

```
mae(madalena, manoel).
```

```
pais(X, Y) :- pai(X, Y).  
pais(X, Y) :- mae(X, Y).
```

```
descendente(X, Y) :- pais(Y, X).  
descendente(X, Y) :- pais(Z, X), descendente(Z, Y).
```

Variáveis Anônimas

- Uma **variável anônima** é uma variável cujo valor instanciado não é relevante
- Exemplo: consultar se Felipe tem pai

```
?- pai(_, felipe).  
true.
```

Listas

- Listas são um tipo especial de estrutura
- Exemplos
 - **[1, 2, 3, 4]**: lista com elementos de 1 a 4
 - **[1 | [2, 3, 4]]**: lista com cabeça igual a 1 e o rabo igual a [2,3,4]
 - **[1, [2, 3], 4]**: lista cujos elementos podem ser outras listas
 - **[]**: lista vazia



Unificação de Listas

- Exemplos de unificações de listas

Predicado 1	Predicado 2	Unificação
$[X \mid Y]$	$[a, b, c]$	$X = a, Y = [b, c]$
$[X \mid [Y \mid Z]]$	$[a, b, c, d]$	$X = a, Y = b, Z = [c, d]$
$[X \mid Y]$	$[a]$	$X = a, Y = []$
$[X, Y \mid Z]$	$[a, b, c, d]$	$X = a, Y = b, Z = [c, d]$

Exemplos com Listas

- Verificar se existe um elemento na lista

```
elemento(X, [ X | _ ]).  
elemento(X, [ _ | L ]) :- elemento(X, L).
```

```
?- elemento(2, [2, 3, 5]).  
true .
```

```
?- elemento(V, [2, 3, 5]).  
V = 2 ;  
V = 3 ;  
V = 5 .
```

Exemplos com Listas

- Verificar se duas listas são iguais

```
igual([], []).  
igual([X | L1], [X | L2]) :- igual(L1, L2).
```

```
?- igual([1, 2], [1, 2]).  
true.
```

```
?- igual([], []).  
true.
```

```
?- igual([1, 2], X).  
X = [1, 2].
```

```
?- igual([1, 2], [3, 4]).  
false.
```

Exemplos com Listas

- Concatenar duas listas

```
concatenar([], L, L).  
concatenar([X|L1], L2, [X|L3]) :- concatenar(L1, L2, L3).
```

```
?- concatenar([2, 3],[5, 7], L).  
L = [2, 3, 5, 7].
```

```
?- concatenar([2, 3], L, [2, 3, 5, 7]).  
L = [5, 7].
```

```
?- concatenar(L1, L2, [5, 7]).  
L1 = [], L2 = [5, 7] ;  
L1 = [5], L2 = [7] ;  
L1 = [5, 7], L2 = [] .
```

Exemplos com Listas

- Remover um elemento da lista

```
remover(X, [X|T], T).  
remover(X, [Y|T], [Y|NT]) :- X \== Y, remover(X, T, NT).
```

```
?- remover(2, [1,2,3,4], [1,3,4]).  
true .
```

```
?- remover(2, [1,2,3,4], L).  
L = [1, 3, 4] .
```

```
?- remover(5, [1,2,3,4], L).  
false.
```

Como remover todas as ocorrências do elemento?

Expressões Aritméticas

- O operador `is` é usado para realizar operações aritméticas
 - Exemplo 1: Densidade

```
densidade(X,Y) :- populacao(X,P), area(X,A), Y is P/A.
```

- Exemplo 2: MDC (Algoritmo de Euclides)

```
mdc(X,X,X).  
mdc(X,Y,Z) :- X > Y, W is X-Y, mdc(W,Y,Z).  
mdc(X,Y,Z) :- Y > X, W is Y-X, mdc(X,W,Z).
```

Expressões Aritméticas

- O operador **is** é usado para realizar operações aritméticas
 - Exemplo 3: Fatorial

```
fatorial(0, 1).  
fatorial(X, Y) :- X1 is X-1, fatorial(X1, Y1), Y is X*Y1.
```

- Exemplo 4: Somar os elementos de uma lista

```
soma([], 0).  
soma([X | L], S) :- soma(L, S1), S is X+S1.
```

RESUMO



Linguagens de Programação

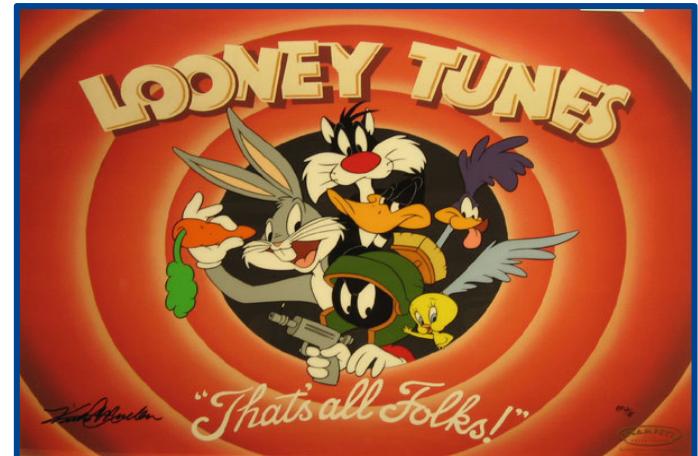
Paradigma Lógico vs Imperativo

- Custo de tempo e espaço de armazenamento alto
- *Backtracking*
 - Imperativa: erro de execução (se não puder prosseguir)
 - Lógica: computação desfeita e um caminho alternativo é testado
- Ausência de tipos e declarações
- Variáveis se referem a indivíduos, ao invés de posições da memória (não há atribuição destrutiva)

Paradigma Lógico vs Imperativo

- Programas em lógica não possuem estruturas de controle no sentido usual
- Manipulação de dados é feita inteiramente via unificação
 - Atribuição simples
 - Passagem de parâmetros
 - Alocação de estruturas e escrita/leitura de campos de estruturas
- Um programa em lógica pode ficar mais
 - Conciso (de 5 a 10 vezes menor)
 - Limpo
 - Legível

ISSO É TUDO PESSOAL!



Linguagens de Programação