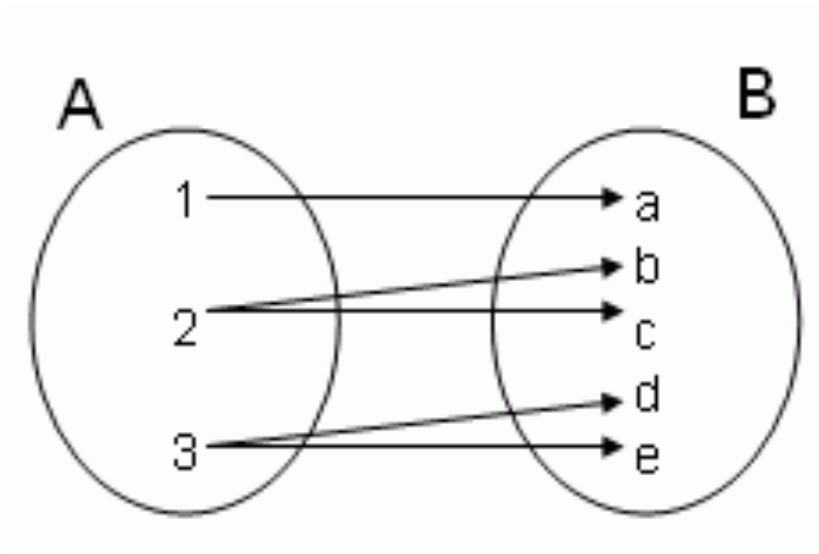


PARADIGMA LÓGICO

Everton L. G. Alves
everton@computacao.ufcg.edu.br

Introdução

- Mesmo paradigmas diferentes como funcional e imperativo têm algo em comum:
 - Um programa lê **entradas** e gera **saídas**
 - **Mapeamento**



Dada a entrada x ,
determine o valor de $m(x)$

Introdução

- Um programa segundo o paradigma lógico implementa uma **relação**

Considerando dois conjuntos S e T , dizemos que existe uma relação r entre S e T se, para cada x em S e y em T , a avaliação $r(x,y)$ é **verdadeiro** ou **falso**

- Relações são mais gerais que mapeamentos
- Programação lógica é mais alto nível que a programação imperativa ou funcional

Relação - Exemplos

- A relação de maior (“>”) entre números
 - Para qualquer par de números x e y , $x > y$ é sempre **verdadeiro** ou **falso**
 - Ex: ? $7 > 10 \rightarrow \text{false}$
? $1234 > 23 \rightarrow \text{true}$
- A relação “corta” entre rios e territórios
 - ? “O rio Nilo corta o Egito” $\rightarrow \text{true}$
 - ? “O rio Amazonas corta a Paraíba” $\rightarrow \text{false}$

Programação Lógica

- A programação lógica é baseada em **relações**
- Tendo implementado a relação r , podemos fazer consultas como:

- Dado x e y , determine se $r(x,y)$ é true
- Dado x , encontre **todos os y** onde $r(x,y)$ é true
- Dado y , encontre **todos os x** onde $r(x,y)$ é true
- Encontre **todos os x e y** onde $r(x,y)$ é true

Lógica e Programação

- Nenhuma linguagem de programação lógica consegue explorar totalmente o potencial da lógica matemática
 - O formalismo matemático não é 100% implementável

Lógica e Programação

- As linguagens de programação em lógica estão restritas à cláusulas de Horn

Cláusulas de *Horn*:

A_0 if A_1 and ... and A_n .

onde:

- A_i é uma assertiva da forma $R_i(...)$, onde R_i é o nome de uma relação.
- Se $A_1, ..., A_n$ são verdadeiras, podemos inferir que A_0 também é verdadeira.
- Se A_i é falso, não podemos concluir que A_0 também é.
- Se $n = 0$, uma cláusula de *Horn* é dita ser um fato: A_0 .
- Um programa lógico é uma coleção de cláusulas de *Horn*

Exemplo em Prolog

Cláusulas de
Horn

```
star(sun).  
star(sirius).  
star(betelgeuse).  
via-lactea(X) :- star(X).
```

Relações com
uma cláusulas
→ Fatos

Relação A_0 if A_1

```
Avaliando:  
?- star(sun).  
true  
?- star(jupiter).  
false  
?- via-lactea(sun).  
true  
?- via-lactea(earth).  
false
```


Paradigma Lógico

- Também parte da sub categoria “paradigma declarativo”
- Os programas lógicos se preocupam com os fatos e regras, não com os procedimentos
- O conhecimento acerca do problema é descrito usando lógica simbólica

Programação em Lógica

- Formalismo lógico-computacional fundamentado em três princípios:
 - Uso de linguagem formal para **representação de conhecimento** (linguagem formal)
 - Uso de **regras de inferência** para manipulação de conhecimento
 - Uso de uma **estratégia de busca** para controle de inferências
- O programador deve preocupar-se em **descrever** o problema a ser solucionado (especificação), deixando a critério da **máquina** a busca pela solução

Linguagem Formal

- Uma linguagem natural pode ser ambígua
 - “Ana viu um homem numa montanha usando um binóculo”
 - Quem usava o binóculo?
 - “Ana, usando um binóculo, viu um homem numa montanha”
 - “Ana, estando numa montanha, viu um homem que usava um binóculo”
- Uma linguagem formal é **precisa**
 - Suas sentenças são objetos (fórmulas) com significados únicos, e têm sintaxe e semântica bem definidas
 - Mas também pode ser menos expressiva

Regras de Inferência

- Uma regra de inferência é um padrão de manipulação sintática que:
 - Permite criar novas fórmulas a partir de outras existentes
 - Em geral, simulam formas de raciocínio válidas

- Exemplo (modus ponens)

$$\frac{P \rightarrow Q, P}{\therefore Q}$$

Se estiver chovendo, eu encontrarei você no cinema.
Está chovendo.
Então, encontrarei você no cinema.

- P implica Q, P é verdade, portanto, Q deve ser verdade

Regras de Inferência



Estratégia de Busca

- Um agente pode ter uma enorme quantidade de **conhecimento armazenado**
- Normalmente, ele precisa usar apenas **parte** de seu conhecimento para **resolver** um problema
- A estratégia de busca serve para **decidir que parte do conhecimento armazenado deve ser explorada em busca da solução**



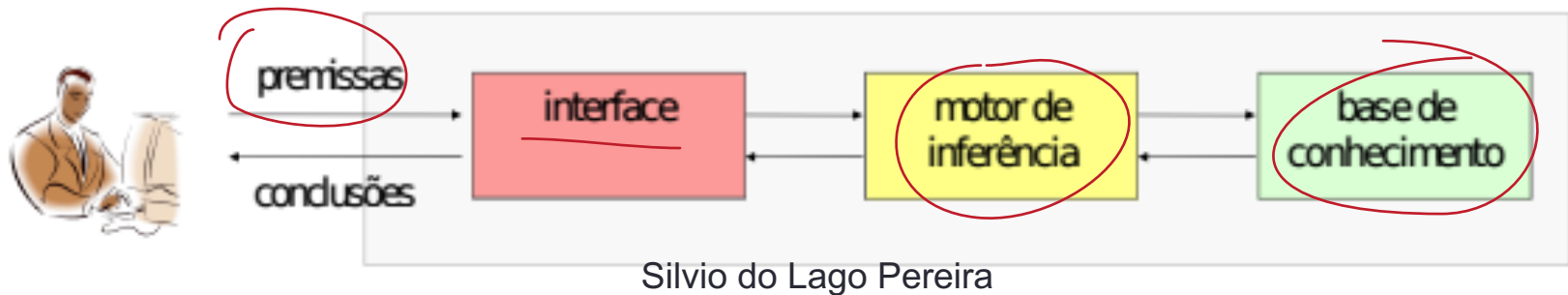
Prolog



- LP lógica geralmente associada com o contexto de IA e/ou linguística
- Possui suas raízes na lógica de primeira-ordem
- A linguagem foi concebida por um grupo coordenado por **Alain Colmerauer** em 1970
- Foi uma das primeiras LPs lógicas e se mantém como a **mais popular**
- Bastante usada para prova de teoremas, sistemas especialistas, processamento de linguagens naturais, ...

Prolog

- Sistema de programação em lógica mais popular



- Composto:
 - Interface: permite que o usuário entre com premissas e faça consultas para extrair conclusões
 - Motor de inferência: atualiza a base de conhecimento com premissas fornecidas pelo usuário e faz inferências para extrair informações implícitas
 - Base de conhecimento: armazena as premissas fornecidas pelo usuário

Prolog - Vantagens

- Permite representar o conhecimento que um agente tem sobre seu mundo de uma forma
 - Simples
 - Direta
 - Alto nível

Programação imperativa → como o computador deve proceder para resolver um problema

Programação Lógica → declarar o conhecimento que temos acerca do problema. O sistema deve ser capaz de encontrar a solução

Tipos

- Prolog aplica um único tipo, **Termo**
- Um termo pode ser um número, átomo, variáveis ou termos compostos
 - Átomos começam com letra minúscula ou entre aspas simples:
 - elefante, xYZ, a_123, 'Meu nome'
 - Variáveis começam com letra maiúscula ou underscore:
 - X, Elefante, _G177, MinhaVariável, _
 - Um termo composto tem uma relação e seus argumentos:
 - is_bigger(horse, X)
 - f(g(Alpha, _), 7)
 - 'My Functor'(dog)

Cláusulas de Horn em Prolog

- Um subconjunto das cláusulas da lógica dos predicados

A_1 if A_2 and ... and A_n .

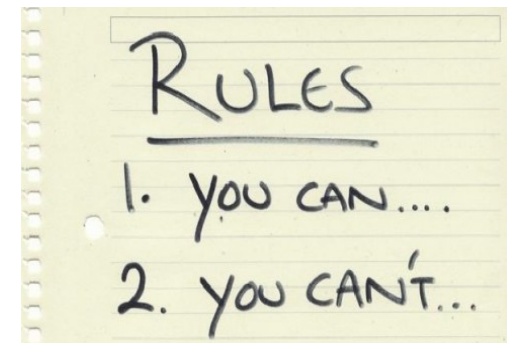
- Em Prolog:

$B \text{ :- } A_1, A_2, \dots, A_n$.

- Onde 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
 - Se algum A_i for falso, não se pode deduzir que B é falso, mas apenas que não se pode inferir verdade

Cláusulas de Horn

- Toda cláusula termina com um ponto (.)
- Nomenclatura:
 - Se $n = 0$ a cláusula é chamada de fato
 - Se $n \geq 1$ a cláusula é chamada de regra



Exemplo



Fatos

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

- Consultas:
 - 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.

Exemplo

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

Fatos

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

Regras









- ?- 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.

Prolog - Unificação

- Ao responder a uma consulta, o interpretador Prolog pode precisar **atribuir valores às variáveis**
- O conceito de **unificação** é uma das principais idéias de Prolog
- Representa o mecanismo de ligar os conteúdos das variáveis e pode ser visto como um tipo de atribuição
- Dois termos podem ser unificados se:
 - Os dois termos são constantes
 - 1 e 1, casa e 'casa'
 - Um dos termos é uma variável
 - X e 1, Y e masc(jose), X e Y
 - Ambos são termos complexos com o mesmo nome e mesmo número de argumentos
 - pai(james, jonh) e pai (X, jonh) → james é unificado com X

Prolog - Unificação

- Unificação é o mecanismo usado para determinar se existe uma maneira de instanciar as variáveis de dois predicados de modo a torná-los iguais

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

Prolog - Unificação

- Unificação é o mecanismo usado 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
$p(G, jose)$	$p(X, Y)$	$G = X, Y = jose$

Prolog – Algoritmo de Execução

1. Ao receber uma consulta, o Prolog tenta unificá-la com cada um dos fatos e com a cabeça das regras que estão na base de conhecimento

Regra → cabeça :- corpo.

2. Se houver unificação com um fato, retorna as instâncias das variáveis
3. Se existe unificação com a cabeça de uma regra, consulta, recursivamente, o corpo da regra (da esquerda para a direita)
 - Podem haver retrocessos (backtracking) no caso de uma unificação não ser bem sucedida, tentando explorar outras unificações alternativas
4. Se todas as alternativas forem exploradas sem sucesso, então a resolução falha

Exemplo

- Conhecimento: Paulo e Roberta têm dois filhos, João e Francisco
- Como representar tal conhecimento usando Prolog de modo que eu consiga descobrir, por exemplo, se um sujeito é irmão de outro?
 - Quais os fatos?
 - Quais as regras?

Exemplo



```
homem(francisco).  
  
mae(joao, roberta).  
mae(francisco, roberta).  
  
pai(joao, paulo).  
pai(francisco, paulo).  
  
pais(X, M, P) :- mae(X, M),  
                  pai(X, P).  
  
irmao(X, Y) :- homem(Y),  
               pais(X, M, P),  
               pais(Y, M, P).
```

- Consulta:

- ?- irmao(joao, francisco).
- Qual a resposta?

Exemplo

```
irmão(joao, francisco)
```

Com qual regra essa consulta
pode ser unificada?

Exemplo

`irmao(joao, francisco)`



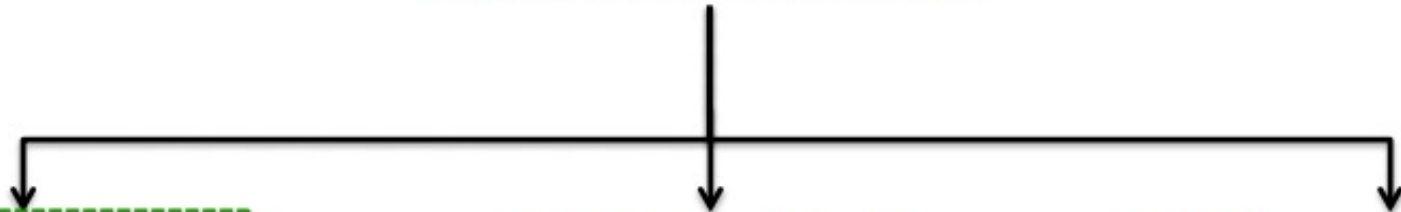
$X = \text{joao}, Y = \text{francisco}$

`irmao(X, Y)`

Expandir a regra `irmao(X, Y)`
com suas cláusulas a direita

Exemplo

irmão(joao, francisco)



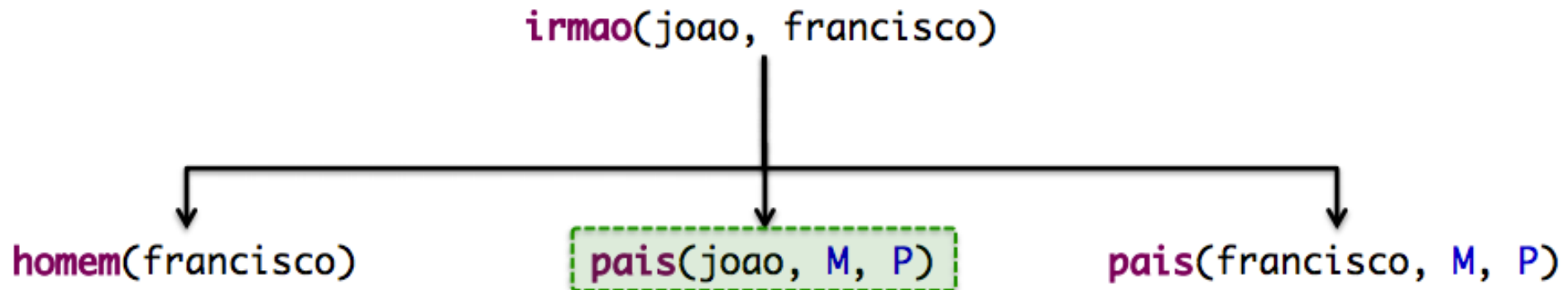
homem(francisco)

pais(joao, M, P)

pais(francisco, M, P)

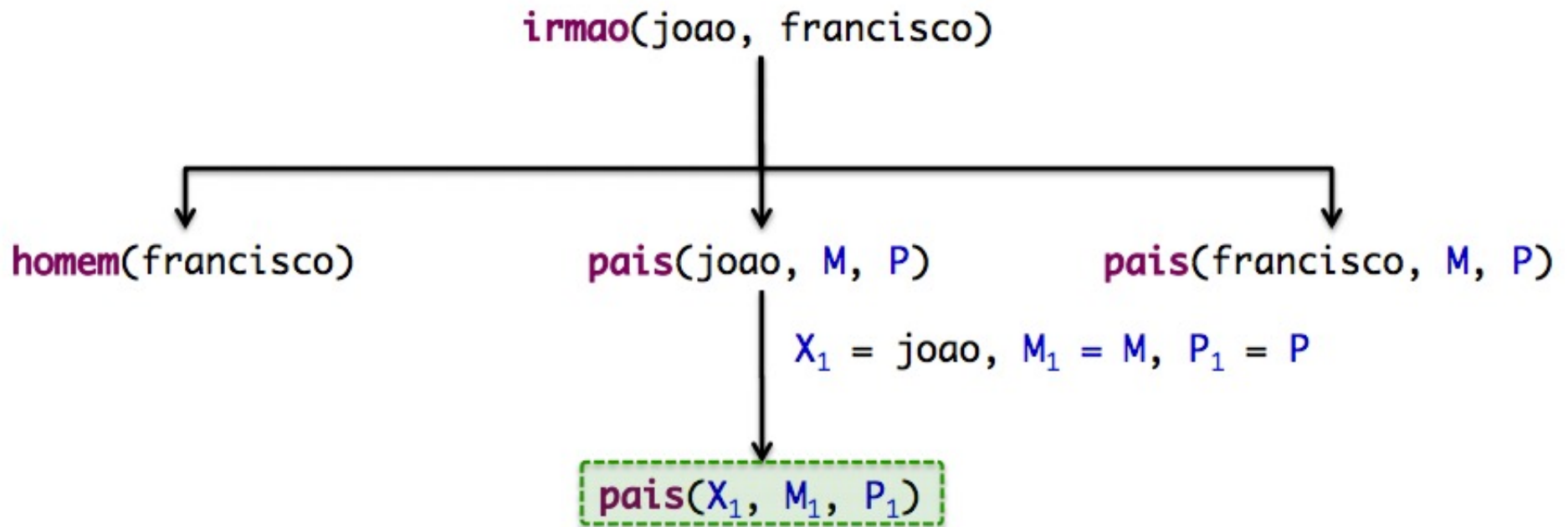
A cláusula **homem**(francisco) é um fato

Exemplo



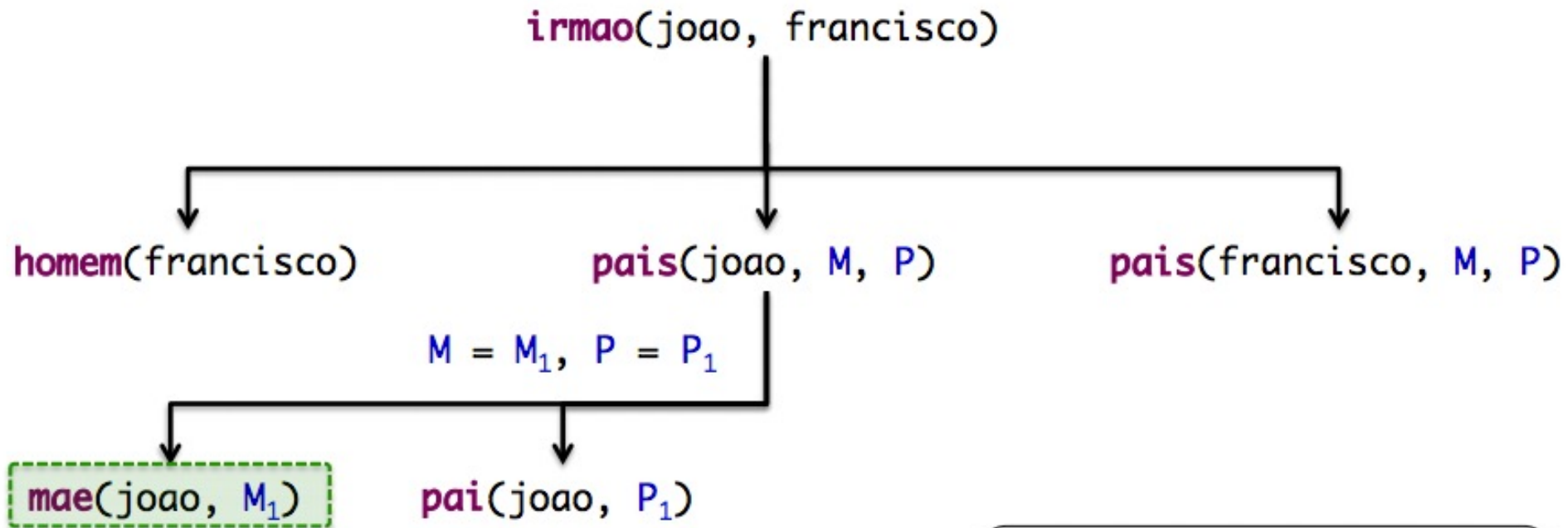
Qual regra pode ser unificada nesse momento?

Exemplo



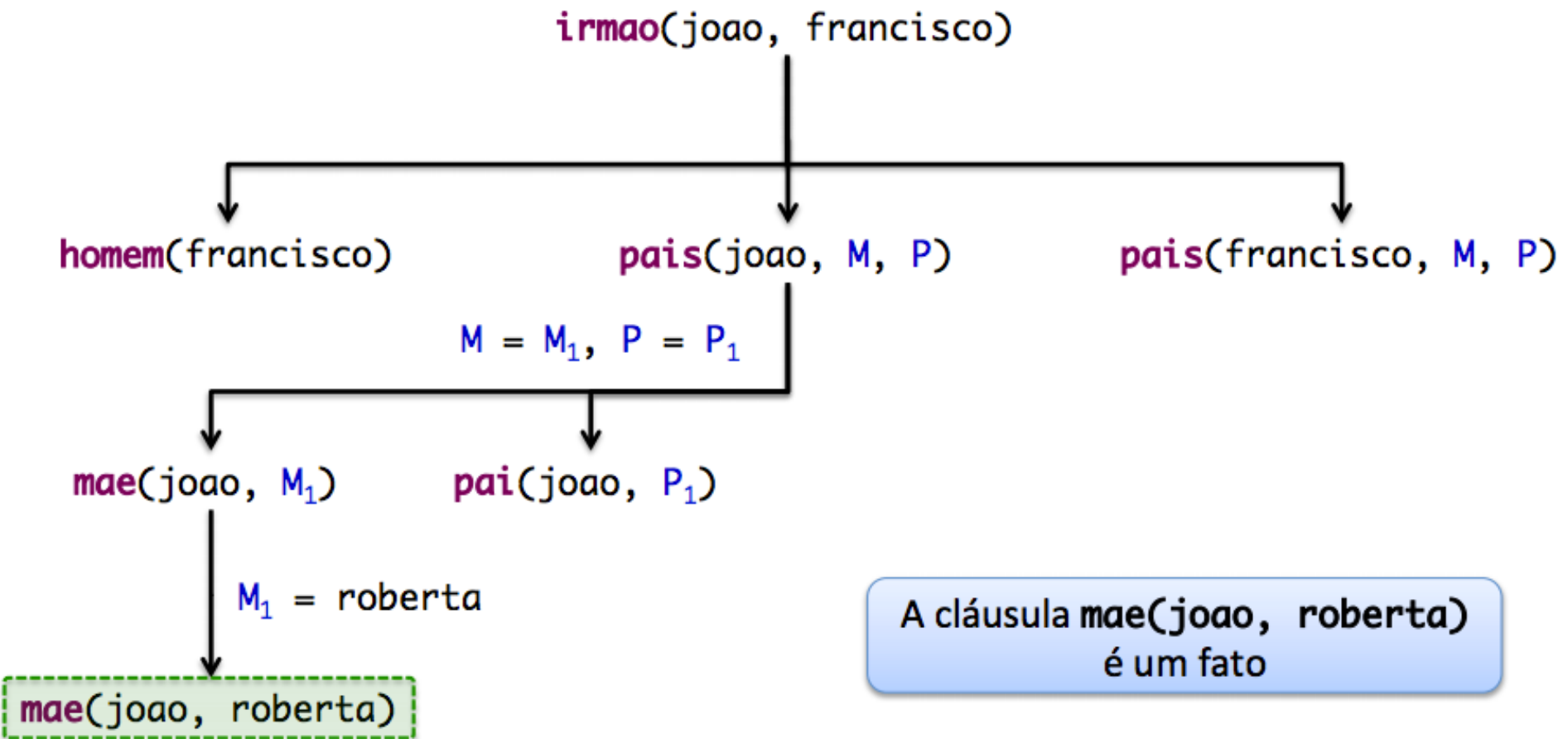
Expandir a regra `pais(X1, M1, P1)`
com suas cláusulas à direita

Exemplo

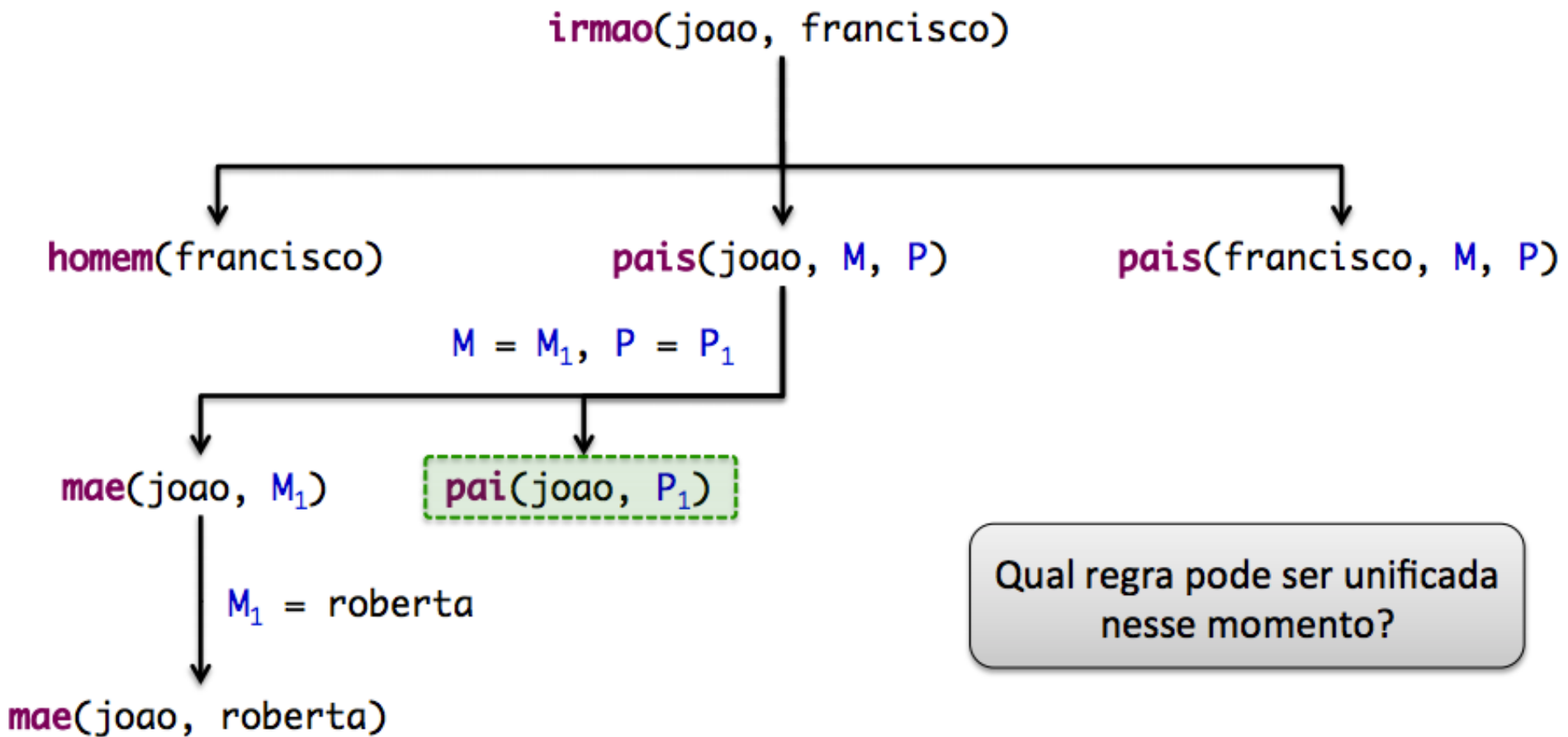


Qual regra pode ser unificada nesse momento?

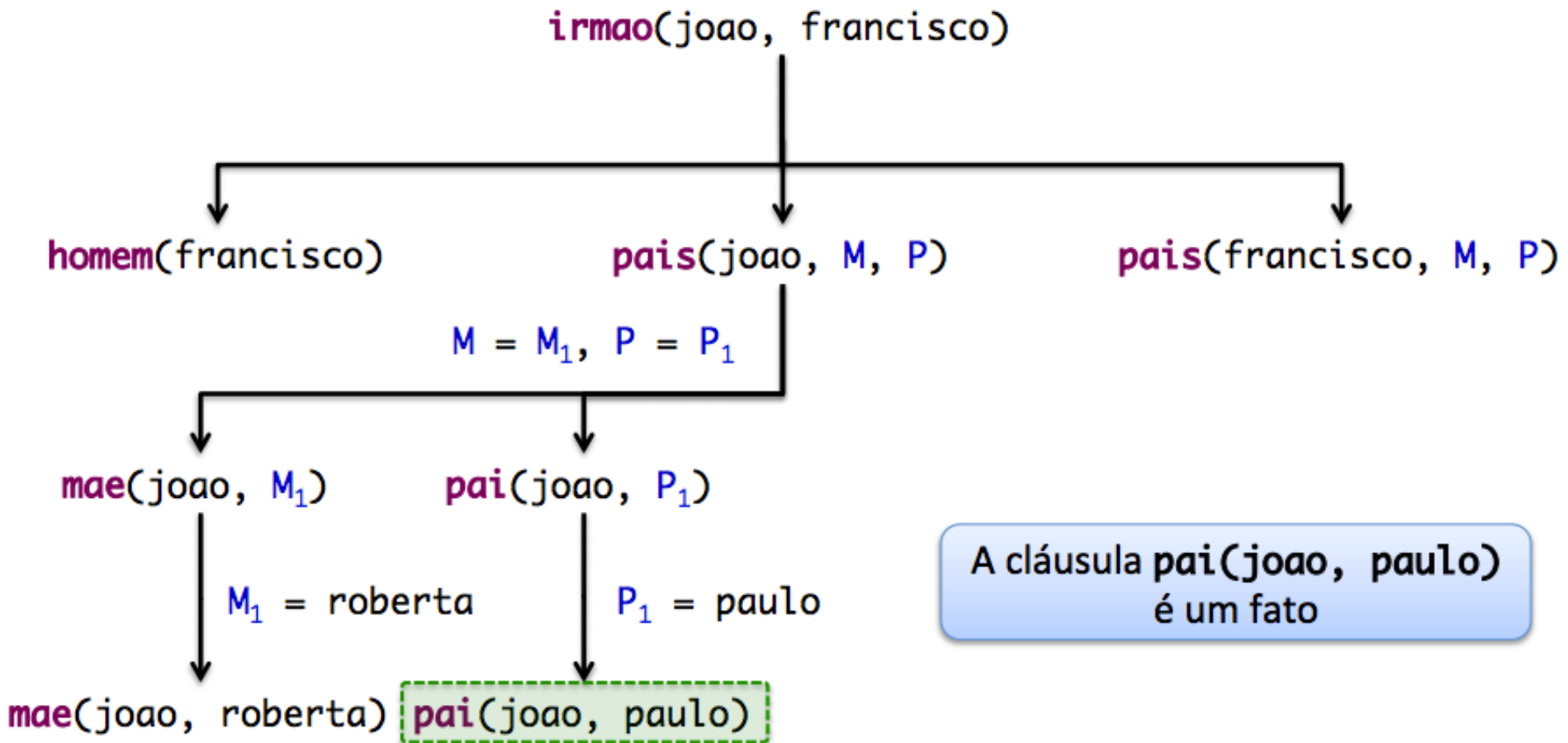
Exemplo



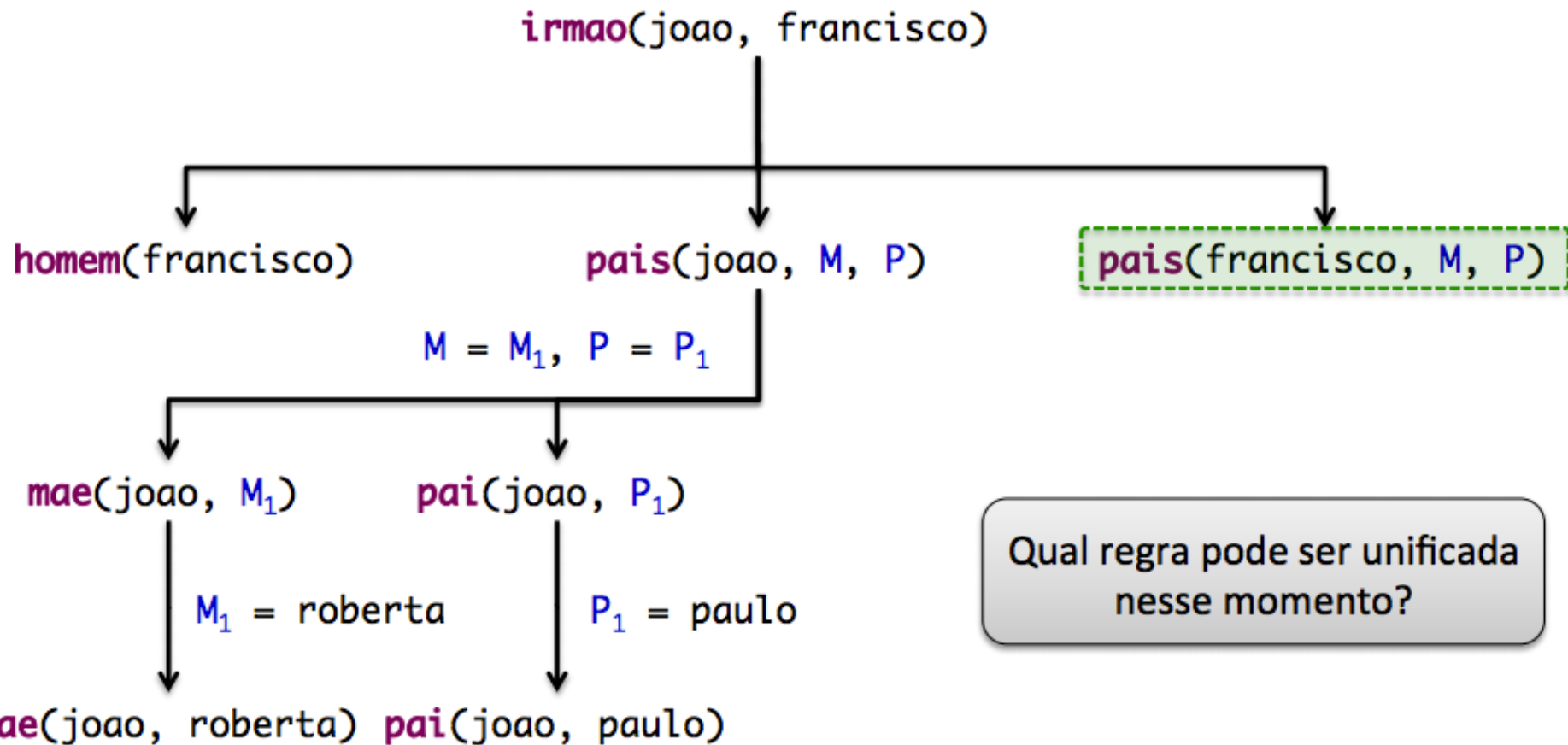
Exemplo



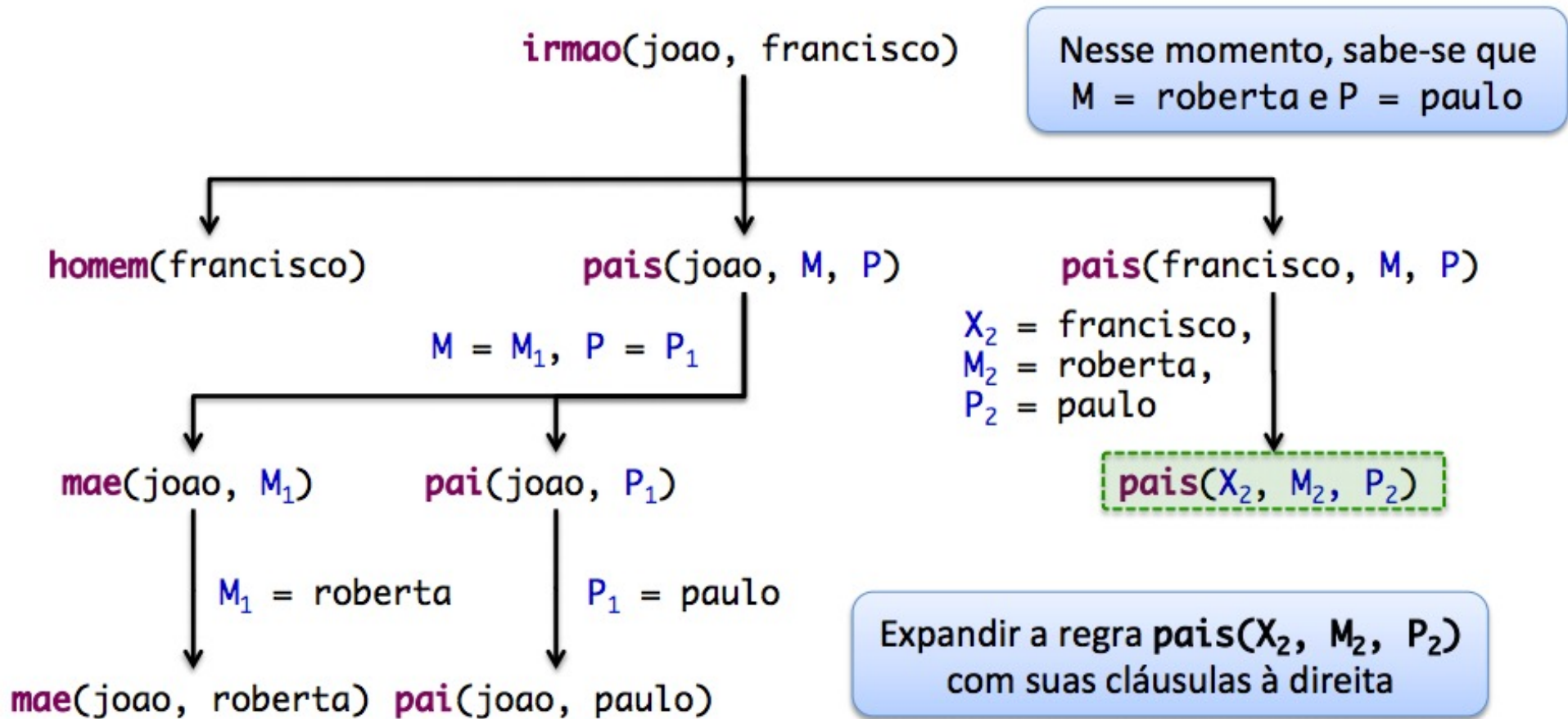
Exemplo



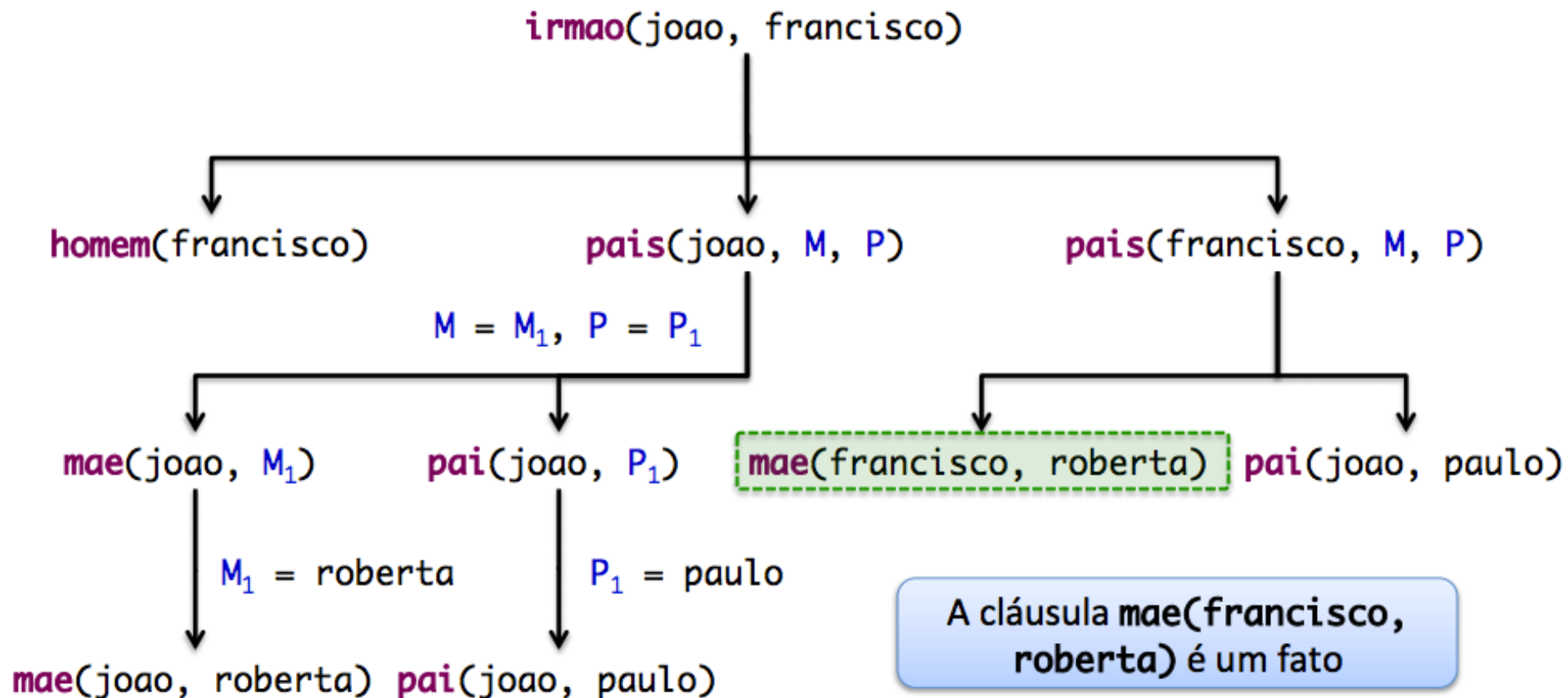
Exemplo



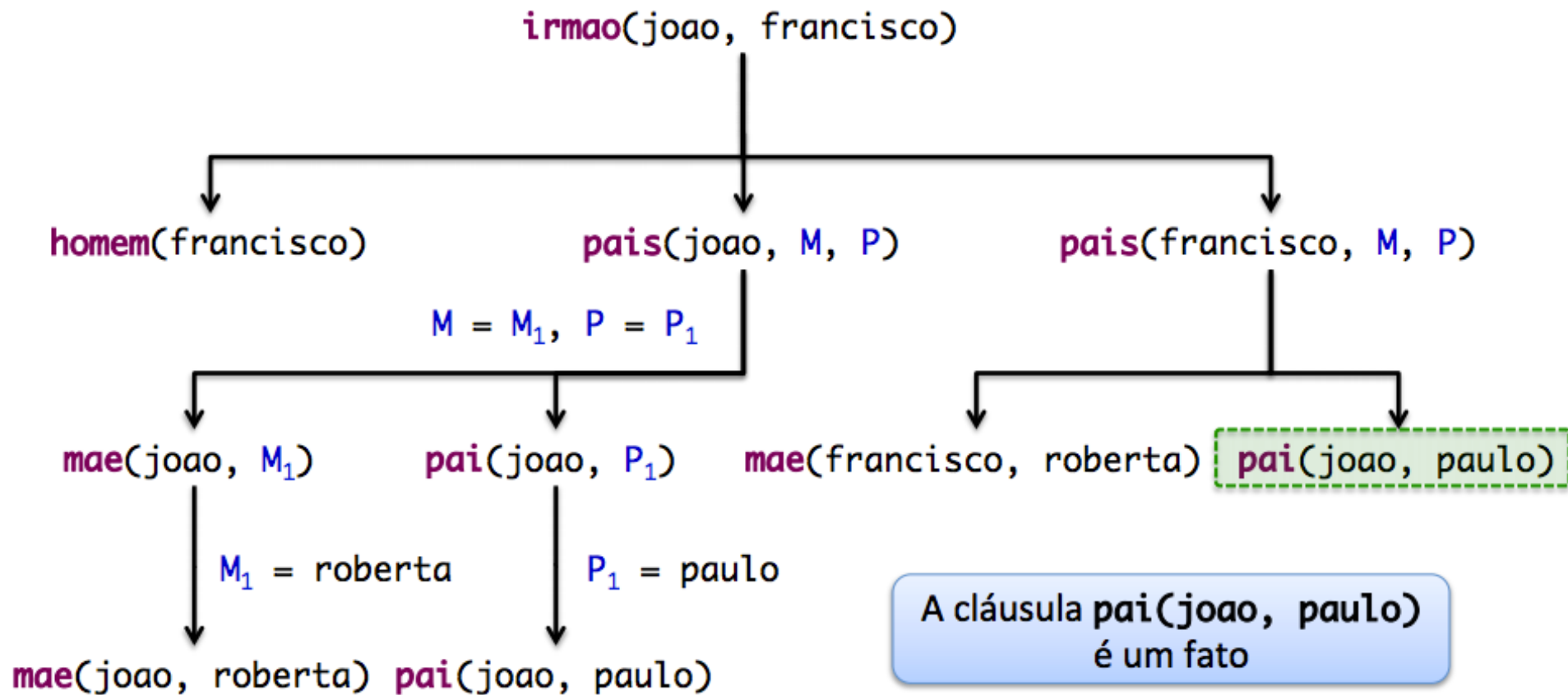
Exemplo



Exemplo

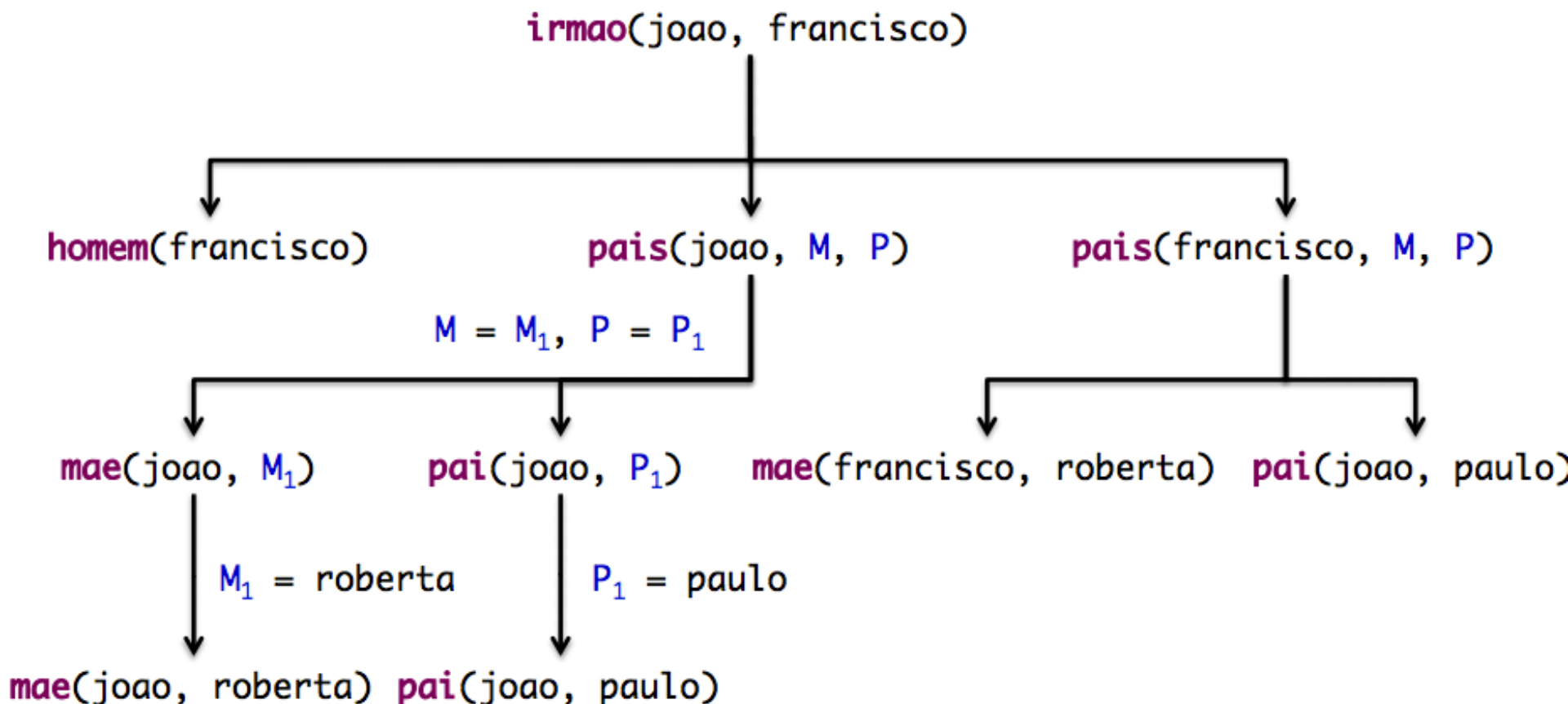


Exemplo



Exemplo

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



Exemplo 2

```
 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).
```

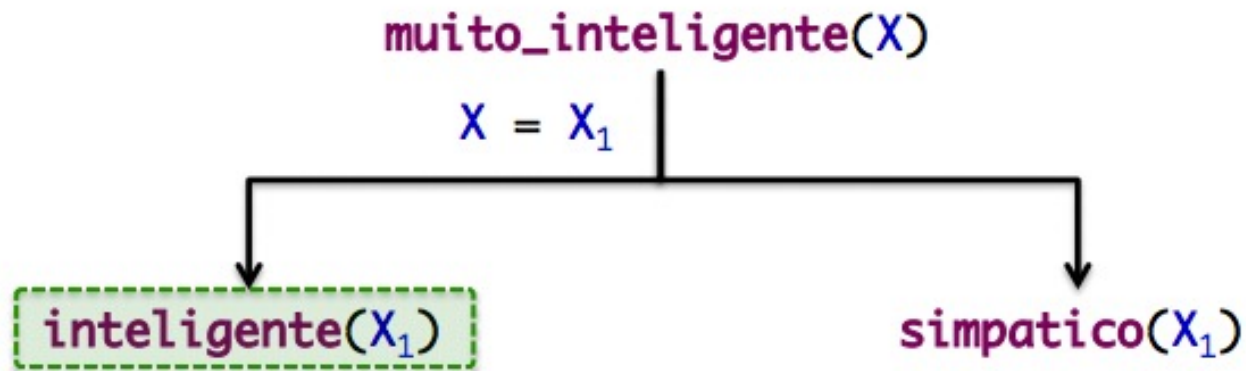
- Consulta:
 - ?- muito_inteligente(X)
- Qual a resposta?

Exemplo 2

`muito_inteligente(X)`

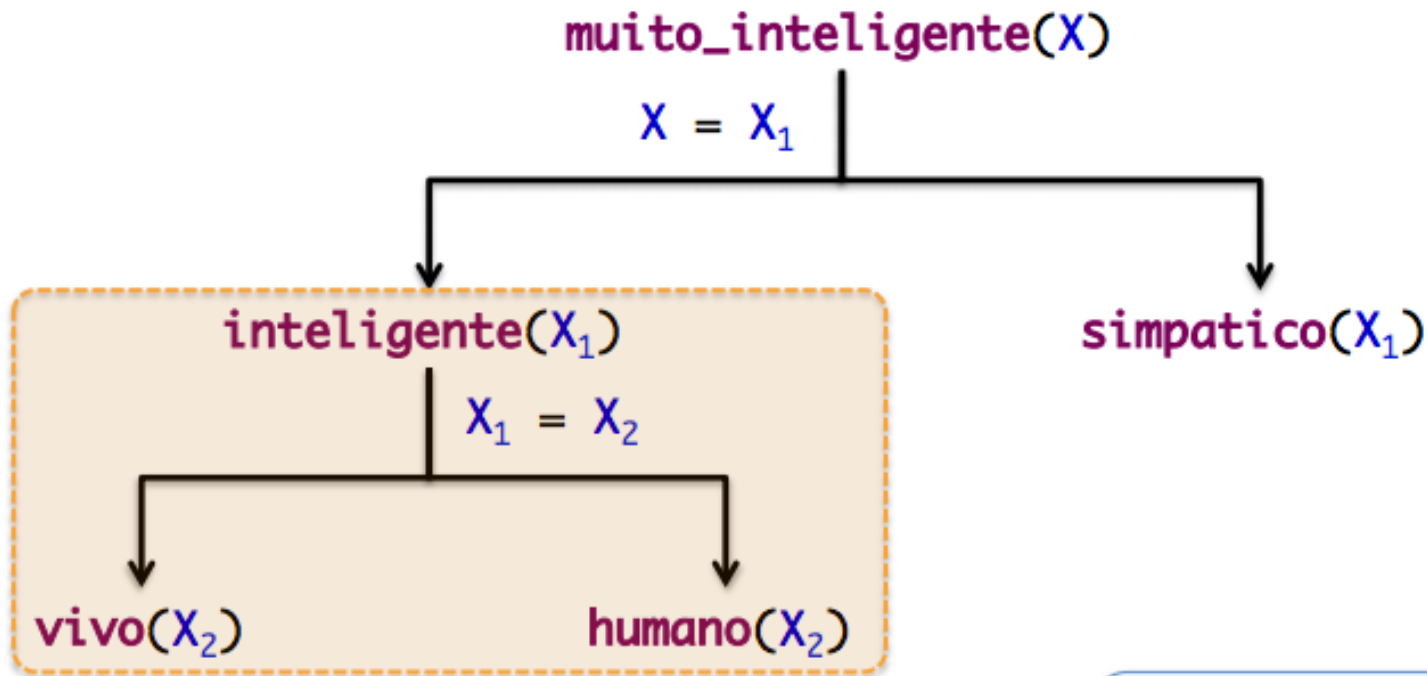
Com qual regra essa consulta
pode ser unificada?

Exemplo 2



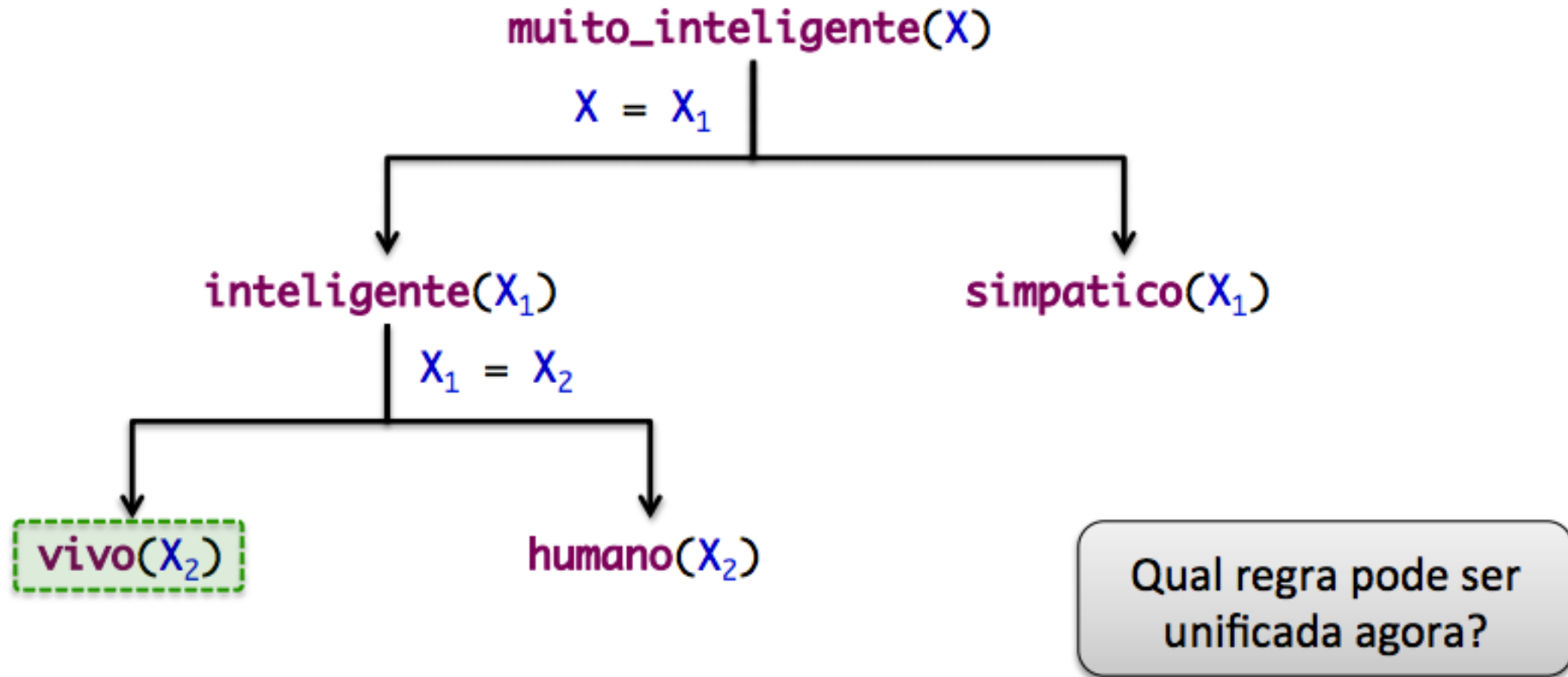
Quais regras podem ser unificadas agora?

Exemplo 2

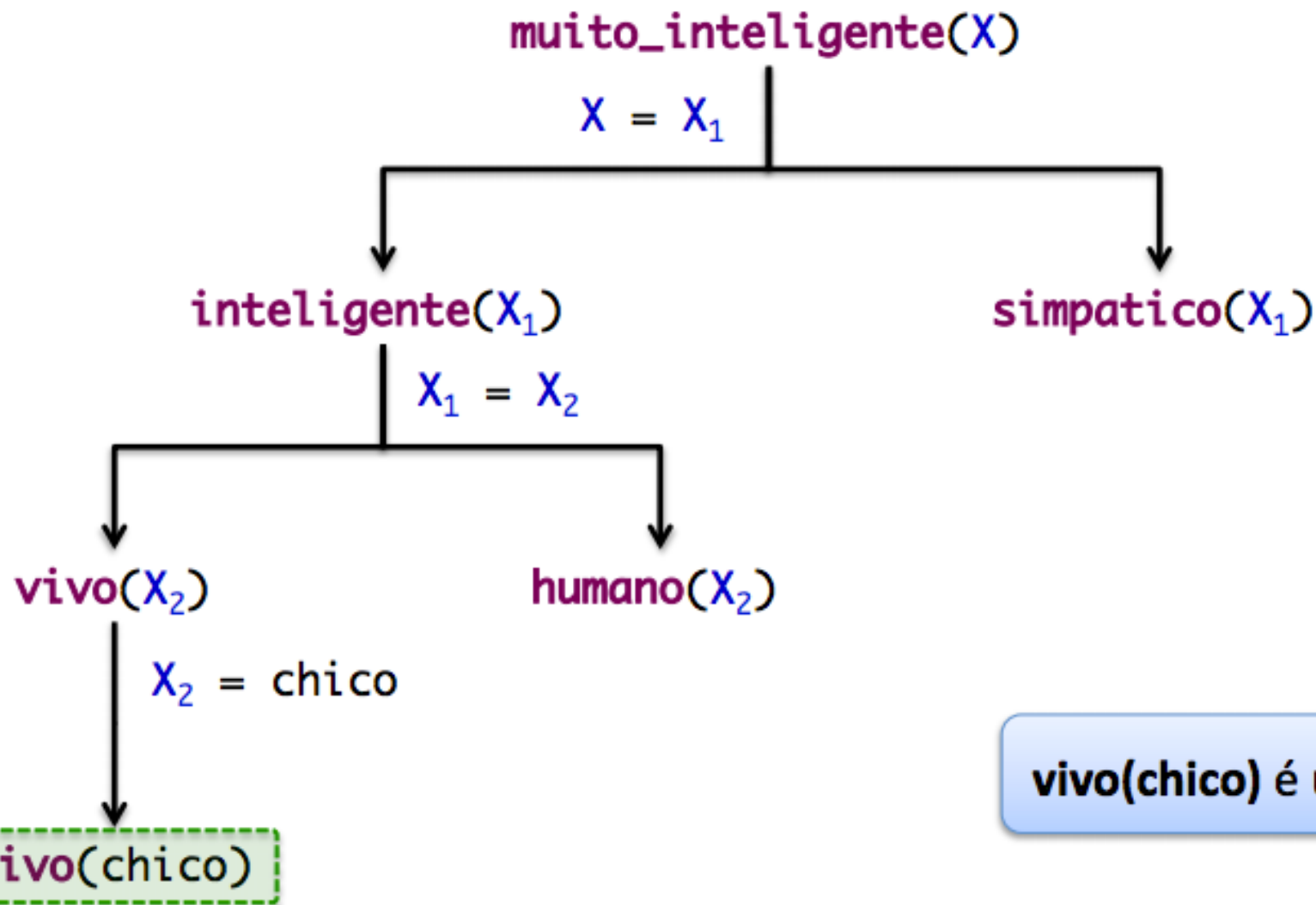


Expandida a primeira regra
(na ordem do programa)
que pôde ser unificada

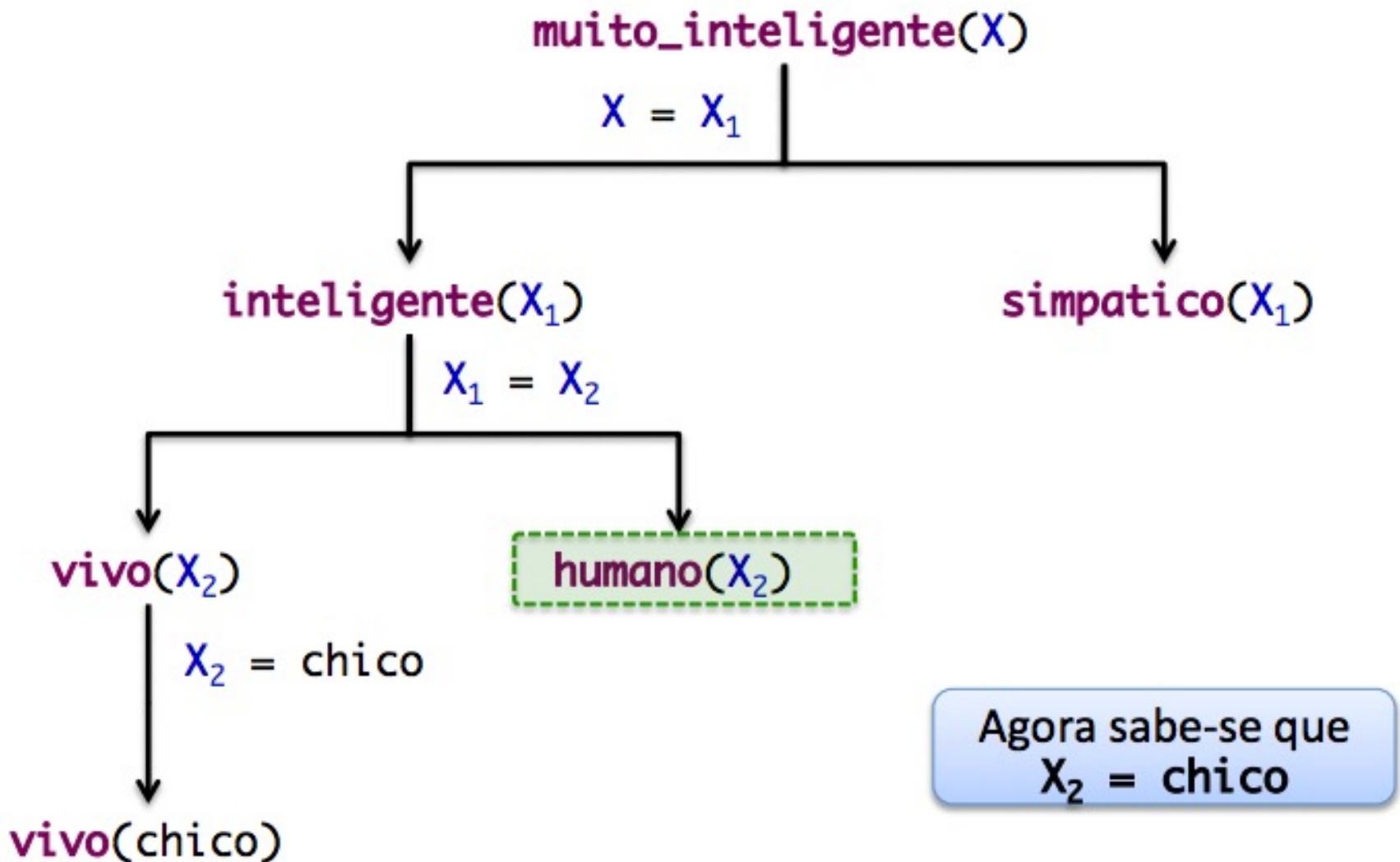
Exemplo 2



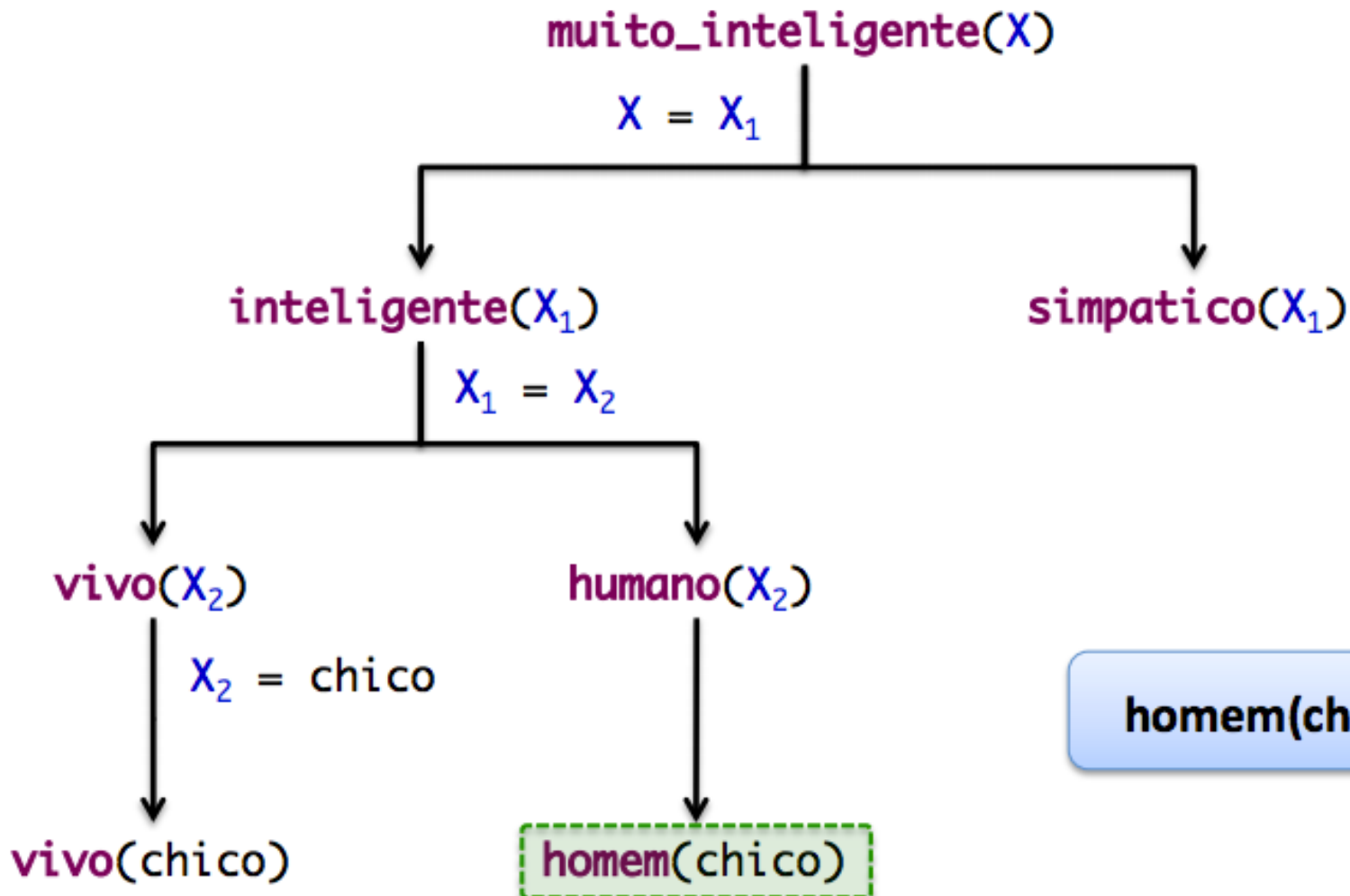
Exemplo 2



Exemplo 2

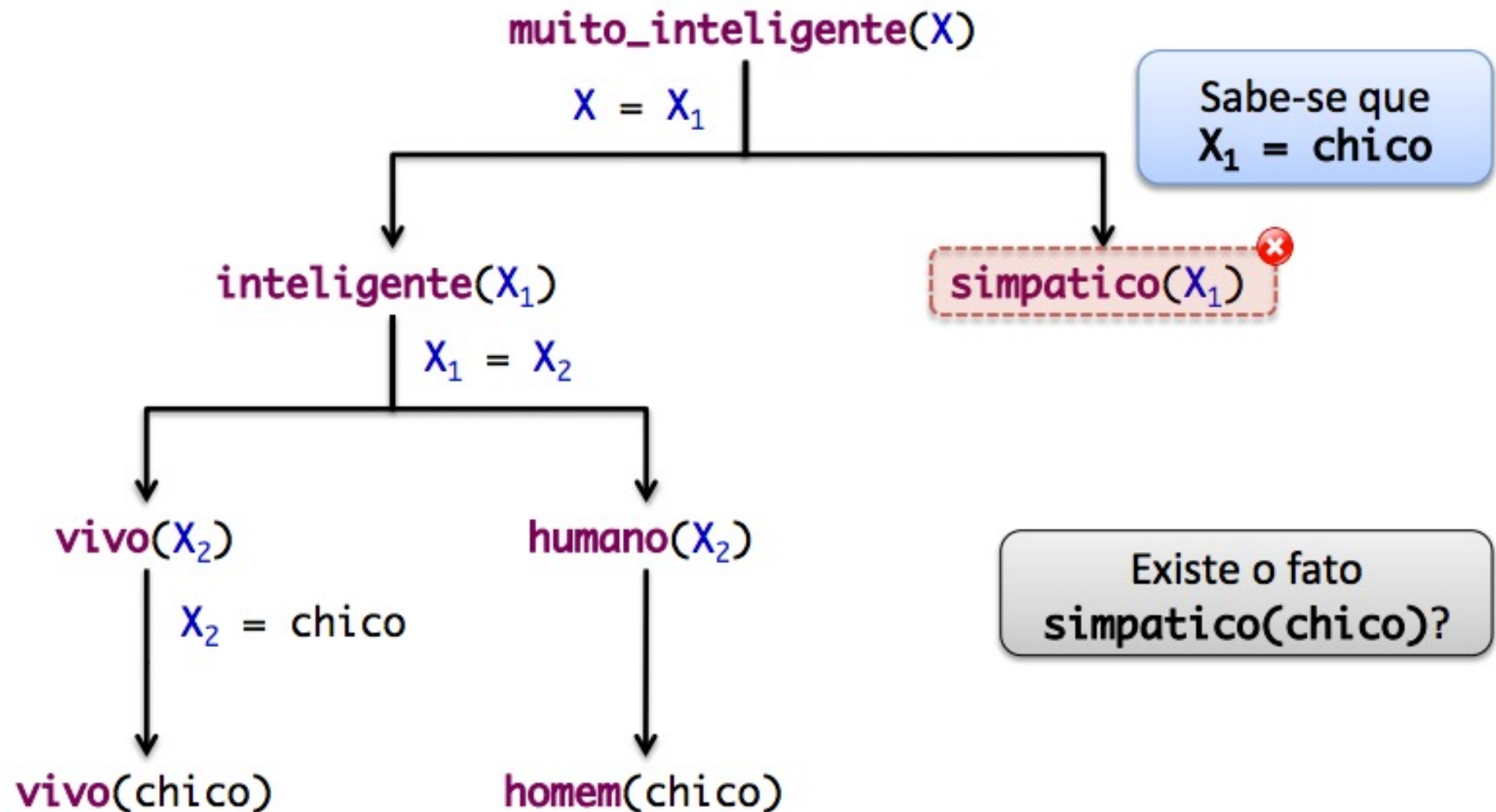


Exemplo 2

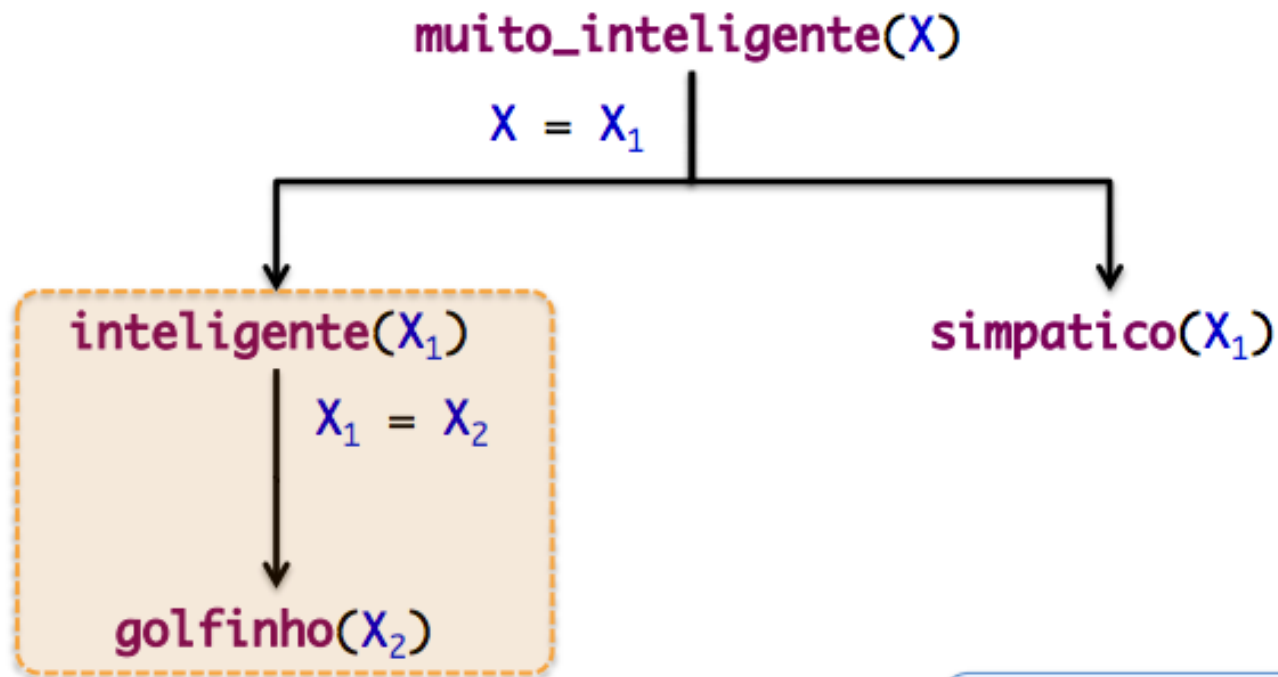


homem(chico) é um fato

Exemplo 2

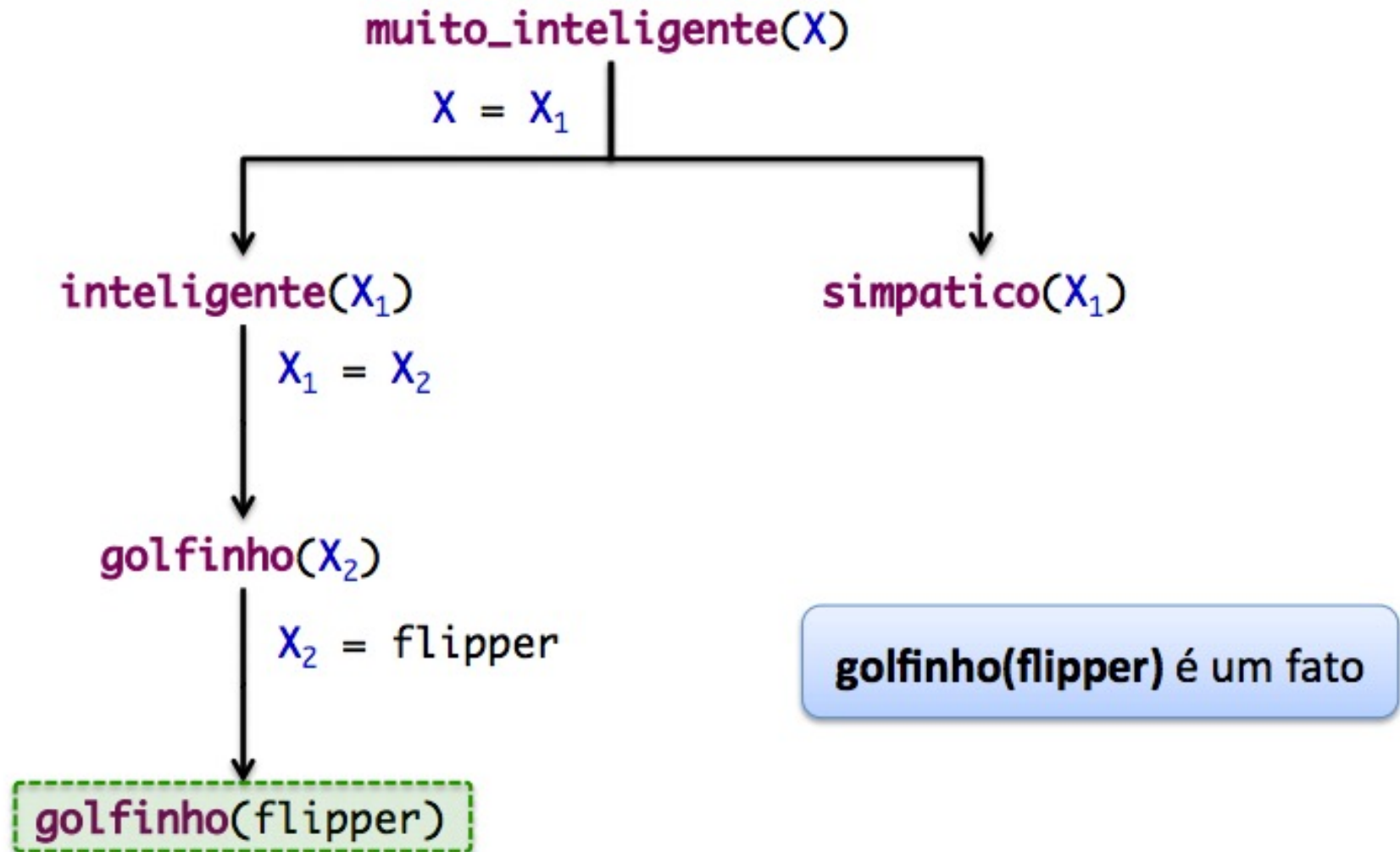


Exemplo 2

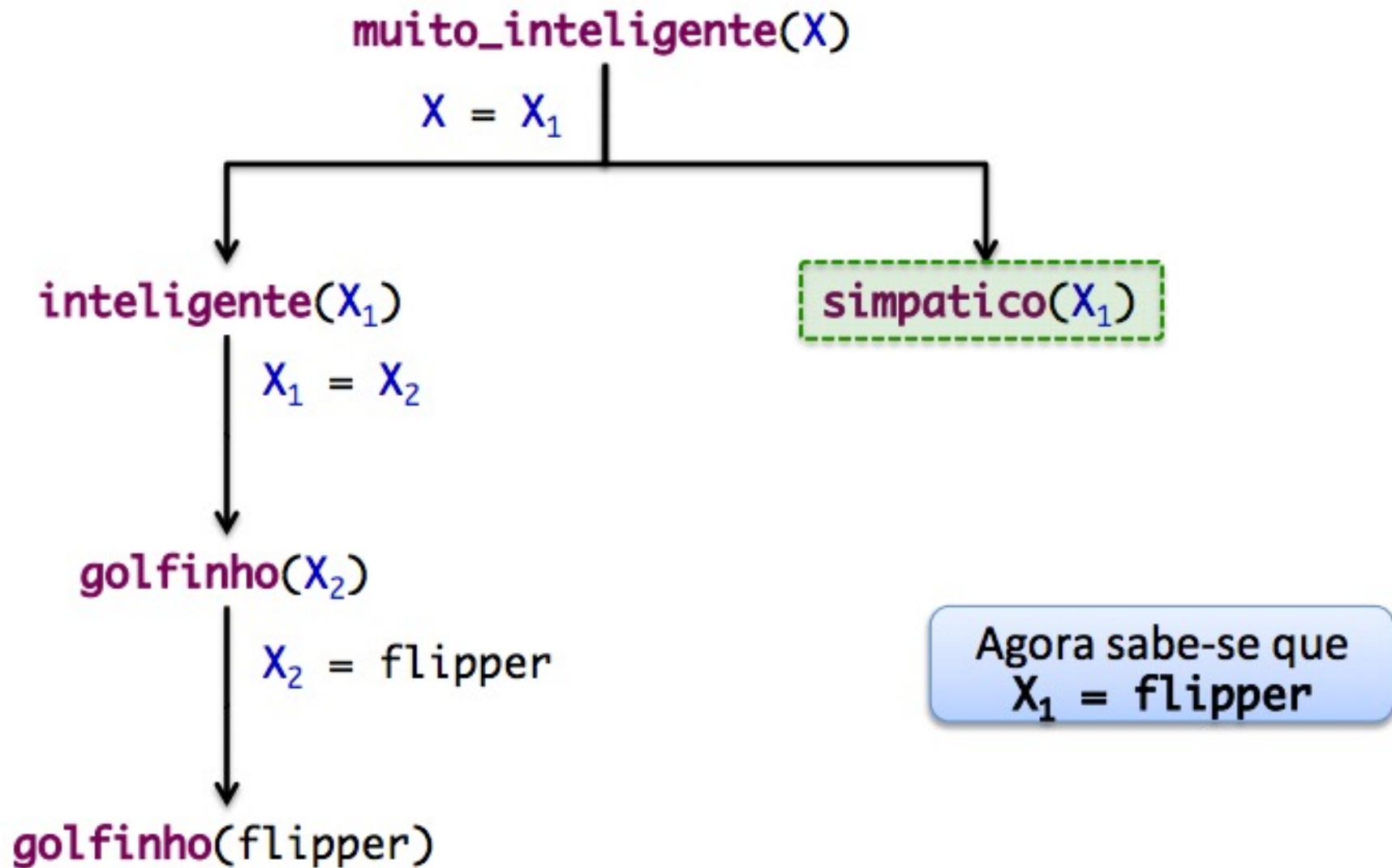


Backtracking até o ponto
onde havia outra alternativa

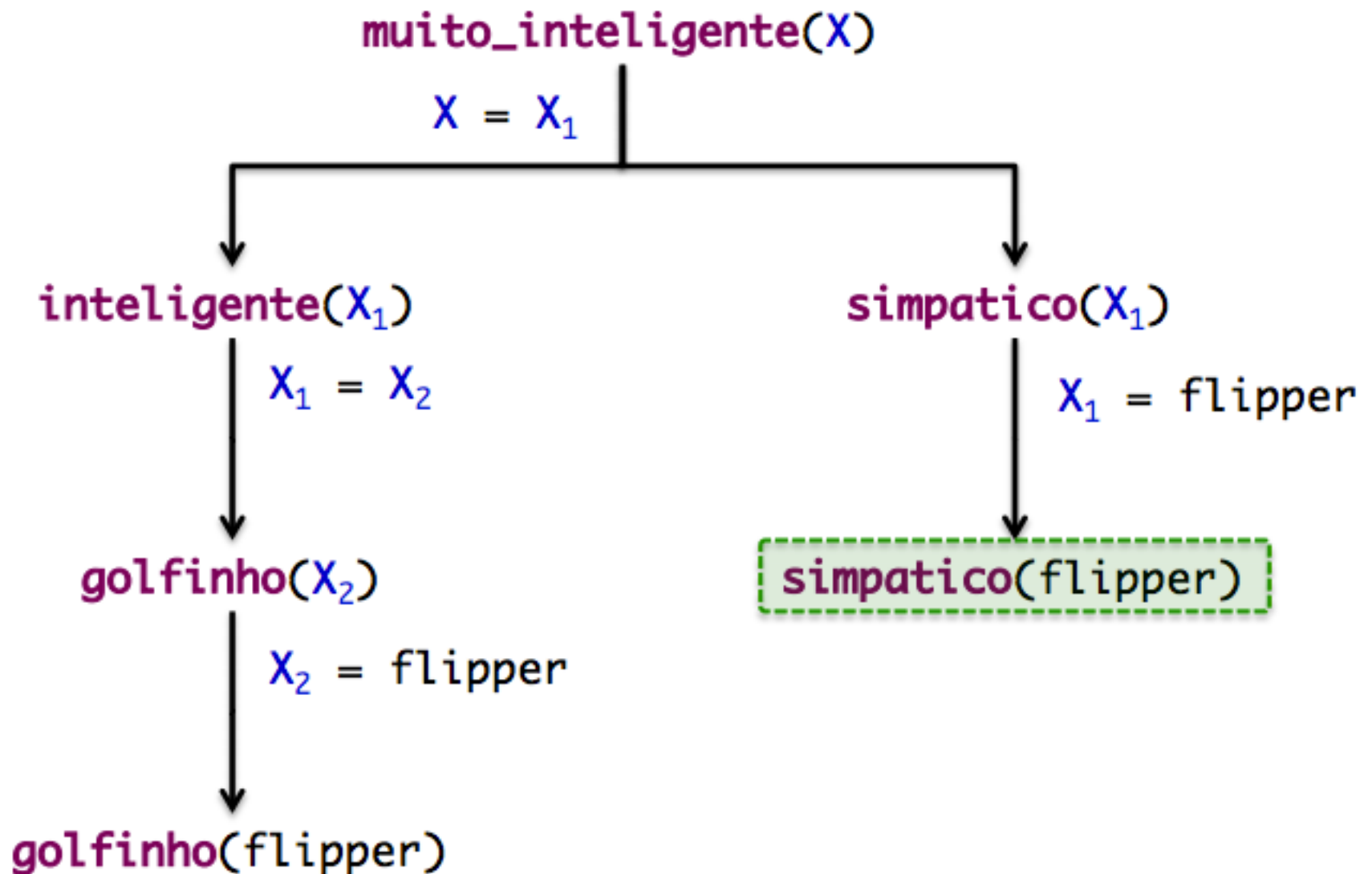
Exemplo 2



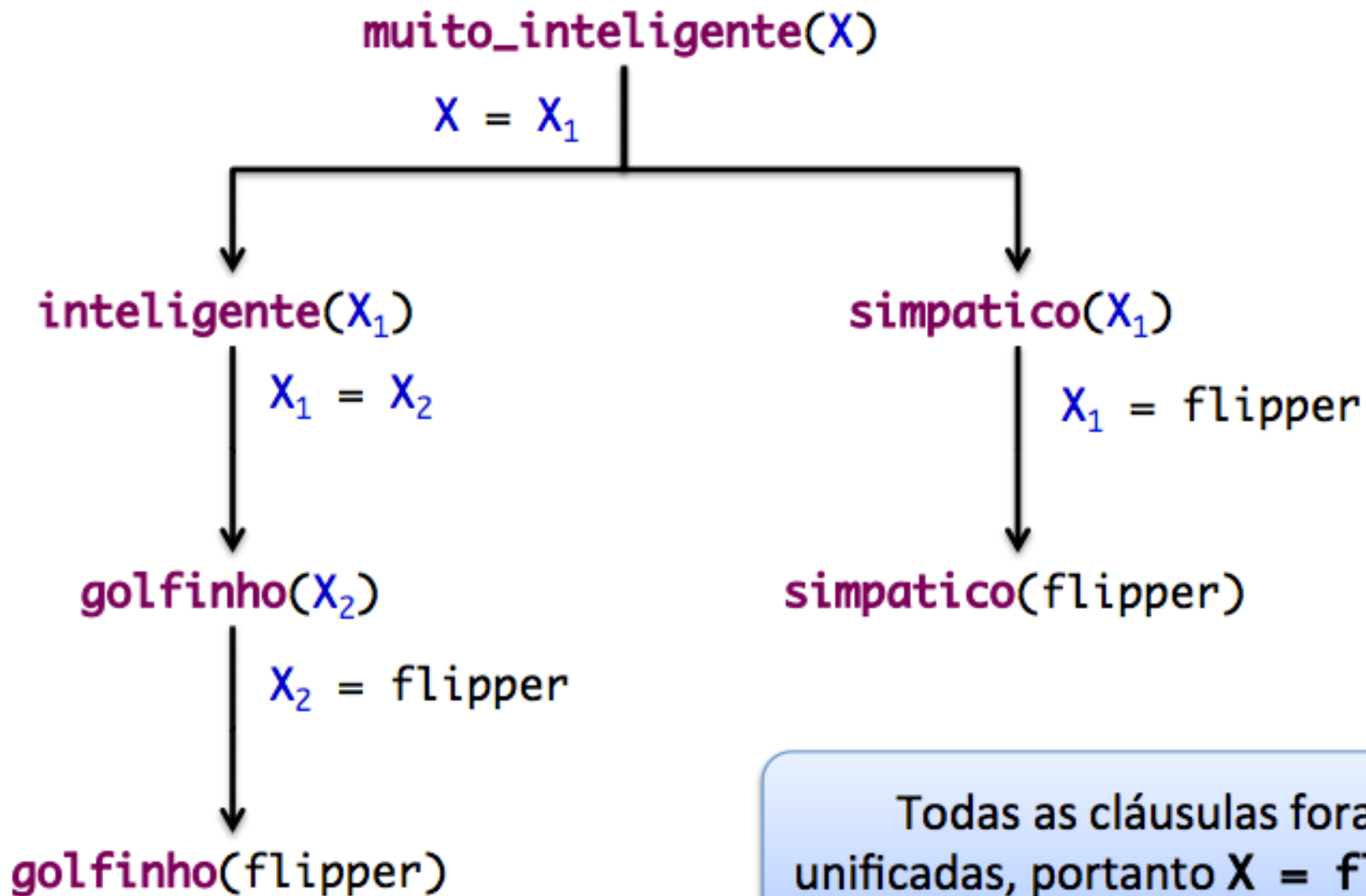
Exemplo 2



Exemplo 2



Exemplo 2



Aritmética

- Prolog tem uma série de predicados pré-definidos para cálculos lógicos e aritméticos:

comparação		cálculo	
<code>==</code>	Igual	<code>+</code>	Soma
<code>=\</code>	Diferente	<code>-</code>	Subtração
<code><</code>	Menor	<code>*</code>	Multiplicação
<code>></code>	Maior	<code>/</code>	Divisão
<code>=<</code>	Menor ou igual	<code>//</code>	Divisão inteira
<code>>=</code>	Maior ou igual	<code>mod</code>	Resto da divisão

- Para cálculos aritméticos é necessário usar o predicado especial ***is***:
 - Papel: transforma uma estrutura envolvendo operadores aritméticos no resultado desta expressão
 - ***X is Y/Z***

Exemplo: Incrementar um Número Inteiro

```
acc(X, R):-R is X +1.
```

```
main:-  
read(X),  
acc(X,Y),  
write(Y).
```

```
acc(X, R):-R = X +1.
```

```
main:-  
read(X),  
acc(X,Y),  
write(Y).
```

```
Consulta:  
? - main.  
4.  
5
```

```
Consulta:  
? - main.  
4.  
4+1
```

Exemplo

- Programa que lê o nome e notas de um aluno e verifica sua situação (APROVADO, REPROVADO ou NA FINAL)

```
leEntradas:- write("Digite o Nome: "),
              read(Nome),
              write("Digite as Notas: "),
              read(N1),
              read(N2),
              read(N3),
              situacao(N1,N2,N3,R),
              write(R).
```

Consulta:

? - leEntradas.

Digite o Nome:

"José"

Digite Notas:

6

8

9

APROVADO

```
situacao(N1,N2,N3,R):- Media is (N1+N2+N3)/3,
                       classifica(Media,R).
```

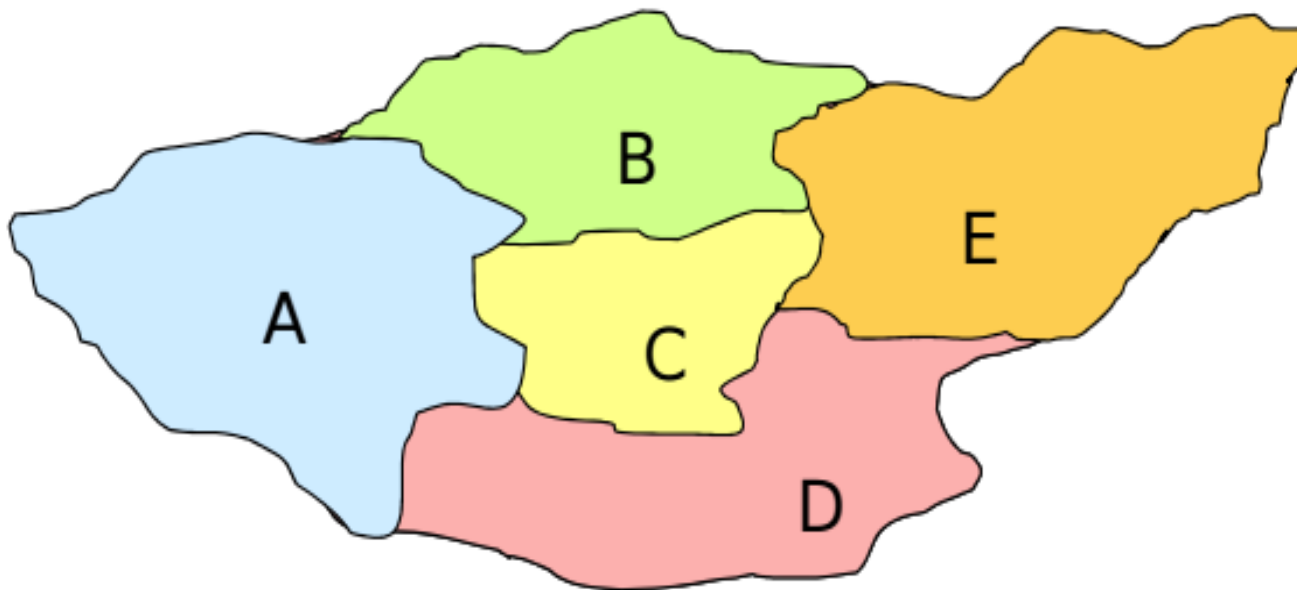
```
classifica(M, "Aprovado"):- M >= 7, !.
```

```
classifica(M, "Reprovado"):- M < 4, !.
```

```
classifica(M, "Final").
```

Exemplo: Coloração de Mapas

- Problema: para o mapa a seguir, como colorir usando no máximo quatro cores (azul, verde, amarelo e vermelho), de modo que regiões adjacentes tenham cores distintas?



Exemplo: Coloração de Mapas

- Solução:

- **Fatos?**

- As cores que podem ser usadas para coloração

- **Regras?**

- (A,B,C,D,E), componentes que correspondem às regiões do mapa, têm uma **coloração válida se:**
 - Cada um de seus componentes **é de uma cor**
 - Regiões **adjacentes** no mapa têm **cores distintas**

Exemplo: Coloração de Mapas

```
cor(azul).  
cor(verde).  
cor(amarelo).  
cor(vermelho).
```

```
coloracao(A,B,C,D,E) :-  
    cor(A), cor(B), cor(C), cor(D), cor(E),  
    A\=B, A\=C, A\=D, B\=C, B\=E, C\=D, C\=E, D\=E.
```

Consulta:

```
? - coloracao(A,B,C,D,E).  
A = E, E = azul,  
B = D, D = verde,  
C = amarelo
```

Exemplo: Geração de Binários

- Problema: gerar todos os números binários de três dígitos
- Solução:
 - Fatos:
 - Quais dígitos podem ser usados na composição de um número binário?
 - Regras:
 - Quais as restrições sobre componentes de uma estrutura representando um número binário de três dígitos?

Exemplo: Geração de Binários

digito(0).
digito(1).

binario(N) :- N = (A,B,C),
digito(A), digito(B), digito(C).

Consulta:

? - binario(N).

N = (0,0,0)

N = (0,0,1)

N = (0,1,0)

N = (0,1,1)

N = (1,0,0)

N = (1,0,1)

N = (1,1,0)

N = (1,1,1)