

# Programação Lógica

Paulo Torrens

paulotorrens@gnu.org

Departamento de Ciência da Computação  
Centro de Ciências e Tecnologias  
Universidade do Estado de Santa Catarina

2019/02

- Além de linguagens de programações **imperativas** e **funcionais**, podemos citar o conjunto das linguagens de programação **lógicas**
  - Enquanto linguagens funcionais são caracterizadas pelo uso de funções de alta ordem (isto é, funções que aceitam outras funções como parâmetro ou que retornam outras funções como resultado), **linguagens lógicas são caracterizadas pela representação de código como cláusulas lógicas**
  - Linguagens de programação lógicas costumam usar como base a lógica de primeira ordem, e programas representam relações entre termos
  - Exemplos: Prolog, Mercury, Picat
- Do ponto de vista teórico, a execução de um programa lógico é dada através de um algoritmo de unificação: o programa irá procurar uma substituição de variáveis capaz de satisfazer as relações informadas pelo código

- Além de linguagens de programações **imperativas** e **funcionais**, podemos citar o conjunto das linguagens de programação **lógicas**
  - Enquanto linguagens funcionais são caracterizadas pelo uso de funções de alta ordem (isto é, funções que aceitam outras funções como parâmetro ou que retornam outras funções como resultado), **linguagens lógicas são caracterizadas pela representação de código como cláusulas lógicas**
  - Linguagens de programação lógicas costumam usar como base a lógica de primeira ordem, e programas representam relações entre termos
  - Exemplos: Prolog, Mercury, Picat
- Do ponto de vista teórico, a execução de um programa lógico é dada através de um algoritmo de unificação: o programa irá procurar uma substituição de variáveis capaz de satisfazer as relações informadas pelo código

- Além de linguagens de programações **imperativas** e **funcionais**, podemos citar o conjunto das linguagens de programação **lógicas**
  - Enquanto linguagens funcionais são caracterizadas pelo uso de funções de alta ordem (isto é, funções que aceitam outras funções como parâmetro ou que retornam outras funções como resultado), **linguagens lógicas são caracterizadas pela representação de código como cláusulas lógicas**
  - Linguagens de programação lógicas costumam usar como base a lógica de primeira ordem, e programas representam relações entre termos
    - Exemplos: Prolog, Mercury, Picat
- Do ponto de vista teórico, a execução de um programa lógico é dada através de um algoritmo de unificação: o programa irá procurar uma substituição de variáveis capaz de satisfazer as relações informadas pelo código

- Além de linguagens de programações **imperativas** e **funcionais**, podemos citar o conjunto das linguagens de programação **lógicas**
  - Enquanto linguagens funcionais são caracterizadas pelo uso de funções de alta ordem (isto é, funções que aceitam outras funções como parâmetro ou que retornam outras funções como resultado), **linguagens lógicas são caracterizadas pela representação de código como cláusulas lógicas**
  - Linguagens de programação lógicas costumam usar como base a lógica de primeira ordem, e programas representam relações entre termos
  - Exemplos: Prolog, Mercury, Picat
- Do ponto de vista teórico, a execução de um programa lógico é dada através de um algoritmo de unificação: o programa irá procurar uma substituição de variáveis capaz de satisfazer as relações informadas pelo código

- Além de linguagens de programações **imperativas** e **funcionais**, podemos citar o conjunto das linguagens de programação **lógicas**
  - Enquanto linguagens funcionais são caracterizadas pelo uso de funções de alta ordem (isto é, funções que aceitam outras funções como parâmetro ou que retornam outras funções como resultado), **linguagens lógicas são caracterizadas pela representação de código como cláusulas lógicas**
  - Linguagens de programação lógicas costumam usar como base a lógica de primeira ordem, e programas representam relações entre termos
  - Exemplos: Prolog, Mercury, Picat
- Do ponto de vista teórico, **a execução de um programa lógico é dada através de um algoritmo de unificação**: o programa irá procurar uma **substituição** de variáveis capaz de satisfazer as relações informadas pelo código

# Introdução

```
1 % Prolog
2 father(marcos, marcelo).
3 father(marcos, pedro).
4 mother(maria, pedro).
5 mother(maria, joana).
6
7 sibling(X, Y) :- father(F, X), father(F, Y);
8                 mother(M, X), mother(M, Y).
```

```
? sibling(joana, marcelo). % no
```

```
? sibling(joana, X). % yes, X = pedro
```

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$



- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

- Na lógica e na ciência da computação, a **unificação** é um processo para encontrar igualdade entre dois termos de uma lógica simbólica através da **substituição** de variáveis
  - Além de servir como base para a programação lógica, também é usado em algumas outras áreas
  - Por exemplo, é usada em algoritmos de inferência de tipos, como o usado pela linguagem Haskell
- A unificação retorna uma substituição, que é um mapa de variáveis para termos
  - Usamos “termos” para nos referir ao resultado desejado; por exemplo, no caso do inferidor de tipos de Haskell, os termos serão exatamente os tipos
- Por exemplo:  $a \rightarrow b \sim Bool \rightarrow Int$ 
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe:  $\{ a \mapsto Bool, b \mapsto Int \}$

# Substituições

- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em  $\theta$  devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$



- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em  $\theta$  devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$

# Substituições

- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em  $\theta$  devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$

- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em  $\theta$  devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$

- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em  $\theta$  devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- **Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral**
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$

- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em  $\theta$  devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- **Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral**
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$

- Por convenção, anotamos uma substituição como  $\theta$ , a qual possui a forma  $\{ x_1 \mapsto e_1, \dots, x_n \mapsto e_n \}$ 
  - Onde  $x_i$  são as variáveis que podem ser substituídas, e  $e_i$  são os termos
- Podemos aplicar uma substituição  $\theta$  a um termo  $e$  da forma  $\theta e$ , que representa a reescrita do termo  $e$  com todas as variáveis contidas em *theta* devidamente substituídas
  - Por exemplo,  $\{ a \mapsto int, c \mapsto bool \}(a \rightarrow b) = int \rightarrow b$
- Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral
  - Por exemplo, o unificador mais geral entre  $a \rightarrow bool$  e  $int \rightarrow b$  é  $\theta = \{ a \mapsto int, b \mapsto bool \}$ , pois  $\theta(a \rightarrow bool) = \theta(int \rightarrow b)$
- Podemos compor duas substituições:  $\theta_1 \circ \theta_2$ , retornando a união dos dados, tendo a primeira substituição  $\theta_1$  aplicada a todos os termos da segunda substituição  $\theta_2$

# Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$\tau$	$::=$	<code>int</code>	Números inteiros
		<code>a</code>	Variáveis
		$\tau \rightarrow \tau$	Funções

Ou, em Haskell:

```
1 data Type = TypeInt
2           | TypeVar Name
3           | TypeArrow Type Type
```

- Podemos definir uma operação de unificação que encontra o unificador mais geral usando tipos como termos, na forma da expressão  $\tau_1 \sim \tau_2 = \theta$
- Essa operação é usada no sistema de tipos de Hindley-Milner, um subconjunto do sistema de tipos de Haskell

# Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$\tau$	$::=$	$\text{int}$	Números inteiros
	$ $	$a$	Variáveis
	$ $	$\tau \rightarrow \tau$	Funções

Ou, em Haskell:

```
1 data Type = TypeInt
2           | TypeVar Name
3           | TypeArrow Type Type
```

- Podemos definir uma operação de unificação que encontra o unificador mais geral usando tipos como termos, na forma da expressão  $\tau_1 \sim \tau_2 = \theta$
- Essa operação é usada no sistema de tipos de Hindley-Milner, um subconjunto do sistema de tipos de Haskell



# Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$\tau$	$::=$	<code>int</code>	Números inteiros
		<code>a</code>	Variáveis
		$\tau \rightarrow \tau$	Funções

Ou, em Haskell:

```
1 data Type = TypeInt
2           | TypeVar Name
3           | TypeArrow Type Type
```

- Podemos definir uma operação de unificação que encontra o unificador mais geral usando tipos como termos, na forma da expressão  $\tau_1 \sim \tau_2 = \theta$
- Essa operação é usada no sistema de tipos de Hindley-Milner, um subconjunto do sistema de tipos de Haskell

# Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$\tau$	$::=$	<code>int</code>	Números inteiros
		<code>a</code>	Variáveis
		$\tau \rightarrow \tau$	Funções

Ou, em Haskell:

```
1 data Type = TypeInt
2           | TypeVar Name
3           | TypeArrow Type Type
```

- Podemos definir uma operação de unificação que encontra o unificador mais geral usando tipos como termos, na forma da expressão  $\tau_1 \sim \tau_2 = \theta$
- Essa operação é usada no sistema de tipos de Hindley-Milner, um subconjunto do sistema de tipos de Haskell

# Exemplo: sistema de tipos

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } \tau}{a \sim \tau = \{ a \mapsto \tau \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } \tau}{\tau \sim a = \{ a \mapsto \tau \}} \text{ (RIGHT)}$$

# Exemplo: sistema de tipos

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } \tau}{a \sim \tau = \{ a \mapsto \tau \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } \tau}{\tau \sim a = \{ a \mapsto \tau \}} \text{ (RIGHT)}$$

# Exemplo: sistema de tipos

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } \tau}{a \sim \tau = \{ a \mapsto \tau \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } \tau}{\tau \sim a = \{ a \mapsto \tau \}} \text{ (RIGHT)}$$

## Exemplo: sistema de tipos

- Podemos unificar o tipo `int` com o tipo `int`; não há variáveis para substituir, porém os termos já são iguais, retornando uma substituição vazia

$$\frac{}{int \sim int = \{ \}} \text{ (INT)}$$

- Se podemos unificar os argumentos de uma função, e com essa substituição podemos unificar seus resultados, então podemos unificar as funções compondo os resultados

$$\frac{\tau_1 \sim \tau_2 = \theta_1 \quad \theta_1 \sigma_1 \sim \theta_1 \sigma_2 = \theta_2}{\tau_1 \rightarrow \sigma_1 \sim \tau_2 \rightarrow \sigma_2 = \theta_2 \circ \theta_1} \text{ (ARROW)}$$

## Exemplo: sistema de tipos

- Podemos unificar o tipo `int` com o tipo `int`; não há variáveis para substituir, porém os termos já são iguais, retornando uma substituição vazia

$$\frac{}{int \sim int = \{ \}} \text{ (INT)}$$

- Se podemos unificar os argumentos de uma função, e com essa substituição podemos unificar seus resultados, então podemos unificar as funções compondo os resultados

$$\frac{\tau_1 \sim \tau_2 = \theta_1 \quad \theta_1 \sigma_1 \sim \theta_1 \sigma_2 = \theta_2}{\tau_1 \rightarrow \sigma_1 \sim \tau_2 \rightarrow \sigma_2 = \theta_2 \circ \theta_1} \text{ (ARROW)}$$

# Unificação em Prolog

- Considere uma versão simplificada de Prolog; termos dentro da linguagem podem ter as seguintes formas:
  - Átomos: nomes simbólicos representando valores fixos na linguagem, sempre começando com letra minúscula
  - Variáveis: variáveis que podem ser trocadas por outros termos, sempre começando com letra maiúscula
  - Predicados: uma expressão, similar a uma chamada de função, contendo uma sequência de parâmetros
- Podemos considerar a seguinte gramática simplificada para termos:

$e$	$::=$	$x$	Átomos
		$a$	Variáveis
		$x(e_1, \dots, e_n)$	Predicados
- O trabalho do ambiente é, dado um predicado e uma lista de regras, verificar se existe uma substituição capaz de tornar o predicado verdadeiro conforme as regras, através de unificação



# Unificação em Prolog

- Considere uma versão simplificada de Prolog; termos dentro da linguagem podem ter as seguintes formas:
  - Átomos: nomes simbólicos representando valores fixos na linguagem, sempre começando com letra minúscula
  - Variáveis: variáveis que podem ser trocadas por outros termos, sempre começando com letra maiúscula
  - Predicados: uma expressão, similar a uma chamada de função, contendo uma sequência de parâmetros

- Podemos considerar a seguinte gramática simplificada para termos:

$e$	$::=$	$x$	Átomos
		$a$	Variáveis
		$x(e_1, \dots, e_n)$	Predicados

- O trabalho do ambiente é, dado um predicado e uma lista de regras, verificar se existe uma substituição capaz de tornar o predicado verdadeiro conforme as regras, através de unificação

# Unificação em Prolog

- Considere uma versão simplificada de Prolog; termos dentro da linguagem podem ter as seguintes formas:
  - Átomos: nomes simbólicos representando valores fixos na linguagem, sempre começando com letra minúscula
  - Variáveis: variáveis que podem ser trocadas por outros termos, sempre começando com letra maiúscula
  - Predicados: uma expressão, similar a uma chamada de função, contendo uma sequência de parâmetros

- Podemos considerar a seguinte gramática simplificada para termos:

$e$	$::=$	$x$	Átomos
		$a$	Variáveis
		$x(e_1, \dots, e_n)$	Predicados

- O trabalho do ambiente é, dado um predicado e uma lista de regras, verificar se existe uma substituição capaz de tornar o predicado verdadeiro conforme as regras, através de unificação

# Unificação em Prolog

- Considere uma versão simplificada de Prolog; termos dentro da linguagem podem ter as seguintes formas:
  - Átomos: nomes simbólicos representando valores fixos na linguagem, sempre começando com letra minúscula
  - Variáveis: variáveis que podem ser trocadas por outros termos, sempre começando com letra maiúscula
  - Predicados: uma expressão, similar a uma chamada de função, contendo uma sequência de parâmetros
- Podemos considerar a seguinte gramática simplificada para termos:

$e$	$::=$	$x$	Átomos
		$a$	Variáveis
		$x(e_1, \dots, e_n)$	Predicados

- O trabalho do ambiente é, dado um predicado e uma lista de regras, verificar se existe uma substituição capaz de tornar o predicado verdadeiro conforme as regras, através de unificação

# Unificação em Prolog

- Considere uma versão simplificada de Prolog; termos dentro da linguagem podem ter as seguintes formas:
  - Átomos: nomes simbólicos representando valores fixos na linguagem, sempre começando com letra minúscula
  - Variáveis: variáveis que podem ser trocadas por outros termos, sempre começando com letra maiúscula
  - Predicados: uma expressão, similar a uma chamada de função, contendo uma sequência de parâmetros
- Podemos considerar a seguinte gramática simplificada para termos:

$e$	$::=$	$x$	Átomos
		$ $	
		$a$	Variáveis
		$ $	
		$x(e_1, \dots, e_n)$	Predicados

- O trabalho do ambiente é, dado um predicado e uma lista de regras, verificar se existe uma substituição capaz de tornar o predicado verdadeiro conforme as regras, através de unificação

# Unificação em Prolog

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- De forma parecida, dois átomos iguais podem ser unificados, nos lembrando que átomos não podem ser substituídos

$$\frac{}{x \sim x = \{ \}} \text{ (ATOM)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } e}{a \sim e = \{ a \mapsto e \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } e}{e \sim a = \{ a \mapsto e \}} \text{ (RIGHT)}$$

# Unificação em Prolog

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- De forma parecida, dois átomos iguais podem ser unificados, nos lembrando que átomos não podem ser substituídos

$$\frac{}{x \sim x = \{ \}} \text{ (ATOM)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } e}{a \sim e = \{ a \mapsto e \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } e}{e \sim a = \{ a \mapsto e \}} \text{ (RIGHT)}$$

# Unificação em Prolog

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- De forma parecida, dois átomos iguais podem ser unificados, nos lembrando que átomos não podem ser substituídos

$$\frac{}{x \sim x = \{ \}} \text{ (ATOM)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } e}{a \sim e = \{ a \mapsto e \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } e}{e \sim a = \{ a \mapsto e \}} \text{ (RIGHT)}$$

# Unificação em Prolog

- Duas variáveis iguais podem ser unificadas

$$\frac{}{a \sim a = \{ \}} \text{ (REFL)}$$

- De forma parecida, dois átomos iguais podem ser unificados, nos lembrando que átomos não podem ser substituídos

$$\frac{}{x \sim x = \{ \}} \text{ (ATOM)}$$

- Uma variável à esquerda pode ser unificada se não aparecer livre na direita

$$\frac{a \text{ não aparece livre em } e}{a \sim e = \{ a \mapsto e \}} \text{ (LEFT)}$$

- Uma variável à direita pode ser unificada se não aparecer livre na esquerda

$$\frac{a \text{ não aparece livre em } e}{e \sim a = \{ a \mapsto e \}} \text{ (RIGHT)}$$



- Considerando que os nomes e os tamanhos sejam iguais, e que possamos unificar seus corpos retornando uma substituição, então podemos unificar dois predicados

$$\frac{[e_1, \dots, e_n] \sim [f_1, \dots, f_n] = \theta}{x(e_1, \dots, e_n) \sim x(f_1, \dots, f_n) = \theta} \text{ (PRED)}$$

- Se dois corpos de predicados são vazios, podemos unificá-los retornando uma substituição vazia

$$\frac{}{[] \sim [] = \{ \}} \text{ (NIL)}$$

- Se dois corpos de predicados forem células, caso possamos unificar suas cabeças e suas caudas, podemos unificar as listas compondo os resultados

$$\frac{x \sim y = \theta_1 \quad \theta_1 x_s \sim \theta_1 y_s = \theta_2}{(x : x_s) \sim (y : y_s) = \theta_2 \circ \theta_1} \text{ (CONS)}$$

- Considerando que os nomes e os tamanhos sejam iguais, e que possamos unificar seus corpos retornando uma substituição, então podemos unificar dois predicados

$$\frac{[e_1, \dots, e_n] \sim [f_1, \dots, f_n] = \theta}{x(e_1, \dots, e_n) \sim x(f_1, \dots, f_n) = \theta} \text{ (PRED)}$$

- Se dois corpos de predicados são vazios, podemos unificá-los retornando uma substituição vazia

$$\frac{}{[] \sim [] = \{ \}} \text{ (NIL)}$$

- Se dois corpos de predicados forem células, caso possamos unificar suas cabeças e suas caudas, podemos unificar as listas compondo os resultados

$$\frac{x \sim y = \theta_1 \quad \theta_1 x_s \sim \theta_1 y_s = \theta_2}{(x : x_s) \sim (y : y_s) = \theta_2 \circ \theta_1} \text{ (CONS)}$$

- Considerando que os nomes e os tamanhos sejam iguais, e que possamos unificar seus corpos retornando uma substituição, então podemos unificar dois predicados

$$\frac{[e_1, \dots, e_n] \sim [f_1, \dots, f_n] = \theta}{x(e_1, \dots, e_n) \sim x(f_1, \dots, f_n) = \theta} \text{ (PRED)}$$

- Se dois corpos de predicados são vazios, podemos unificá-los retornando uma substituição vazia

$$\frac{}{[] \sim [] = \{ \}} \text{ (NIL)}$$

- Se dois corpos de predicados forem células, caso possamos unificar suas cabeças e suas caudas, podemos unificar as listas compondo os resultados

$$\frac{x \sim y = \theta_1 \quad \theta_1 x s \sim \theta_1 y s = \theta_2}{(x : xs) \sim (y : ys) = \theta_2 \circ \theta_1} \text{ (CONS)}$$

- Além da unificação, usada para verificar possíveis respostas, um programa em Prolog deve procurar uma solução em uma sequência de regras informadas
- O algoritmo *resolve* pode ser descrito da seguinte forma:
  - As entradas são *goal*, o predicado que está tentando ser provado, e *rules*, uma lista de regras existentes no sistema
  - Um valor *rules'* é criado, contendo uma versão de *rules* **com novas variáveis**, necessário para evitar colisões de nomes
  - Para cada regra *rule* contida em *rules'*, dividida em uma cabeça *head* e um corpo *body*...
    - Caso a unificação *goal*  $\sim$  *head* retorne um unificador mais geral *mgu*, então os unificadores contidos em *resolveBody rules' mgu body* devem ser adicionados à lista de respostas
    - Caso a unificação acima falhe, o processo apenas é continuado sem adicionar nenhuma possível resposta

- O algoritmo *resolveBody* pode ser descrito da seguinte forma:
  - As entradas são *rule*, a sequência de regras existentes no sistema,  $\theta_1$ , o unificador mais geral até o momento, e uma lista de predicados
  - Caso a lista esteja vazia, ou seja, chegamos ao fim do corpo do predicado, o resultado é apenas uma lista contendo  $\theta_1$
  - Um valor *rules'* é criado, contendo uma versão de *rules* **com novas variáveis**, necessário para evitar colisões de nomes
  - Caso a lista seja uma célula ( $p : ps$ ), devemos criar uma lista *sub* com o resultado de *resolve* usando  $\theta_1 p$  (aplicando a substituição) como *goal* e *rules'*
    - Para cada item  $\theta_2$  retornado acima, os resultados contidos em *resolveBody* com as regras, com parâmetros *rules'*,  $\theta_2 \circ \theta_1$  (composição) e *ps* devem ser adicionados à resposta