Programação Lógica

Paulo Torrens

paulotorrens@gnu.org

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

2019/02

Introdução

- 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 outros 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

BY NC ND

- 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 outros 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

BY NC ND

Introdução

- 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 outros 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

BY NC ND

- 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 outros 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

BY NC ND

Introdução

- 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 outros 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

BY NC ND

```
% Prolog
father(marcos, marcelo).
father(marcos, pedro).
mother(maria, pedro).
mother(maria, joana).

sibling(X, Y) :- father(F, X), father(F, Y);
                 mother(M, X), mother(M, Y).
```

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

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

BY NC ND

Unificação

- 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: $a \to b \sim Bool \to Int$
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

Unificação

- 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: $a \to b \sim Bool \to Int$
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

Unificação

- 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: $a \to b \sim Bool \to Int$
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

- 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: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

Unificação

- 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: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

Unificação

- 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: $a \to b \sim Bool \to Int$
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

Unificação

- 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: $a \to b \sim Bool \to Int$
  - Queremos encontrar uma substituição que torne os termos iguais
  - Nesse caso, ela existe: $\{ a \mapsto Bool, b \mapsto Int \}$

BY NC ND

Unificação

- 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: $\{\ a \mapsto Bool, b \mapsto Int\ \}$

BY NC ND

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{\ x_1 \mapsto e_1, ..., x_n \mapsto e_n\ \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, 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** θ_1 **aplicada a todos os termos da segunda substituição** θ_2

BY NC ND

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{\ x_1 \mapsto e_1, ..., x_n \mapsto e_n\ \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, que representa a reescrita do termo e com todas as variáveis contidas em *theta* devidamente substituídas
  - Por exemplo, $\{\ a \mapsto int, c \mapsto bool\ \}(a \to b) = int \to b$
- **Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral**
  - Por exemplo, o unificador mais geral entre $a \to bool$ e $int \to b$ é $\theta = \{\ a \mapsto int, b \mapsto bool\ \}$, pois $\theta(a \to bool) = \theta(int \to b)$
- **Podemos compor duas substituições:** $\theta_1 \circ \theta_2$, retornando a união dos dados, **tendo a primeira substituição** θ_1 **aplicada a todos os termos da segunda substituição** θ_2

BY NC ND

Substituições

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{ \; x_1 \mapsto e_1, ..., x_n \mapsto e_n \; \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, que representa a reescrita do termo e com todas as variáveis contidas em *theta* devidamente substituídas
  - Por exemplo, $\{ \; a \mapsto int, c \mapsto bool \; \}(a \to b) = int \to b$
- **Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral**
  - Por exemplo, o unificador mais geral entre $a \to bool$ e $int \to b$ é $\theta = \{ \; a \mapsto int, b \mapsto bool \; \}$, pois $\theta(a \to bool) = \theta(int \to b)$
- **Podemos compor duas substituições:** $\theta_1 \circ \theta_2$, retornando a união dos dados, **tendo a primeira substituição** θ_1 **aplicada a todos os termos da segunda substituição** θ_2

BY NC ND

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{ x_1 \mapsto e_1, ..., x_n \mapsto e_n \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, que representa a reescrita do termo e com todas as variáveis contidas em *theta* devidamente substituídas
  - Por exemplo, $\{ a \mapsto int, c \mapsto bool \}(a \to b) = int \to b$
- Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral
  - Por exemplo, o unificador mais geral entre $a \to bool$ e $int \to b$ é $\theta = \{ a \mapsto int, b \mapsto bool \}$, pois $\theta(a \to bool) = \theta(int \to b)$
- **Podemos compor duas substituições:** $\theta_1 \circ \theta_2$, retornando a união dos dados, **tendo a primeira substituição** θ_1 **aplicada a todos os termos da segunda substituição** θ_2

BY NC ND

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{ x_1 \mapsto e_1, ..., x_n \mapsto e_n \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, que representa a reescrita do termo e com todas as variáveis contidas em *theta* devidamente substituídas
  - Por exemplo, $\{ a \mapsto int, c \mapsto bool \}(a \to b) = int \to b$
- **Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral**
  - Por exemplo, o unificador mais geral entre $a \to bool$ e $int \to b$ é $\theta = \{ a \mapsto int, b \mapsto bool \}$, pois $\theta(a \to bool) = \theta(int \to b)$
- **Podemos compor duas substituições:** $\theta_1 \circ \theta_2$, retornando a união dos dados, **tendo a primeira substituição θ_1 aplicada a todos os termos da segunda substituição θ_2**

BY NC ND

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{ x_1 \mapsto e_1, ..., x_n \mapsto e_n \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, que representa a reescrita do termo e com todas as variáveis contidas em *theta* devidamente substituídas
  - Por exemplo, $\{ a \mapsto int, c \mapsto bool \}(a \to b) = int \to b$
- **Chamamos a menor substituição capaz de tornar dois termos iguais de unificador mais geral**
  - Por exemplo, o unificador mais geral entre $a \to bool$ e $int \to b$ é $\theta = \{ a \mapsto int, b \mapsto bool \}$, pois $\theta(a \to bool) = \theta(int \to b)$
- **Podemos compor duas substituições:** $\theta_1 \circ \theta_2$, retornando a união dos dados, **tendo a primeira substituição θ_1 aplicada a todos os termos da segunda substituição θ_2**

BY NC ND

- Por convenção, anotamos uma substituição como θ, a qual possui a forma $\{\ x_1 \mapsto e_1, ..., x_n \mapsto e_n\ \}$
  - Onde x_i são as variáveis que podem ser substituídas, e e_i são os termos
- Podemos aplicar uma substituição θ a um termo e da forma θe, 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 θ_1 aplicada a todos os termos da segunda substituição θ_2**

BY NC ND

Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$$
\begin{array}{lcll}
\tau & ::= & \text{int} & \text{Números inteiros} \\
 & | & a & \text{Variáveis} \\
 & | & \tau \to \tau & \text{Funções}
\end{array}
$$

Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar Name
          | TypeArrow Type Type
```

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

cc BY NC ND

- Considere a seguinte gramática simplificada para tipos:

$$
\begin{array}{lcll}
\tau & ::= & \mathsf{int} & \text{Números inteiros} \\
 & \mid & a & \text{Variáveis} \\
 & \mid & \tau \rightarrow \tau & \text{Funções}
\end{array}
$$

Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar Name
          | TypeArrow Type Type
```

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

BY NC ND

Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$$
\begin{array}{lcll}
\tau & ::= & \mathsf{int} & \text{Números inteiros} \\
 & | & a & \text{Variáveis} \\
 & | & \tau \to \tau & \text{Funções}
\end{array}
$$

  Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar Name
          | TypeArrow Type Type
```

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

CC BY NC ND

- Considere a seguinte gramática simplificada para tipos:

τ	$::=$	int	Números inteiros
	$\mid$	a	Variáveis
	$\mid$	$\tau \rightarrow \tau$	Funções

  Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar Name
          | TypeArrow Type Type
```

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

cc BY NC ND

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

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

BY NC ND

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

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

BY NC ND

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

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

BY NC ND

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

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

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

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

cc BY NC ND

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

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

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

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

BY NC ND

- 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
	$\mid$	a	Variáveis
	$\mid$	$x(e_1, ..., e_n)$	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

BY NC ND

- 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:

$$
\begin{array}{lcll}
e & ::= & x & \text{Átomos} \\
  & | & a & \text{Variáveis} \\
  & | & x(e_1, ..., e_n) & \text{Predicados}
\end{array}
$$

- 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

BY NC ND

- 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:

$$
\begin{array}{lcll}
e & ::= & x & \text{Átomos} \\
 & | & a & \text{Variáveis} \\
 & | & x(e_1, ..., e_n) & \text{Predicados}
\end{array}
$$

- 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

BY NC ND

- 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

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

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

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

BY NC ND

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

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

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

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

cc BY NC ND

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

$$\frac{x = y \qquad n = m \qquad [e_1, ..., e_n] \sim [f_1, ..., f_m] = \theta}{x(e_1, ..., e_n) \sim y(f_1, ..., f_m) = \theta} \text{ (PRED)}$$

- 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

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

BY NC ND

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

$$\frac{x = y \qquad n = m \qquad [e_1, ..., e_n] \sim [f_1, ..., f_m] = \theta}{x(e_1, ..., e_n) \sim y(f_1, ..., f_m) = \theta}\ \text{(PRED)}$$

- 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

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

BY NC ND

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

$$\frac{x = y \qquad n = m \qquad [e_1, ..., e_n] \sim [f_1, ..., f_m] = \theta}{x(e_1, ..., e_n) \sim y(f_1, ..., f_m) = \theta} \text{(PRED)}$$

- 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

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

cc BY NC ND