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

- 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

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

- 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

- 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

```
% 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
```

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

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

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

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

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

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

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

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

- 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

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, $\{ \ a \mapsto int, c \mapsto bool \ \}(a \rightarrow b) = int \rightarrow b$
- **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

- 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**

- 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

- 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

- 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**

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**

- Considere a seguinte gramática simplificada para tipos:

$$\begin{array}{lcl} \tau & ::= & \mathsf{int} \\ & \mid & x \\ & \mid & \tau \to \tau \end{array}$$

Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar String
          | 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

Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$$\begin{array}{lcl} \tau & ::= & \text{int} \\ & | & x \\ & | & \tau \to \tau \end{array}$$

Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar String
          | 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

Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$$
\begin{array}{rcl}
\tau & ::= & \mathsf{int} \\
 & | & x \\
 & | & \tau \to \tau
\end{array}
$$

  Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar String
          | 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

Exemplo: sistema de tipos

- Considere a seguinte gramática simplificada para tipos:

$$\begin{array}{rcl} \tau & ::= & \text{int} \\ & | & x \\ & | & \tau \rightarrow \tau \end{array}$$

Ou, em Haskell:

```haskell
data Type = TypeInt
          | TypeVar String
          | 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

- 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

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

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

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

- 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

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

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

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

- 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

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

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

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

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

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

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

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