



Sistemas de Tipos

Rafael Castro

rafaelcgs10@gmail.com

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

10 de Julho de 2019

Plano de Aula - Sistema de Tipos

Objetivos

Explicar o que são e para que servem sistemas de tipos na Ciência da Computação.

Conteúdos

- Breve visão histórica.
- Usos informais em linguagens de programação.
- Sistemas de tipos formais em linguagens de programação.
- Outras aplicações de sistemas de tipos.

Método

Exposição do conteúdo com exemplos, questionamentos e exercícios.

Visão histórica - Russel

Teoria dos Tipos

- Bertrand Russell
- Teorias de Tipos são uma resposta ao paradoxo de Russel presente na *naive set theory*.
- Uma teoria alternativa a Teoria dos Conjuntos para uma função formal da matemática.



Visão histórica - Church

Teoria dos Tipos

- Alonzo Church
- Aplicou a teoria de tipos de Russel no modelo de computação chamado Cálculo Lambda, a fim de evitar paradoxos e computações sem fim.
- Essa junção entre Teoria de Tipos e Cálculo Lambda é o fundamento teórico para sistemas de tipos em linguagens de programação.



Questões preliminares

O que são tipos (em linguagens de programação)?

Questões preliminares

O que são tipos (em linguagens de programação)?

- Classificações dos dados de um programa: carácter, inteiro, função etc.

Questões preliminares

O que são tipos (em linguagens de programação)?

- Classificações dos dados de um programa: carácter, inteiro, função etc.

Por que (algumas) linguagens de programação utilizam tipos?



Questões preliminares

O que são tipos (em linguagens de programação)?

- Classificações dos dados de um programa: carácter, inteiro, função etc.

Por que (algumas) linguagens de programação utilizam tipos?

```
void foo() {  
    int n = 1;  
    char m = '2';  
    printf("%d\n", n + m);  
}
```

```
void spam() {  
    int n;  
    scanf("%d", &n);  
    printf("%d\n", n);  
    return 0;  
}
```



Questões preliminares

O que são tipos (em linguagens de programação)?

- Classificações dos dados de um programa: carácter, inteiro, função etc.

Por que (algumas) linguagens de programação utilizam tipos?

```
void foo() {  
    int n = 1;  
    char m = '2';  
    printf("%d\n", n + m);  
}
```

```
void spam() {  
    int n;  
    scanf("%d", &n);  
    printf("%d\n", n);  
    return 0;  
}
```

Os tipos fazem parte da construção semântica dos programas.

Tipagem estática vs dinâmica

- Na tipagem estática a verificação dos tipos é feita em por análise do código durante a compilação.
- Na tipagem dinâmica as verificações dos tipos é feita somente na execução.
Se uma parte do código com erro (de tipo) não for executada, o erro ficará oculto.

Tipagem estática vs dinâmica

- Na tipagem estática a verificação dos tipos é feita em por análise do código durante a compilação.
- Na tipagem dinâmica as verificações dos tipos é feita somente na execução.
Se uma parte do código com erro (de tipo) não for executada, o erro ficará oculto.
- Estática: atribuições de tipo não mudam em *run time*.
- Dinâmica: atribuições de tipo **podem mudar** em *run time*.

Tipagem estática vs dinâmica

- Na tipagem estática a verificação dos tipos é feita em por análise do código durante a compilação.
- Na tipagem dinâmica as verificações dos tipos é feita somente na execução.
Se uma parte do código com erro (de tipo) não for executada, o erro ficará oculto.
- Estática: atribuições de tipo não mudam em *run time*.
- Dinâmica: atribuições de tipo **podem mudar** em *run time*.

Exemplos de linguagens de tipagem estática?

Tipagem estática vs dinâmica

- Na tipagem estática a verificação dos tipos é feita em por análise do código durante a compilação.
- Na tipagem dinâmica as verificações dos tipos é feita somente na execução.
Se uma parte do código com erro (de tipo) não for executada, o erro ficará oculto.
- Estática: atribuições de tipo não mudam em *run time*.
- Dinâmica: atribuições de tipo **podem mudar** em *run time*.

Exemplos de linguagens de tipagem estática?

C, C++, Java, Rust, Haskell, OCaml...

Tipagem estática vs dinâmica

- Na tipagem estática a verificação dos tipos é feita em por análise do código durante a compilação.
 - Na tipagem dinâmica as verificações dos tipos é feita somente na execução.
Se uma parte do código com erro (de tipo) não for executada, o erro ficará oculto.
- Estática: atribuições de tipo não mudam em *run time*.
 - Dinâmica: atribuições de tipo **podem mudar** em *run time*.

Exemplos de linguagens de tipagem estática?

C, C++, Java, Rust, Haskell, OCaml...

Exemplos de linguagens de tipagem dinâmica?

Tipagem estática vs dinâmica

- Na tipagem estática a verificação dos tipos é feita em por análise do código durante a compilação.
 - Na tipagem dinâmica as verificações dos tipos é feita somente na execução.
Se uma parte do código com erro (de tipo) não for executada, o erro ficará oculto.
- Estática: atribuições de tipo não mudam em *run time*.
 - Dinâmica: atribuições de tipo **podem mudar** em *run time*.

Exemplos de linguagens de tipagem estática?

C, C++, Java, Rust, Haskell, OCaml...

Exemplos de linguagens de tipagem dinâmica?

Python, PHP, JavaScript, Lisp, Erlang...



Vantagens e desvantagens: estática e dinâmica

Estática:

- Vantagens: Reduz os possíveis erros em tempo de execução; Possibilita algumas otimizações de código pelo compilador; Verificação automática; Útil em códigos grandes.
- Desvantagens: Mais uma coisa para aprender sobre a linguagem; Requerem que tipos sejam explicitados (não todas as linguagens); Podem limitar o reuso de código.

Dinâmica:

- Vantagens: Mais permissível; Mais fácil de aprender; geralmente tem mais interatividade (REPL).
- Desvantagens: Testes são muito mais importantes; abre espaço para erros inesperados em execução; geralmente são menos eficientes.

Tipagem forte e fraca

Esse é um conceito meio nebuloso!

Não é preto ou branco

- Forte: as regras de tipagem são rígidas; não há cast de tipos implícitos; os casts de tipos fazem sentido.
- Fraca: as regras de tipagem são flexíveis; casts automáticos podem ocorrer.



Tipagem forte e fraca

Esse é um conceito meio nebuloso!

Não é preto ou branco

- Forte: as regras de tipagem são rígidas; não há cast de tipos implícitos; os casts de tipos fazem sentido.
- Fraca: as regras de tipagem são flexíveis; casts automáticos podem ocorrer.

Em Python:

```
"Dia " + str(1) + " de Janeiro"
```

Tipagem forte e fraca

Esse é um conceito meio nebuloso!

Não é preto ou branco

- Forte: as regras de tipagem são rígidas; não há cast de tipos implícitos; os casts de tipos fazem sentido.
- Fraca: as regras de tipagem são flexíveis; casts automáticos podem ocorrer.

Em Python:

```
"Dia " + str(1) + " de Janeiro"
```

Em JavaScript:

```
[] + [] ; // ""  
[] + 1; // "1"  
1 + {} ; // "1[object object]"  
{ } + [] // 0
```

Introdução - Sistemas de Tipos

- Sistemas de tipos são regras que ditam quais expressões podem ser associadas a quais tipos.
- Nas linguagens de programação, classificavam os dados para serem tratados corretamente pelo processador;
- passaram a ser utilizados como regras para identificar falhas na consistência de programas
- Tipagem estática e dinâmica:
 - Estática: atribuições de tipo não mudam em *run time*; uma forma de verificação automática.
 - Dinâmica: atribuições de tipo **mudam** em *run time*; mais permissível; mais permissível a erros.

[1, 'a', objeto, [2]]



Introdução - Inferência de Tipos e Polimorfismo

- A inferência de tipos é o processo de encontrar a assinatura de tipo mais geral de um programa.
- Liberdade ao programador de escolher anotar ou não os tipos das funções.
- Polimorfismo é quando uma função pode assumir vários usos com diferentes tipos de dados. Em especial, o polimorfismo paramétrico é feito pela quantificação de variáveis de tipos.

`len :: [a] -> Int`

- Existem situações onde a inferência de tipos é mais difícil que a verificação. Nesses casos, algoritmos podem rejeitar programas bem tipados.

Fundamentos - Formalismos de Sistemas de Tipos

- Sistemas de tipos quando definidos formalmente podem garantir importantes propriedades da semântica de programas, como o famoso lema de Robin Milner “Well-typed programs cannot go wrong”.
- Damas-Milner: polimorfismo via *let*. Base para diversas linguagens de programação funcional. Não permite recursão polimórfica pela regra *fix*.
- O primeiro sistema de tipos a permitir recursão polimórfica é o Sistema-F.
- Milner-Mycroft altera a regra de definições recursivas *fix* para permitir recursões polimórficas.

$$\frac{\Gamma, \mathbf{x} : \tau \vdash \mathbf{e} : \tau}{\Gamma \vdash \mathbf{fix}\ \mathbf{x}.\mathbf{e} : \tau} (\textit{fix}) \quad \frac{\Gamma, \mathbf{x} : \sigma \vdash \mathbf{e} : \sigma}{\Gamma \vdash \mathbf{fix}\ \mathbf{x}.\mathbf{e} : \sigma} (\textit{fix+})$$



Conclusão

- Fundamentos teóricos para formalização do MMo.
- Assistentes de provas Coq.
- Como o MMo funciona.
- Formalização do MMo parcial.
- Principais dificuldades encontradas.
- Alternativas para a prova de terminação.