Linguagens de Programação

Sistemas de Tipos

Samuel da Silva Feitosa

Aula 20

Sistemas de Tipos

Introdução

- Na última aula vimos como descrever precisamente a semântica de uma linguagem de expressões.
  - Também implementamos um pequeno interpretador, que avalia uma dada expressão e produz um novo resultado.
- Entretanto, apenas com o que vimos até agora não é possível garantir que uma expressão será avaliada por completo.
  - Expressões podem “travar”, ou chegar em algum estado que não podem mais ser reduzidas.

Sistema de Tipos - Motivação

- Vejamos um exemplo:
  - Se em algum momento, chegarmos a uma expressão como *true* + *2*.
  - Qual é o comportamento esperado para isso?
  - Não é possível somar um valor *booleano* com um número.
- Geralmente, em situações como esta, o programa é considerado errado.
- Vamos estudar o conceito de Sistema de Tipos, também chamado de Semântica Estática.
  - Verifica se um termo foi corretamente definido.
  - Diferencia termos bem tipados de termos mal formados.

Sistema de Tipos

- Nossa linguagem de expressões possui apenas dois tipos válidos.
  - *Bool* para representar booleanos.
  - *Num* para representar valores numéricos.
- Definimos *T* como sendo uma meta-variável que permite representar ambos os tipos válidos da linguagem.

$T ::=$	types:
$Bool$	type of booleans
Num	type of numbers

Regras do Sistema de Tipos

- Para definir o sistema de tipos, usamos uma regra da forma $\vdash e : T$, que significa que “uma expressão *e* tem tipo *T*”.
- A Figura abaixo apresenta as regras do sistema de tipos da nossa linguagem de expressões.

$$\vdash n : Num \quad \text{[T-Num]} \qquad \vdash true : Bool \quad \text{[T-True]} \qquad \vdash false : Bool \quad \text{[T-False]}$$

$$\frac{\vdash e_1 : Num \quad \vdash e_2 : Num}{\vdash e_1 + e_2 : Num} \quad \text{[T-Add]} \qquad \frac{\vdash e_1 : Bool \quad \vdash e_2 : Bool}{\vdash e_1 \wedge e_2 : Bool} \quad \text{[T-And]}$$

Detalhamento das Regras

- T-Num: atribui o tipo *Num* para constantes numéricas.
- T-True e T-False: Atribui o tipo *Bool* para as constantes booleanas *true* e *false*.
- T-Add: atribui o tipo *Num* para o resultado da operação, considerando que ambos e_1 e e_2 são do tipo *Num*.
- T-And: similar a T-Add, que atribui o tipo *Bool* como resultado da operação, desde que e_1 e e_2 sejam do tipo *Bool*.
- **Formalmente:** uma expressão *e* é dita bem-tipada se existe algum *T*, tal que $\vdash e : T$.

Provas de Propriedades

- Tendo definido a semântica estática (sistema de tipos) e a semântica dinâmica (regras de redução), é possível provar propriedades matemáticas a respeito da linguagem.
  - Por exemplo: é possível provar que um termo bem tipado não “trava”, e que um termo bem tipado sempre pode ser avaliado, até se tornar um valor.
  - Estas propriedades são conhecidas como *type safety* ou *soundness*.
  - Principais teoremas: *Progress* e *Preservation*.
- Não faremos as provas nesta disciplina, mas vocês podem procurar informações extras a respeito.

Implementação do Sistema de Tipos

- A partir das regras de tipo, podemos implementar uma função em Haskell que retorna o tipo de uma expressão.

```haskell
-- Função que verifica o tipo de uma expressão
typeof :: Expr -> Maybe Ty
typeof BTrue = Just TBool
typeof BFalse = Just TBool
typeof (Num _) = Just TNum
typeof (Add e1 e2) = case (typeof e1) of
                       Just TNum -> case (typeof e2) of
                                      Just TNum -> Just TNum
                                      _         -> Nothing
                       _         -> Nothing
typeof (And e1 e2) = case (typeof e1, typeof e2) of
                       (Just TBool, Just TBool) -> Just TBool
                       _                        -> Nothing
```

Considerações Finais

- Nesta aula estudamos o conceito de Sistema de Tipos de linguagens de programação.
- Definimos as regras de tipo da LP.
  - Utilizamos a mesma ideia de regras de inferência vistos na aula passada.
- Implementamos um verificador de tipos.
  - Função simples em Haskell que percorre as expressões e verifica se elas estão bem-tipadas.

Exercícios

1. Adicionar os construtores e adaptar as funções ***typeof*** e ***step*** para realizar as operações aritméticas de **subtração** e **multiplicação**.
2. Adicionar os construtores e adaptar as funções ***typeof*** e ***step*** para realizar as operações lógicas **OR** e **XOR**.
3. Implementar uma função recursiva chamada ***eval*** para avaliar uma expressão **até que ele se torne um valor final**.