

Avaliador e Inferência de Tipos para L1 com Listas e Exceções

A linguagem da gramática abstrata abaixo é L1 aumentada com exceções e listas. A linguagem de tipos também é aumentada com tipos para listas e **variáveis de tipo**, representadas por letras maiúsculas do final do alfabeto.

$$\begin{array}{lcl} e & \in & \text{Expr} \\ e & ::= & n \\ & | & b \\ & | & e_1 \text{ op } e_2 \\ & | & \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \\ & | & x \\ & | & e_1 e_2 \\ & | & \text{fn } x:T \Rightarrow e \\ & | & \text{let } x:T = e_1 \text{ in } e_2 \\ & | & \text{let rec } f:T_1 \rightarrow T_2 = (\text{fn } y:T_1 \Rightarrow e_1) \text{ in } e_2 \\ & | & \text{nil} \\ & | & e_1 :: e_2 \\ & | & \text{isempty } e \\ & | & \text{hd } e \\ & | & \text{tl } e \\ & | & \text{raise} \\ & | & \text{try } e_1 \text{ with } e_2 \end{array}$$
$$T \in \text{Types}$$
$$T ::= X \mid \text{int} \mid \text{bool} \mid T_1 \rightarrow T_2 \mid T \text{ list}$$

onde

$$\begin{array}{lcl} n & \in & \text{conjunto de numerais inteiros} \\ b & \in & \{\text{true}, \text{false}\} \\ x & \in & \text{Ident} \\ \text{op} & \in & \{+, -, *, \text{div}, ==, \text{and}, \text{or}, \text{not}\} \end{array}$$

Inferência de tipos para linguagem explicitamente tipada

O algoritmo **typeInfer** de inferência de tipos para a linguagem acima (sem polimorfismo) consiste basicamente das seguintes etapas principais:

1. coleta de equações de tipo que representam restrições que devem ser respeitadas por tipos (*type constraints*)
2. resolução das equações de tipo coletadas (*constraint solving*)

A etapa (1) acima de coleta das equações de tipo (os *type constraints*) nunca falha, ou seja sempre vai terminar e retornar um conjunto de *type constraints*, e também um tipo (possivelmente com variáveis de tipo). Se, na etapa (2), não houver como *resolver* esse conjunto de equações de tipos isso significa que o programa submetido para **typeInfer** é mal tipado.

Se houver como *resolver* esse conjunto de equações de tipo, o resultado da etapa (2) é uma substituição de variáveis de tipo por tipos que torna todas as equações de tipo coletadas verdadeiras. Essa substituição então é aplicada ao tipo produzido na etapa (1) da coleta resultando no tipo final do programa submetido para **typeinfer**.

```

typeinfer( $\Gamma, P$ ) =
  let
    ( $T, C$ ) = collectTyEqs( $\Gamma, P$ )  (* nunca falha *)
     $\sigma$  = Unify( $C$ ) (* resolve as equações em  $C$  – pode ativar exceção *)
  in
    applysubs( $\sigma, T$ ) (* produz tipo final do programa  $P$  *)
  end

```

A função collectTyEqs do algoritmo acima é implementada seguindo as regra de coleta das equações de tipo abaixo. Note que premissas e conclusão das regras abaixo tem o formato:

$$\Gamma \vdash e : T \mid C$$

onde a barra vertical é só um separador do tipo T do conjunto de constraints coletados C :

$$\Gamma \vdash n : \text{int} \mid \{\} \quad (\text{C-NUM})$$

$$\Gamma \vdash b : \text{bool} \mid \{\} \quad (\text{C-BOOL})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 + e_2 : \text{int} \mid C_1 \cup C_2 \cup \{T_1 = \text{int}, T_2 = \text{int}\}} \quad (\text{C-SUM})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 = e_2 : \text{bool} \mid C_1 \cup C_2 \cup \{T_1 = \text{int}, T_2 = \text{int}\}} \quad (\text{C-EQ})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2 \quad \Gamma \vdash e_3 : T_3 \mid C_3}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T_2 \mid C_1 \cup C_2 \cup C_3 \cup \{T_1 = \text{bool}, T_2 = T_3\}} \quad (\text{C-IF})$$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T \mid \{\}} \quad (\text{C-ID})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2 \quad X \text{ is new}}{\Gamma \vdash e_1 \ e_2 : X \mid C_1 \cup C_2 \cup \{T_1 = T_2 \rightarrow X\}} \quad (\text{C-APP})$$

$$\frac{\Gamma, x : T \vdash e : T_1 \mid C}{\Gamma \vdash \text{fn } x : T \Rightarrow e : T \rightarrow T_1 \mid C} \quad (\text{C-FN})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma, x : T \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \text{let } x : T = e_1 \text{ in } e_2 : T_2 \mid C_1 \cup C_2 \cup \{T = T_1\}} \quad (\text{C-LET})$$

$$\frac{\Gamma, f : T' \rightarrow T, y : T' \vdash e_1 : T_1 \mid C_1 \quad \Gamma, f : T' \rightarrow T \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \text{let rec } f : T' \rightarrow T = (\text{fn } y : T' \Rightarrow e_1) \text{ in } e_2 : T_2 \mid C_1 \cup C_2 \cup \{T = T_1\}} \quad (\text{C-LETR})$$

$$\begin{array}{c}
\frac{X \text{ is new}}{\Gamma \vdash \text{nil} : X \text{ list} \mid \{\}} \quad (\text{C-NIL}) \\
\\
\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 :: e_2 : T_2 \mid C_1 \cup C_2 \cup \{T_1 \text{ list} = T_2\}} \quad (\text{C-CONS}) \\
\\
\frac{\Gamma \vdash e : T \mid C \quad X \text{ is new}}{\Gamma \vdash \text{hd } e : X \mid C \cup \{T = X \text{ list}\}} \quad (\text{C-HD}) \\
\\
\frac{\Gamma \vdash e : T \mid C \quad X \text{ is new}}{\Gamma \vdash \text{tl } e : X \text{ list} \mid C \cup \{T = X \text{ list}\}} \quad (\text{C-TL}) \\
\\
\frac{\Gamma \vdash e : T \mid C \quad X \text{ is new}}{\Gamma \vdash \text{isempty } e : \text{bool} \mid C \cup \{T = X \text{ list}\}} \quad (\text{C-ISEMPTY}) \\
\\
\frac{X \text{ is new}}{\Gamma \vdash \text{raise} : X \mid \{\}} \quad (\text{C-RAISE}) \\
\\
\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \text{try } e_1 \text{ with } e_2 : T_2 \mid C_1 \cup C_2 \cup \{T_1 = T_2\}} \quad (\text{C-TRY})
\end{array}$$

As regras acima podem ser usadas diretamente como uma especificação de uma rotina de coleta de equações de tipo. Uma regra como por exemplo:

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \text{try } e_1 \text{ with } e_2 : T_2 \mid C_1 \cup C_2 \cup \{T_1 = T_2\}}$$

pode ser vista da seguinte maneira seguinte

$$\frac{\text{collectTyEqs}(\Gamma, e_1) = (T_1, C_1) \quad \text{collectTyEqs}(\Gamma, e_2) = (T_2, C_2)}{\text{collectTyEqs}(\Gamma, \text{try } e_1 \text{ with } e_2) = (T_2, C_1 \cup C_2 \cup \{T_1 = T_2\})}$$

A produção de um par (T, C) onde T é um tipo (que pode ter ou não variáveis de tipo), e C é um conjunto de equações de tipo, é apenas a primeira etapa do processo de inferência de tipo. A etapa (2) consiste em tentar **resolver** o conjunto C de equações de tipo.

Isso é feito pela função **Unify** que recebe como argumento um conjunto de equações de tipo e falha, caso o conjunto de equações não tenha solução, ou retorna uma **substituição** σ que consiste de um mapeamento de variáveis de tipos para tipos. Se **Unify** retornar uma substituição de tipos σ isso significa que o conjunto de equações tem solução.

O algoritmo `typeInfer` então aplica essa substituição ao tipo T retornado por `collectTyEqs`, produzindo assim o tipo final da expressão submetida.

Um algoritmo para `Unify` encontra-se na Figura 22.2 do capítulo 2 do livro *Types and programming language*, de Benjamin Pierce (ver referências e link para o livro no programa da disciplina).

Inferência de tipos com tipagem implícita

Nessa versão a sintaxe da linguagem é modificada de tal forma que o programador não precisa informar o tipo de identificadores em funções e em expressões **let** e **let rec**. Na gramática, as informações de tipo passam a ser opcionais. O que equivale a **adicionar** as seguintes cláusulas na gramática abstrata linguagem (ou seja as versões explicitamente tipadas permanecem, mas os tipos são opcionais):

$$\begin{array}{lcl} e & ::= & \dots \\ & | & \text{fn } x \Rightarrow e \\ & | & \text{let } x = e_1 \text{ in } e_2 \\ & | & \text{let rec } f = (\text{fn } y \Rightarrow e_1) \text{ in } e_2 \end{array}$$

As regras para coleta de *type constraints* para essas versões sem tipos explícitos são as seguintes:

$$\frac{\Gamma, x : X \vdash e : T \mid C \quad X \text{ is new}}{\Gamma \vdash \text{fn } x \Rightarrow e : X \rightarrow T \mid C}$$
$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \quad \Gamma, x : X \vdash e_2 : T_2 \mid C_2 \quad X \text{ is new}}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : T_2 \mid C_1 \cup C_2 \cup \{X = T_1\}}$$
$$\frac{\Gamma, f : X, y : Y \vdash e_1 : T_1 \mid C_1 \quad \Gamma, f : X \vdash e_2 : T_2 \mid C_2 \quad X, Y \text{ are new}}{\Gamma \vdash \text{let rec } f = (\text{fn } y \Rightarrow e_1) \text{ in } e_2 : T_2 \mid C_1 \cup C_2 \cup \{X = Y \rightarrow T_1\}}$$

Trabalho da disciplina 2018/1

O trabalho consiste em

1. Definir a semântica operacional *big step* da linguagem L1 com listas e exceções
2. Implementar um avaliador de programas L1 com listas e exceções de acordo com essa semântica *big-step*
3. Implementar o algoritmo de inferência de tipos para L1 com listas e exceções com tipagem explícita e implícita

O algoritmo para solução de conjuntos de *constraints* com equações de tipos está na resposta do exercício 22.4.6 do livro *Types and Programming Languages* de Benjamin Pierce.

Prazos e avaliação

O trabalho deverá ser submetido pelo Moodle **até o dia 13 de junho**.

A nota final no trabalho será baseada no material submetido e também na apresentação a ser feita para o professor.