## 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.

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

onde

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

### 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 contraints*)

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**.

$$
\begin{array}{l}
\texttt{typeinfer}(\Gamma, P) \;=\\
\quad \texttt{let}\\
\quad\quad (T, C) = \texttt{collectTyEqs}(\Gamma, P) \quad (*\ nunca falha\ *)\\
\quad\quad \sigma = \texttt{Unify}(C)\ (*\ resolve\ as\ equações\ em C\ -\ pode\ ativar\ exceção\ *)\\
\quad \texttt{in}\\
\quad\quad \texttt{applysubs}(\sigma, T)\ (*\ produz\ tipo\ final\ do\ programa\ P *)\\
\quad \texttt{end}
\end{array}
$$

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 $C$:

$$\Gamma \vdash n : \mathsf{int} \mid \{\} \tag{C-Num}$$

$$\Gamma \vdash b : \mathsf{bool} \mid \{\} \tag{C-Bool}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 + e_2 : \mathsf{int} \mid C_1 \cup C_2 \cup \{T_1 = \mathsf{int}, T_2 = \mathsf{int}\}} \tag{C-Sum}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 = e_2 : \mathsf{bool} \mid C_1 \cup C_2 \cup \{T_1 = \mathsf{int}, T_2 = \mathsf{int}\}} \tag{C-Eq}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2 \qquad \Gamma \vdash e_3 : T_3 \mid C_3}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T_2 \mid C_1 \cup C_2 \cup C_3 \cup \{T_1 = \mathsf{bool}, T_2 = T_3\}} \tag{C-If}$$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T \mid \{\}} \tag{C-Id}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2 \qquad X\ is\ new}{\Gamma \vdash e_1\ e_2 : X \mid C_1 \cup C_2 \cup \{T_1 = T_2 \to X\}} \tag{C-App}$$

$$\frac{\Gamma, x : T \vdash e : T_1 \mid C}{\Gamma \vdash \texttt{fn}\ x{:}T \Rightarrow e : T \to T_1 \mid C} \tag{C-Fn}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma, x : T \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2 : T_2 \mid C_1 \cup C_2 \cup \{T = T_1\}} \tag{C-Let}$$

$$\frac{\Gamma, f : T' \to T, y : T' \vdash e_1 : T_1 \mid C_1 \qquad \Gamma, f : T' \to T \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \texttt{let rec}\ f{:}T' \to T = (\texttt{fn}\ y{:}T' \Rightarrow e_1)\ \texttt{in}\ e_2 : T_2 \mid C_1 \cup C_2 \cup \{T = T_1\}} \tag{C-LetR}$$

$$\frac{X \textit{ is new}}{\Gamma \vdash \texttt{nil} : X \text{ list} \mid \{\}} \quad (\text{C-NIL})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 :: e_2 : T_2 \mid C_1 \cup C_2 \cup \{T_1 \text{ list } = T_2\}} \quad (\text{C-CONS})$$

$$\frac{\Gamma \vdash e : T \mid C \qquad X \textit{ is new}}{\Gamma \vdash \texttt{hd}\ e : X \mid C \cup \{T = X \text{ list}\}} \quad (\text{C-HD})$$

$$\frac{\Gamma \vdash e : T \mid C \qquad X \textit{ is new}}{\Gamma \vdash \texttt{tl}\ e : X \text{ list} \mid C \cup \{T = X \text{ list}\}} \quad (\text{C-TL})$$

$$\frac{\Gamma \vdash e : T \mid C \qquad X \textit{ is new}}{\Gamma \vdash \texttt{isempty}\ e : \text{bool} \mid C \cup \{T = X \text{ list}\}} \quad (\text{C-ISEMPTY})$$

$$\frac{X \textit{ is new}}{\Gamma \vdash \texttt{raise} : X \mid \{\}} \quad (\text{C-RAISE})$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \texttt{try}\ e_1\ \texttt{with}\ e_2 : T_2 \mid C_1 \cup C_2 \cup \{T_1 = T_2\}} \quad (\text{C-TRY})$$

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

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

pode ser vista da seguinte maneira seguinte

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

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 **sustituição** $\sigma$ que consiste de um mapeamento de variáveis de tipos para tipos. Se **Unify** retornar uma substituição de tipos $\sigma$ 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 Benjamim Pierce (ver referências e link para o livro no programa da disciplina).

### Inferência de tipos com tipagem implicíta

Nessa versão a sintaxe da linguagem é modificada de tal forma que o programador não precisa informar o tipo de identicadores 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):

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

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

$$\frac{\Gamma,\ x : X \vdash e : T \mid C \qquad X\ \textit{is new}}{\Gamma \vdash \texttt{fn}\ x \Rightarrow\ e : X \to T \mid C}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma, x : X \vdash e_2 : T_2 \mid C_2 \qquad X\ \textit{is new}}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ \ e_2 : T_2 \mid C_1 \cup C_2 \cup \{X = T_1\}}$$

$$\frac{\Gamma,\ f : X,\ y : Y \vdash e_1 : T_1 \mid C_1 \qquad \Gamma, f : X \vdash e_2 : T_2 \mid C_2 \qquad X, Y\ \textit{are new}}{\Gamma \vdash \texttt{let rec}\ f = (\texttt{fn}\ y \Rightarrow e_1)\ \texttt{in}\ \ e_2 : T_2 \mid C_1 \cup C_2 \cup \{X = Y \to T_1\}}$$

### Trabalho da disciplina 2018/1

O trabalho consite 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.

**Encontrando solução para *type constraints***

O algoritmo para busca de solução para um conjunto de equações de tipo é definido abaixo na forma de um sistema de transição de estados como segue:

- os estados são pares $(\sigma, C)$ onde $\sigma$ é uma substituição de tipos e $C$ é um conjunto de equações de tipos
- Tanto a substituição de tipos $\sigma$ como o conjunto $C$ de equações de tipos serão tratados como listas
- o estado inicial é o par $([], C)$. Ou seja nenhuma substituição foi produzida e $C$ possui todas as equações de tipos coletadas
- O estado final é da forma $(\sigma, [])$. Esse estado final é alcançado quando todas as equações de tipo presentes no estado inicial foram consideradas. A substituição $\sigma$ é uma solução (a mais geral) para todas as equaações de tipo do estado inicial
- Um estado $(\sigma, C)$ é considerado um estado de erro, quando $C \neq []$ e não existe $(\sigma', C')$ tal que $(\sigma, C) \to (\sigma', C')$

Se um estado de erro é atingido isso significa que não há solução para as equações de tipo $C$ presentes no estado incial. Abaixo seguem as regras de transição:

1. $\sigma,\ (\text{int} = \text{int}) :: C \quad \longrightarrow \quad \sigma, C$
2. $\sigma,\ (\text{bool} = \text{bool}) :: C \quad \longrightarrow \quad \sigma,\ C$
3. $\sigma,\ (X = X) :: C \quad \longrightarrow \quad \sigma,\ C$

4. $\sigma,\ (X = T) :: C \quad \longrightarrow \quad \sigma@[(X, T)],\ \{T/X\}C \qquad$ se $X$ não ocorre em $T$
5. $\sigma,\ (T = X) :: C \quad \longrightarrow \quad \sigma@[(X, T)],\ \{T/X\}C \qquad$ se $X$ não ocorre em $T$

6. $\sigma,\ (T_1 \to T_2 = T_3 \to T_4) :: C \quad \longrightarrow \quad \sigma,\ (T_1 = T_3) :: (T_2 = T_4) :: C$
7. $\sigma,\ (T_1\ \text{list} = T_2\ \text{list}) :: C \quad \longrightarrow \quad \sigma,\ (T_1 = T_2) :: C$

A condição para o corrência das transições de número 4 e 5 acima é conhecida como *occur check*. Esse teste é importante pois uma equação que viola essa condição, como por exemplo $X = \text{int} \to X$ claramente não possui solução.