# Trabalho 2018-2
# Interpretador para L1 implicitamente tipada com Listas e Exceções

O trabalho da disciplina consite em

1. (3 pts) Definir a semântica operacional *big step* da linguagem L1 com listas e exceções e implementar um avaliador de programas L1 com listas e exceções de acordo com essa semântica *big-step*
2. (7 pts) Implementar o algoritmo de inferência de tipos para L1 com listas e exceções

O trabalho deve ser feito usando a linguagem OCAM em grupos de 3, e deverá ser submetido pelo Moodle **até o dia 16 de novembro**. A nota final no trabalho será baseada no material submetido e também na apresentação a ser feita em aula perante a turma.

## Sintaxe de L1

A linguagem da gramática abstrata abaixo é L1 implicitamente tipada aumentada com exceções e listas.

$$
\begin{array}{rcl}
e & \in & \mathsf{Expr} \\
e & ::= & n \\
& | & b \\
& | & e_1 \ \mathit{bop}\ e_2 \\
& | & \mathit{uop}\ e \\
& | & \texttt{(}e_1\texttt{,}e_2\texttt{)} \\
& | & \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \\
& | & x \\
& | & e_1\ e_2 \\
& | & \texttt{fn}\ x \Rightarrow\ e \\
& | & \texttt{let}\ x = e_1\ \texttt{in}\ \ e_2 \\
& | & \texttt{let rec}\ f = (\texttt{fn}\ y \Rightarrow e_1)\ \texttt{in}\ \ e_2 \\
& | & \texttt{nil} \\
& | & e_1\texttt{::}\ e_2 \\
& | & \texttt{isempty}\ e \\
& | & \texttt{hd}\ e \\
& | & \texttt{tl}\ e \\
& | & \texttt{raise} \\
& | & \texttt{try}\ e_1\ \texttt{with}\ e_2
\end{array}
$$

onde

$$
\begin{array}{lcl}
n & \in & \textit{conjunto de numerais inteiros} \\
b & \in & \{\texttt{true}, \texttt{false}\} \\
x & \in & \mathsf{Ident} \\
\mathit{bop} & \in & \{+, -, *, /, =, <>, <, <=, >, >=, and, or\} \\
\mathit{uop} & \in & \{\ \neg\ \}
\end{array}
$$

## Inferência de tipos para linguagem L1 implicitamente tipada

A linguagem de tipos é aumentada com tipos para listas e **variáveis de tipo**, representadas por letras maiúsculas do final do alfabeto. Observe que informações de tipo não estão presentes em programas escritos nessa versão de L1 implicitamente tipada. O tipo de expressões deverá ser inferido/reconstruido a partir de informações do contexto.

$$T \quad ::= \quad X \mid \mathsf{int} \mid \mathsf{bool} \mid T_1 \rightarrow T_2 \mid T\ \mathsf{list} \mid T_1 \times T_2$$

O tipo é inferido com base em restrições impostas pelo contexto. Considere o seguinte exemplo abaixo:

- a expressão $\mathtt{fn}\ f \Rightarrow f\ (f\ 3)$ é bem tipada do tipo $(\mathsf{int} \rightarrow \mathsf{int}) \rightarrow \mathsf{int}$
- a expressão $\mathtt{fn}\ f \Rightarrow f\ (f\ \mathtt{true})$ é bem tipada do tipo $(\mathsf{bool} \rightarrow \mathsf{bool}) \rightarrow \mathsf{bool}$
- a expressão $\mathtt{fn}\ f \Rightarrow f\ (f\ 3, f\ 4)$ é mal tipada. O contexto tem restrições de tipo conflitantes
- a expressão $\mathtt{fn}\ f \Rightarrow f\ 3$ tem tipo $(\mathsf{int} \rightarrow X) \rightarrow X$
- a expressão $\mathtt{fn}\ x \Rightarrow x$ tem tipo $X \rightarrow X$
- $\mathtt{let}\ id = \mathtt{fn}\ x \Rightarrow x\ \mathtt{in}\ (id\ 3, id\ \mathtt{true})$ é mal tipada. O contexto tem restrições de tipo conflitantes[1]

O algoritmo **typeInfer** de inferência de tipos para a linguagem L1 consiste basicamente das seguintes etapas principais:

1. coleta de equações de tipo que representam restrições que devem ser respeitadas por tipos
2. resolução das equações de tipo coletadas

A etapa (1) de coleta das equações de tipo só falha se o programa tiver algun identificador não declarado. Se isso não ocorrer essa etapa sempre termina e retorna: (i) um conjunto de equações de tipo, e (ii) um tipo (possivelmente com variáveis de tipo).

Há dois resultados possíveis na etapa (2):

- Se não houver como *resolver* o conjunto de equações de tipos coletados na etapa (i) isso significa que não é possivel satisfazer todas as restrições de tipo coletadas (os *type constraints*). O programa submetido para **typeInfer** é portanto, considerado 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.

Se a resolução de equações de tipo devolver uma substituição então ela é aplicada ao tipo produzido na etapa (1) resultando no tipo final do programa submetido para **typeInfer**.

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

[1]Se a linguagem L1 suportasse polimorfismo paramétrico essa expressão seria be tipada e do tipo int $\times$ bool

A função `collect` do algoritmo acima é implementada seguindo as regra de coleta das equações de tipo abaixo[2]. As premissas e conclusão das regras abaixo tem o formato:

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

onde $\Gamma$ é um ambiente de tipo mapeando variáveis declaradas para seus tipos; $e$ é uma árvore de sintaxe abstrata da linguagem de acordo com a gramárica acima para expressões; $T$ é um tipo de acordo com a gramática para tipos dada acima, e $C$ é um conjunto de equações de tipo.

$$\frac{}{\Gamma \vdash n : \mathsf{int} \mid \{\}} \qquad \frac{}{\Gamma \vdash b : \mathsf{bool} \mid \{\}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 \ \texttt{+}\ e_2 : \mathsf{int} \mid C_1 \cup C_2 \cup \{T_1 = \mathsf{int}, T_2 = \mathsf{int}\}} \qquad \frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 \ \texttt{and}\ e_2 : \mathsf{bool} \mid C_1 \cup C_2 \cup \{T_1 = \mathsf{bool}, T_2 = \mathsf{bool}\}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 \ \texttt{=}\ e_2 : \mathsf{bool} \mid C_1 \cup C_2 \cup \{T_1 = \mathsf{int}, T_2 = \mathsf{int}\}} \qquad \frac{\Gamma \vdash e : T \mid C}{\Gamma \vdash \texttt{not}\ e : \mathsf{bool} \mid C \cup \{T = \mathsf{bool}\}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash \texttt{(}e_1\texttt{,}e_2\texttt{)} : T_1 \times T_2 \mid C_1 \cup C_2}$$

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

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T \mid \{\}} \qquad \frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2 \qquad X\ \mathit{new}}{\Gamma \vdash e_1\ e_2 : X \mid C_1 \cup C_2 \cup \{T_1 = T_2 \to X\}}$$

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

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

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

$$\frac{X\ \mathit{is\ new}}{\Gamma \vdash \texttt{nil} : X\ \mathsf{list} \mid \{\}} \qquad \frac{\Gamma \vdash e : T \mid C \qquad X\ \mathit{is\ new}}{\Gamma \vdash \texttt{tl}\ e : X\ \mathsf{list} \mid C \cup \{T = X\ \mathsf{list}\}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2}{\Gamma \vdash e_1 :: e_2 : T_2 \mid C_1 \cup C_2 \cup \{T_1\ \mathsf{list}\ = T_2\}} \qquad \frac{\Gamma \vdash e : T \mid C \qquad X\ \mathit{is\ new}}{\Gamma \vdash \texttt{isempty}\ e : \mathsf{bool} \mid C \cup \{T = X\ \mathsf{list}\}}$$

$$\frac{\Gamma \vdash e : T \mid C \qquad X\ \mathit{is\ new}}{\Gamma \vdash \texttt{hd}\ e : X \mid C \cup \{T = X\ \mathsf{list}\}} \qquad \frac{X\ \mathit{is\ new}}{\Gamma \vdash \texttt{raise} : X \mid \{\}}$$

[2] Para simplificar vamos considerar que a igualdade pode ser usada somente com inteiros.

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

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

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

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.

A solução de um conjunto de equações de tipo é feita pela função **Unify** que recebe como argumento um conjunto de equações de tipo e (i) falha, caso o conjunto de equações de tipo não tenha solução, ou (ii) retorna uma **sustituição de tipo** $\sigma$ 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 e que o programa submetido a **typeInfer** é bem tipado

O algoritmo **typeInfer** então aplica essa substituição $\sigma$ ao tipo $T$ retornado por collect, produzindo assim o tipo final da programa.

**Resolvendo equações de tipo com Unify**

O algoritmo de Unificação **Unify** para busca de solução para um conjunto de equações de tipo (construçao de uma substituição de tipos) é definido abaixo na forma de um sistema de transição de estados como segue:

- os estados do sistema e transição de estados são pares $(\sigma, C)$ onde $\sigma$ é uma substituição de tipos e $C$ é um conjunto de equações de tipos
- uma substituição de tipos $\sigma$ será tratatadaa como uma lista de pares ordenados $(X, T)$ onde $X$ é uma variável de tipo de $T$ é um tipo
- um conjunto de equações de tipo $C$ será tratado como uma lista de pares ordenados de tipos $(T_1, T_2)$
- o estado inicial do sistema de transição de estados é o par $([], C)$. Ou seja nenhuma substituição ainda foi produzida e $C$ possui todas as equações de tipos coletadas
- o estado final do sistema de transição é da forma $(\sigma, [])$. Esse estado final é alcançado somente quando todas as equações de tipo presentes no estado inicial foram consideradas. A substituição $\sigma$ é uma solução (a mais geral) para todas as equações de tipo $C$ do estado inicial
- um estado $(\sigma, C)$ é considerado um estado de erro, quando $C \neq []$ e não existe $(\sigma', C')$ tal que $(\sigma, C) \rightarrow (\sigma', C')$.

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. $\sigma,\ (T_1 \text{ list} = T_2 \text{ list}) :: C \quad \longrightarrow \quad \sigma,\ (T_1 = T_2) :: C$
4. $\sigma,\ (T_1 \to T_2 = T_3 \to T_4) :: C \quad \longrightarrow \quad \sigma,\ (T_1 = T_3) :: (T_2 = T_4) :: C$
5. $\sigma,\ (T_1 \times T_2 = T_3 \times T_4) :: C \quad \longrightarrow \quad \sigma,\ (T_1 = T_3) :: (T_2 = T_4) :: C$
6. $\sigma,\ (X = X) :: C \quad \longrightarrow \quad \sigma,\ C$
7. $\sigma,\ (X = T) :: C \quad \longrightarrow \quad \sigma@[(X, T)],\ \{T/X\}C$ se $X$ não ocorre em $T$
8. $\sigma,\ (T = X) :: C \quad \longrightarrow \quad \sigma@[(X, T)],\ \{T/X\}C$ se $X$ não ocorre em $T$

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.

Abaixo segue o algoritmo **Unify** em outra versão:

$$
\begin{array}{lcl}
\text{Unify}(\sigma,\ [\,]) & = & \sigma \\
\text{Unify}(\sigma,\ (\text{int}, \text{int}) :: C) & = & \text{Unify}(\sigma,\ C) \\
\text{Unify}(\sigma,\ (\text{bool}, \text{bool}) :: C) & = & \text{Unify}(\sigma,\ C) \\
\text{Unify}(\sigma,\ (T_1 \text{ list}, T_2 \text{ list}) :: C) & = & \text{Unify}(\sigma,\ (T_1, T_2) :: C) \\
\text{Unify}(\sigma,\ (T_1 \to T_2, T_3 \to T_4) :: C) & = & \text{Unify}(\sigma,\ (T_1, T_3) :: (T_2, T_4) :: C) \\
\text{Unify}(\sigma,\ (T_1 \times T_2, T_3 \times T_4) :: C) & = & \text{Unify}(\sigma,\ (T_1, T_3) :: (T_2, T_4) :: C) \\
\text{Unify}(\sigma,\ (X, X) :: C) & = & \text{Unify}(\sigma,\ C) \\
\text{Unify}(\sigma,\ (X, T) :: C) & = & \\
\text{Unify}(\sigma,\ (T, X) :: C) & = & \text{se } X \text{ não ocorre em } T \text{ então Unify}(\sigma@[(X, T)],\ \{T/X\}C) \\
 & & \text{senão } \textbf{falha} \\
\text{Unify}(\sigma,\ (\_,\ \_) :: C) & = & \textbf{falha}
\end{array}
$$

**Substituições de Tipo**

Uma *substituição de tipo* é um mapeamento de variáveis de tipo para tipos. Se $\sigma$ é uma substituição de tipos e $T$ é um tipo a notação $\sigma\ T$ representa a aplicação desse mapeamento $\sigma$ ao tipo $T$ que resulta em um tipo $T'$ onde toda variável de tipo $X$ em $T$ é substituida pelo tipo $\sigma(X)$.

Abaixo segue a definição da operação de aplicação de uma substituição $\sigma$ a um tipo $T$:

$$
\begin{array}{lcl}
\sigma \text{ int} & = & \text{int} \\
\sigma \text{ bool} & = & \text{bool} \\
\sigma\ (T_1 \to T_2) & = & \sigma\ T_1 \to \sigma\ T_2 \\
\sigma\ (T_1 \times T_2) & = & \sigma\ T_1 \times \sigma\ T_2 \\
\sigma\ (T \text{ list}) & = & (\sigma\ T) \text{ list} \\
\sigma\ X & = & T \quad \text{se } (X, T) \in \sigma \\
\sigma\ X & = & X \quad \text{se } X \notin \text{Dom}(\sigma)
\end{array}
$$

Uma substituição de tipos pode ser representada como uma lista de pares: a substituição de tipo

$[(X, \text{bool}),\ (Y, X \to X)]$ quando aplicada ao tipo $X \to Y$ resulta no tipo

$$\text{bool} \to (X \to X)$$

No algoritmo acima, a notação $\{T/X\}C$ (ou $[(X, T)]\ C$ significa a aplicação dessa substituição a todos os tipos de todas as equações de tipo de $C$:

$$\begin{array}{lcl} \{T/X\}[\,] & = & [\,] \\ \{T/X\}((T_1, T_2) :: C) & = & (\{T/X\}T_1,\ \{T/X\}T_2) :: \{T/x\}C \end{array}$$