Sintaxe de L1 implicitamente tipada

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

$$
\begin{array}{rcl}
e & \in & \textsf{Expr} \\
e & ::= & \texttt{nval}(n) \\
 & | & \texttt{bval}(b) \\
 & | & \texttt{binop}(\mathit{bop}, e_1, e_2) \ | \ \texttt{unop}(\mathit{uop}, e) \\
 & | & \texttt{pair}(e_1, e_2) \ | \ \texttt{fst}(e) \ | \ \texttt{snd}(e) \\
 & | & \texttt{if}(e_1, e_2, e_3) \\
 & | & \texttt{id}(x) \\
 & | & \texttt{app}(e_1, e_2) \\
 & | & \texttt{fn}(x, e) \\
 & | & \texttt{let}(x, e_1, e_2) \\
 & | & \texttt{letrec}(f, y, e_1, e_2) \\
 & | & \texttt{nil} \mid \texttt{cons}(e_1, e_2) \mid \texttt{hd}\ (e) \mid \texttt{tl}\ (e) \\
 & | & \texttt{raise} \mid \texttt{try}(e_1, e_2)
\end{array}
$$

onde

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

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.

$$T \ ::= \ X \mid \textsf{int} \mid \textsf{bool} \mid T_1 \to T_2 \mid T \ \textsf{list} \mid T_1 \times T_2$$

Considere os seguintes exemplos abaixo:

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

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

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 algum 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 (1) 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**.

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

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 Γ é um ambiente de tipo mapeando variáveis declaradas para seus tipos; e é uma árvore de sintaxe abstrata da linguagem de acordo com a gramática 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.

$$\frac{}{\Gamma \vdash \texttt{nval}(n) : \mathsf{int} \mid \{\}}$$

$$\frac{}{\Gamma \vdash \texttt{bval}(b) : \mathsf{bool} \mid \{\}}$$

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

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

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

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

$$\frac{\Gamma(x) = T}{\Gamma \vdash \texttt{id}(x) : T \mid \{\}}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C_1 \qquad \Gamma \vdash e_2 : T_2 \mid C_2 \qquad X \ \textit{new}}{\Gamma \vdash \texttt{app}(e_1, e_2) : X \mid C_1 \cup C_2 \cup \{T_1 = T_2 \to X\}}$$

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

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

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

$$\frac{X \ \textit{is new}}{\Gamma \vdash \texttt{nil} : X \ \mathsf{list} \mid \{\}}$$

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

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

$$\frac{\Gamma \vdash e : T \mid C \qquad X \ \textit{is new}}{\Gamma \vdash \texttt{isempty}\ e : \mathsf{bool} \mid C \cup \{T = X \ \mathsf{list}\}}$$

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

$$\frac{X \ \textit{is new}}{\Gamma \vdash \texttt{raise} : X \mid \{\}}$$

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

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

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

pode ser vista da seguinte maneira:

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

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** 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 e que o programa submetido a **typeInfer** é bem tipado

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

Resolvendo equações de tipo com Unify

Abaixo segue o algoritmo **Unify**:

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

Substituições de Tipo

Uma *substituição de tipo* é um mapeamento de variáveis de tipo para tipos. Se σ é uma substituição de tipos e T é um tipo a notação $\sigma\ T$ representa a aplicação desse mapeamento σ 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 σ 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:

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