Type system del linguaggio Myfun

Prof. Gennaro Costagliola
Corso di Compilatori A.A. 2021/22

December 7, 2021

Le seguenti regole di tipo **non coprono** l'intero linguaggio. Lo studente deve fornire le regole mancanti in base a quanto discusso in classe. In casi di dubbi, chiedere al docente.

Le regole sono formate dall'*antecedente*, che è situato nella parte alta della regola e che può anche non essere presente, ed il *conseguente* che è la parte bassa della regola.

Ci sono due modi per leggere le regole: dichiarativo e operativo. Si prenda ad esempio la regola per l'identificatore qui di sotto.

dichiarativo o top-down : **se** l'operazione di lookup di id nell'ambiente Γ restituisce il tipo τ **allora** è dimostrato che il tipo di id in Γ è τ;

operativo o bottom-up : per calcolare il tipo di id nel type environment Γ bisogna fare il lookup di id in Γ ed usare il tipo restituito τ come tipo di id.

Quando si usano le regole per sviluppare l'analizzatore semantico conviene leggere le regole dal basso verso l'alto, dal *conseguente* all'*antecedente*, in modo operativo. Il conseguente, ovvero il costrutto fra i simboli $\vdash$ e :, deve sempre fare riferimento ad un nodo dell'albero sintattico.

Identificatore

$$\frac{\Gamma(id) = \tau}{\Gamma \vdash id : \tau}$$

Costanti

$\Gamma \vdash int\_const : integer$
$\Gamma \vdash string\_const : string$
$\Gamma \vdash true : boolean$
$\Gamma \vdash false : boolean$

Lista di istruzioni

$$\frac{\Gamma \vdash stmt_1 : notype \quad \Gamma \vdash stmt_2 : notype}{\Gamma \vdash stmt_1; stmt_2 : notype}$$

Chiamata a funzione con o senza tipo di ritorno (espressione o istruzione)

$$\frac{\Gamma \vdash f : \tau_1 \times \ldots \times \tau_n \to \tau \quad \Gamma \vdash e_i : \tau_i^{i \in 1 \ldots n}}{\Gamma \vdash f(e_1, \ldots, e_n) : \tau}$$

$$\frac{\Gamma \vdash f : \tau_1 \times \ldots \times \tau_n \to notype \quad \Gamma \vdash e_i : \tau_i^{i \in 1 \ldots n}}{\Gamma \vdash f(e_1, \ldots, e_n) : notype}$$

Assegnazione

$$\frac{\Gamma(id) = \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash id := e : notype}$$

Blocco dichiarazione-istruzione
(il type environment dell'istruzione viene esteso con la dichiarazione, prima di fare il controllo di tipo dell'istruzione):

$$\frac{\Gamma[id \mapsto \tau] \vdash stmt : notype}{\Gamma \vdash \tau\ id; stmt : notype}$$

Istruzione while

$$\frac{\Gamma \vdash e : boolean \quad \Gamma \vdash block : notype}{\Gamma \vdash \textbf{while}\ e\ \textbf{loop}\ block\ \textbf{end loop} : notype}$$

Istruzione if-then

$$\frac{\Gamma \vdash e : boolean \quad \Gamma \vdash block : notype}{\Gamma \vdash \textbf{if}\ e\ \textbf{then}\ block\ \textbf{end if} : notype}$$

Operatori unari (vedi Table 1):

$$\frac{\Gamma \vdash e : \tau_1 \quad optype1(op_1, \tau_1) = \tau}{\Gamma \vdash op1\ e : \tau}$$

Operatori binari (vedi Table 2):

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad optype2(op_2, \tau_1, \tau_2) = \tau}{\Gamma \vdash e_1\ op_2\ e_2 : \tau}$$

op1	operando	risultato
MINUS	integer	integer
MINUS	real	real
NOT	boolean	boolean

Table 1: Tabella per *optype1*. Esempio di uso: *optype1*(NOT, boolean) = boolean

op	operando1	operando2	risultato
PLUS, TIMES, ...	integer	integer	integer
PLUS, TIMES, ...	integer	real	real
PLUS, TIMES, ...	real	integer	real
PLUS, TIMES, ...	real	real	real
STR_CONCAT	string	string	string
AND, OR	boolean	boolean	boolean
LT, LE, GT, ...	integer	integer	boolean
LT, LE, GT, ...	real	integer	boolean
LT, LE, GT, ...	integer	real	boolean
LT, LE, GT, ...	real	real	boolean

Table 2: Tabella per *optype2*. Esempio di uso: *optype2*(TIMES, real, integer) = real