Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

3 Luglio 2020

Nota Bene. Alla fine del compito, fare una foto a tutto il compito col cellulare usando una applicazione che esegue scansioni, tipo CamScanner, e inviarla per email a `cosimo.laneve@unibo.it` .

Si consideri la seguente grammatica (scritta in ANTLR)

```
prg :  'let' dec 'in' stm ;
dec :  (type Id ';')+ ;
type:  'int' | 'double' ;
exp :  Integers | Doubles | Id | exp '+' exp ;
stm :  (Id '=' exp ';')+
```

dove

- gli `Integers` sono sequenze non vuote ndi cifre prefissate dal segno `+` o `-`;
- i `Doubles` sono sequenze non vuote di cifre con esattamente un punto "`.`" e prefissate dal segno `+` o `-`;
- gli `Id` sono gli identificatori (sequenze non vuote di caratteri);
- l'operazione di somma "`+`" è *overloaded*, cioè: in $e1$+$e2$, se sia $e1$ che $e2$ sono interi, allora il risultato è un intero, altrimenti è un double;
- nell'assegnamento `x = e ;`
  - se `x` è intero ed `e` è double allora il valore di `e` viene troncato prima di essere memorizzato in `x`;
  - se `x` è double ed `e` è intero allora il valore di `e` viene esteso con "`.0`" prima di essere memorizzato in `x`.

Esercizi

9 1. dare tutte le regole di inferenza per la verifica dei tipi del linguaggio di sopra.
[**SUGGERIMENTO**: La regola di inferenza del programma ritorna un `stm` in un linguaggio esteso in cui si aggiungono i cast espliciti "`x = (double)e ;`" oppure "`x = (int)e ;`" dove sono necessari;]

4 2. verificare, scrivendo l'albero di prova, che il programma seguente sia correttamente tipato:

```
let double x; int y; in y = 5.4 ; x = 3 + y ;
```

2 3. scrivere un programma che non sia tipabile nel sistema definito e spiegarne il motivo;

9 4. definire il codice intermedio di `e1 + e2`, di `x = e ;` (e, nel caso si siano aggiunti i cast espliciti, di `x = (double)e ;` di `x = (int)e ;`) assumendo che

(a) tutti i registri sono a 8 byte (memorizzano double);

(b) ci siano due operazioni di addizione: `iadd $r1 $r2 $r3` e `dadd $r1 $r2 $r3`. L'operazione `iadd $r1 $r2 $r3` fa la somma prendendo la parte intera di `$r1` ed `$r2` e memorizzano il risultato in `$r3` (con un sufisso "`.0`"); `dadd` fa la somma tra double.

(c) c'è un'operazione `isw $r0 k($r1)` che memorizza la parte intera di `$r0` ad offset `k` dell'indirizzo in `$r1`. In questo caso tale indirizzo occupa 4 byte.

(d) c'è un'operazione standard `sw $r0 k($r1)` che memorizza `$r0` ad offset `k` dell'indirizzo in `$r1`. In questo caso tale indirizzo occupa 8 byte.