Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

19 Giugno 2019

Esercizio 1 (6 punti). Data la grammatica (le lettere minuscole sono simboli terminali)

$$\begin{array}{lcl} S \to & Ab & | \quad Bc \\ A \to & aA & | \quad \varepsilon \\ B \to & acB & | \quad \varepsilon \end{array}$$

Verificare, costruendo l'opportuna tabella, se la grammatica è LL(1). Nel caso non lo sia, esiste un k per cui essa è LL(k). Motivare la risposta.

Esercizio 2 (9 punti). Si assuma di avere un linguaggio con sottotipi (e relazione di sottotipo `<:`).

1. Definire la regola semantica per il comando `x := E` e scrivere in pseudo-codice la funzione `checkStat` che la implementa.

2. Scrivere l'albero di derivazione per il comando

   ```
   x:= y ; y:= z ; z:= new C() ;
   ```

   per l'ambiente $[x \mapsto \mathtt{C}_x, y \mapsto \mathtt{C}_y, z \mapsto \mathtt{C}_z]$. Quela è la relazione tra $\mathtt{C}_x$, $\mathtt{C}_y$ e $\mathtt{C}_z$?

Esercizio 3 (9 punti). Definire la funzione `code_gen` per il comando

```
interleave C and C' upto E times
```

che (1) calcola E e sia v il suo valore e (2) esegue una volta `C` e una volta `C'` in maniera tale che il numero totale di esecuzioni sia v.

Quindi applicare le regole di sopra al comando

```
interleave y := y+1 and x := x-1 upto x+y times
```

assumendo che la variabile `x` si trovi ad offset +4 del frame pointer `$fp`, mentre la variabile `y` si trova nell'ambiente statico immediatamente esterno all'ambiente corrente e a offset +8.

Esercizio n°1

	a	b	c	$
S	S→Ab S→Bc	S→Ab	S→Bc	
A	A→aA	A→ε		
B	B→acB		B→ε	

S→Ab FIRST(Ab) = FIRST(A)\{ε} ∪ FIRST(b) = {a, b}
NULLABLE(A) = true
FIRST(A) = FIRST(aA) ∪ FIRST(ε) = {a, ε}

S→Bc FIRST(Bc) = FIRST(B)\{ε} ∪ FIRST(c) = {a, c}
NULLABLE(B) = true
FIRST(B) = FIRST(acB) ∪ FIRST(ε) = {a, c, ε}

A→aA FIRST(aA) = {a}

A→ε FIRST(ε) = {ε} FOLLOW(A) = FIRST(b)\{ε} ∪ FOLLOW(A) = {b}

B→acB FIRST(acB) = {a}

B→ε FIRST(ε) = {ε} FOLLOW(B) = FOLLOW(B) ∪ FIRST(c)\{ε} = {c}

Essendoci una 'entry' in cui sono presenti più dichiarazioni (LL[S, a]), la grammatica non è LL(1). Osservando le produzioni, possiamo osservare che le stringhe formate sono nella forma a^*b o $(ac)^*c$. Nel caso massimo*, per capire quale delle due produzioni di S deve essere usata, dobbiamo leggere i primi due caratteri, per verificare se è presente una

in cui la stringa sia b o c ci basta un carattere. In ogni caso, a causa della prima considerazione, il linguaggio risulta LL(K) per K=2.
* dimostrare come aab o aca

Esercizio n° 2

$$\frac{\Gamma(x).type = T \quad \Gamma \vdash E : T' \quad T' <: T}{\Gamma \vdash x := E : void}$$

$$\frac{\Gamma(x).type = C_x \quad \Gamma(y).type = C_y}{\Gamma \vdash x := y : void} \qquad \frac{\Gamma(y).type = C_y \quad \Gamma(z).type = C_z}{\Gamma \vdash y := z : void} \qquad \frac{\Gamma(z).type = C_z \quad \Gamma \vdash z = new\ C() : C}{\Gamma \vdash z = new\ C() : void}$$

$$\Gamma \vdash x := y;\ y := z;\ z := new\ C();\ : void$$

$$\Gamma = [x \mapsto C_x,\ y \mapsto C_y,\ z \mapsto C_z]$$

Osservando la regola, supponendo che il programma sia ben tipato, la relazione tra i tipi è $C <: C_z <: C_y <: C_x$

Esercizio n°3 interleave C and C' upto E times

```
code_gen(Stable, interleave C and C' upto E times) =
    Fine = new_label(); inizio = new_label();
    code_gen(Stable, E)
    push $a0
    li $t1 0
    bel $a0 $t1 Fine
    inizio:
    code_gen(Stable, C)
    $a0 <- top
```

bel $r1 $r2 label
salta a label se $r1 ≤ $r2

subli $r1 $r2 n
$r1 = $r2 - n

```
subbi $a0 $a0 1
pop
push $a0
li $t1 0
bel $a0 $t1 Fine
code_gen (stable, c')
$a0 ← top
subbi $a0 $a0 1
pop
push $a0
b inizio
Fine
pop
```

Codice generato per interleave y := y+1 and x := x-1 upto x+y times

```
y+1 { y { lw $a0 0($fp)
          lw $a1 0($a1)
          lw $00 8($a1)
        push $a0
      1 { li $a0 1
          $t1 ← top
          add $a0 $t1 $a0
        { lw $a1 0($fp)
y=e     { lw $a1 $a1
          sw $a0 0($a1)
        push $a0
        li $t1 0
        bel $a0 $t1 Fine
        inizio:
        lw $a1 0($fp)
        lw $a0 0($a1)
```

```
lw $a1 0($fp)
lw $a0 0($a1)
push $a0
lw $a1 0($fp)
lw $a1 0($a1)
lw $a0 8($a1)
$t1 ← top
add $a0 $t1 $a0
pop
push $a0
li $t1 0
bel $a0 $a1 Fine
inizio:
    lw $a0 0($fp)
    lw $a1 0($a1)
    lw $00 8($a1)
```

```
push $a0
li $a0 1
```

```
push $a0
li $a0 1
$t1 <- top
add $a0 $t1 $a0
pop
lw $a1 0($fp)
lw $a1 $a1
sw $a0 ($a1)
$a0 <- top
subbi $a0 $a0 1
pop
push $a0
li $t1 0
bel $a0 $t1 fine
lw $a1 0($fp)
lw $a0 4($fp)
push $a0
li $a0 1
$t1 <- top
sub $a0 $t1 $a0
pop
lw $a1 0($fp)
sw $a0 4($a1)
$a0 <- top
subbi $a0 $a0 1
pop
push $a0
b inizio
fine:
por
```

```
boolean checkStat (EnvStable, exp x, exp e) {

  Type t1 := inferType(stable, x)
  Type t2 := inferType(stable, e)

  if (t1.class == t2.class || t1.isSupertype(t2)) {
      return true;
  else
      return false;
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

19 Dicembre 2019

Esercizio 1 (6 punti) Scrivere le definizioni **formali** di `nullable`, `first`, e `follow` per grammatiche LL(1).

Esercizio 2 (10 punti) In un linguaggio di programmazione i programmi `Prg` sono definiti da questa sintassi

$$\begin{array}{lll} \texttt{Prg} & ::= & \texttt{Fun}^* \ \texttt{Stm} \\ \texttt{Fun} & ::= & \texttt{Type Id "(" FPar ")" = Stm} \end{array}$$

dove `Type` possono essere solamente `int` e `bool`, `FPar` sono i parametri formali, cioè sequenze anche vuote del tipo $\texttt{Type}_1\ \texttt{Id}_1, \cdots, \texttt{Type}_n\ \texttt{Id}_n$, e `Stm` è la categoria sintattica dei comandi (lo `Stm` in `Prg` è il *main*). Definire

1. le regole di inferenza per analizzare programmi con mutua ricorsione [Suggerimento: servono due regole, una per costruire l'ambiente iniziale con tutti i tipi delle funzioni, l'altra per analizzare il programma;
2. definire lo pseudocodice per `CheckProg` che implementa le regole di sopra;
3. fornire l'albero di prova per il programma

```
int f(int x) =  return (g(x,x) + 1) ;
int g(int u, int v) =  return(f(u+v)) ;
print(f(1)+g(2,3)) ;
```

   assumendo i vincoli di tipo standard per i comandi e le espressioni (quelli visti a lezione).

Esercizio 3 (8 punti)

1. Definire la funzione `code_gen` per il termine `do S while E` che esegue S, quindi controlla E e se essa è vera riesegue S, altrimenti l'esecuzione termina.
2. Come verifica, si generi il codice di

```
do do ( x:= x+1 ; y:= y+x ) while (x>y) while (y<x+z)
```

   dove le variabili `x`, `y` e `z` si trovano ad offset `+4` e `+8` e `+12` del frame pointer `FP`.

Esercizio n°1

1) NULLABLE (ε) = True

NULLABLE (t): False $\quad t \in T \quad T$ = insieme simboli terminali

NULLABLE $(\alpha\gamma)$: $\begin{cases} \text{FALSE} & \text{NULLABLE}(\alpha) = \text{FALSE} \\ \text{NULLABLE}(\gamma) & \text{NULLABLE}(\alpha) = \text{TRUE} \end{cases}$

$$\text{NULLABLE}(X) = \bigvee_{X \to \gamma \text{ in } G} \text{NULLABLE}(\gamma)$$

2) FIRST $(\varepsilon) = \{\varepsilon\}$

FIRST $(t) = \{t\}$

FIRST $(\alpha\gamma)$: $\begin{cases} \text{FIRST}(\alpha) & \text{NULLABLE}(\alpha) = \text{FALSE} \\ \text{FIRST}(\alpha) \setminus \{\varepsilon\} \cup \text{FIRST}(\gamma) & \text{NULLABLE}(\alpha) = \text{TRUE} \end{cases}$

$$\text{FIRST}(X) = \bigcup_{X \to \gamma \text{ in } G} \text{FIRST}(\gamma)$$

3) FOLLOW $(S) = \{\$\}$ $\quad S$ simbolo iniziale

FOLLOW $(X) = \bigcup_{Y \to \alpha X \gamma \text{ in } G} \text{FIRST}(\gamma) \setminus \{\varepsilon\}$

$$\bigcup_{Y \to \alpha X \gamma \text{ in } G \text{ and NULLABLE}(\gamma)} \text{FOLLOW}(Y)$$

Esercizio n°2

Prg ::= Fun* stm

Fun ::= Type Id "(" FPar ")" = Stm

$$\frac{\Gamma_{Fun} \vdash F : \Gamma'_{Fun} \quad \Gamma'_{Fun} \vdash F : \Gamma''_{Fun}}{\Gamma_{Fun} \vdash FF : \Gamma''_{Fun}}$$

$$\frac{[] \vdash F : \Gamma_{Fun} \quad \Gamma_{Fun} \cdot [], 0 \vdash F : \Gamma_{Fun} \cdot \Gamma \quad \Gamma_{Fun} \cdot \Gamma \vdash S : void}{[] \vdash F^* \; S : \Gamma}$$

$$\frac{\Gamma \cdot []}{\Gamma_{Fun} \cdot [], n \vdash FF : \Gamma_{fun} \cdot \Gamma}$$

$id \notin dom(\Gamma_{Fun}) \quad [] \vdash FPar : \Gamma_h = [t_1 \to t, t_2 \to t \dots t_n \to t]$

$\Gamma_{Fun} \vdash T\ id\ (Fpar) = stm : \Gamma_{Fun}[id \mapsto \Gamma_0 \to T]$

Esercizio n°3 — do S while E

code_gen(stable, do S while E) =

```
inizio = new_label();
inizio:
    code_gen(Stable, S)
    code_gen(stable, e)
    bt $t0 inizio
```

bt $r1 label
salta a label se $r1 contiene un valore equivalente a "true"

```
inizio:
    inizio:
        lw $a1 0($fp)
        lw $a0 4($a1)
        push $a0
        li $a0 1
        $t1 ← top
        add $a0 $t1 $a0
        pop
        lw $a1 0($fp)
        sw $a0 4($a1)
        lw $a1 0($fp)
        lw $a1 $a1
        lw $a0 8($a1)
        push $a0
        lw $a1 0($fp)
        lw $a0 4($fp)
        $t1 ← top
        add $a0 $t1 $a0
```

cgen(stable, e1 + e2):

```
code_gen(Stable, e1)
push $a0
codgen(stable, e2)
$t1 ← top
add $a0 $t1 $a0
pop
```

cgen(stable, x):

```
lw $a1 0($fp)
for(i=0; i < nesting_level - lookup(stable, x).nesting_level; i++)
    lw $a1 0($a1)
lw $a0 lookup(stable, x).offset($a1)
```

cgen(stable, n): li $a0 n

cgen(stable, e1 > e2) =

```
cgen e1
push $a0
cg en e2
```

mog $r1 $r2 $r3
se $r2 > $r3
mette in $r1 true
altrimenti False

```
    pop
    lw $a1 0($fp)
    lw $t0 8($a1)
    lw $a1 0($fp)
    lw $a0 4($a1)
    push $t0
    lw $a1 0($fp)
    lw $t0 8($a1)
    $t1 ← top
    mag $a0 $t1 $t0
    bt $a0 inizio ← pop
lw $a1 0($fp)
lw $a0 8($fp)
push $a0
lw $a1 0($fp)
lw $a0 4($a1)
push $a0
lw $a1 0($fp)
lw $t0 8($a1)
push $t0
lw $a1 0($fp)
lw $t0 12($a1)
$t1 ← top
add $a0 $t1 $a0
pop
$t1 ← top
min $t0 $t1 $t0
pop
bt $a0 inizio
```

```
$t1 ← top
mag $a0 $t1 $t0
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

19 Febbraio 2020

Esercizio 1 (7 punti). Data la grammatica (le lettere minuscole sono simboli terminali, A è il simbolo iniziale)

$$\begin{array}{lll} A \to & BC & \\ B \to & aB & \mid \quad \varepsilon \\ C \to & CbB & \mid \quad c \end{array}$$

Riscrivere la grammatica rimuovendo la ricorsione sinistra e verificare se la grammatica è LL(1) costruendo l'opportuna tabella. Nel caso non lo sia, esiste un k per cui essa è LL(k)? Motivare la risposta.

Esercizio 2 (7 punti). I seguenti sono potenziali regole di tipo per il costrutto `let` in un linguaggio con sottotipaggio (<:). Dire quali regole sono corrette e quali sbagliate. Per quelle sbagliate dare (a) un codice che dovrebbe essere tipabile e non lo è; (b) un codice che è tipabile e invece non dovrebbe essere.

1. $$\frac{\Gamma \vdash e : T' \qquad \Gamma \vdash e' : T'' \qquad T' <: T}{\Gamma \vdash \texttt{let}\ T\ x = e\ \texttt{in}\ e' :\ T''}$$

2. $$\frac{\Gamma \vdash e : T' \qquad \Gamma[x : T] \vdash e' : T'' \qquad T <: T'}{\Gamma \vdash \texttt{let}\ T\ x = e\ \texttt{in}\ e' :\ T''}$$

3. $$\frac{\Gamma \vdash e : T' \qquad \Gamma[x : T'] \vdash e' : T'' \qquad T' <: T}{\Gamma \vdash \texttt{let}\ T\ x = e\ \texttt{in}\ e' :\ T''}$$

Nel caso in cui nessuna regola sia corretta, (i) dare la regola giusta e (ii) controllare che i codici di prima siano correttamente tipabili/non tipabili.

Esercizio 3 (10 punti). Definire la funzione `code_gen` per

1. la dichiarazione di funzione void come: `void f(T1 x, T2 y){ S }` ;
2. l'invocazione di funzione `f(e, e')` (`e`, `e'` sono espressioni).

Quindi, assumendo che l'etichetta che corrisponde alla seguente funzione `fact` sia `fact_label`, scrivere il codice per

```
int x = 1 ;
void fact(int n, int z){
   if (n == 0) x = z ;
   else fact(n-1, z*n) ;
}
```

Esercizio n°1

$A \to BC$

$B \to aB \mid \varepsilon$

$C \to cD$ $\quad$ $D \to bBD \mid \varepsilon$

	a	b	c	$
A	$A \to BC$		$A \to BC$	
B	$B \to ab$	$B \to \varepsilon$	$B \to \varepsilon$	$B \to \varepsilon$
C			$C \to cD$	
D		$D \to bBD$		$D \to \varepsilon$

$A \to BC$ $\quad$ $FIRST(BC) = FIRST(B) \setminus \{\varepsilon\} \cup FIRST(C) = \{a, c\}$

$NULLABLE(B) = true$

$FIRST(B) = \{a, \varepsilon\}$

$FIRST(C) = \{c\}$

$B \to aB$ $\quad$ $FIRST(aB) = \{a\}$

$B \to \varepsilon$ $\quad$ $FIRST(\varepsilon) = \{\varepsilon\}$ $\quad$ $FOLLOW(B) = FIRST(C) \setminus \{\varepsilon\} \cup FOLLOW(B) \cup FIRST(D) \setminus \{\varepsilon\} \cup FOLLOW(D) =$

$\{c, b\} \cup FOLLOW(C) \cup FOLLOW(D) =$

$\{c, b\} \cup FOLLOW(A) = \{c, b, \varepsilon\}$

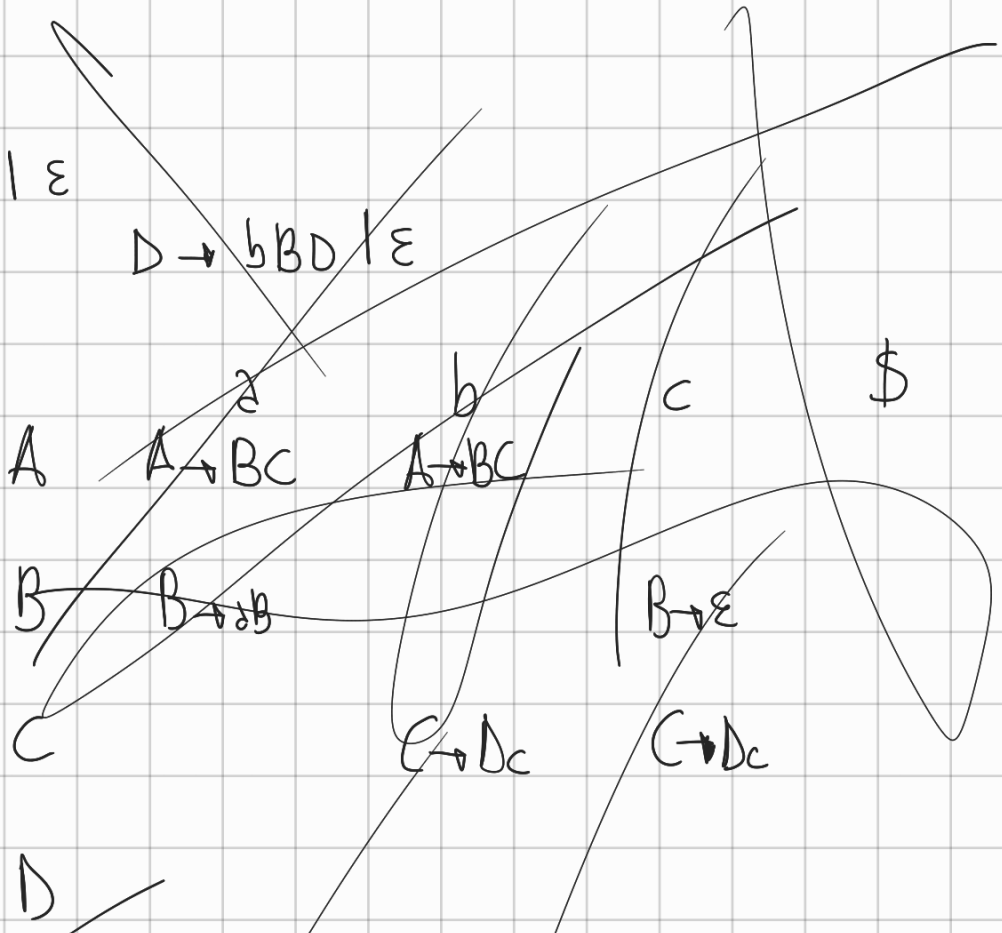
$C \to cD$ $\quad$ $FIRST(cD) = \{c\}$

$D \to bBD$ $\quad$ $FIRST(bBD) = \{b\}$

$D \to \varepsilon$ $\quad$ $FIRST(\varepsilon)$ $\quad$ $FOLLOW(D) = \{\varepsilon\}$

Esercizio n° 1

$A \to BC$
$B \to aB \mid \varepsilon$
$C \to cD$ $\quad D \to bBD \mid \varepsilon$



$A \to BC$ $\quad FIRST(BC) = FIRST(B) \setminus \{\varepsilon\} \cup FIRST(C) = \{a, b\}$

NULLABLE(B) = true

$FIRST(B) = FIRST(aB) \cup FIRST(\varepsilon) = \{a, \varepsilon\}$

$FIRST(C) = FIRST(D) = FIRST(bBD) = \{b,$ (unfinished)

$B \to aB$ $\quad FIRST(aB) = \{aB\}$

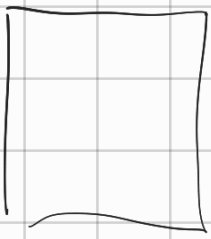
$B \to \varepsilon$ $\quad FIRST(\varepsilon) = \{\varepsilon\}$

$FOLLOW(B) = FOLLOW(B) \cup FIRST(C) \setminus \{\varepsilon\} \cup FIRST(D) \setminus \{\varepsilon\} \cup FOLLOW(D) = \{b\} \cup$

$A \to \alpha A'$ $\quad$ $A' \to \gamma A'$

Esercizio n°2

$$\frac{T \vdash e : T' \quad T \vdash e' : T'' \quad T' <: T}{\Gamma \vdash \text{let } T\ x = e \text{ in } e' : T''}$$



Es n°3

codgen (st, void F(T1 x, T2 y) {s})

```
F_act_label:
    move $fp $sp
    push $ra
    Cgen (stable, s)
    $t2 ← top
    addi $sp $sp K
    $fp ← top
    pop
    j← $n
```

codgen(stable, F(e,e')):

```
    push $fp
    Cgen (stable, e')
    push $t0
    Sgen(stable, e)
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

15 Giugno 2020

Nota Bene. Alla fine del compito, fare una foto a tutto il compito col cellulare e inviare le foto per email a `cosimo.laneve@unibo.it` .

Esercizio 1 (6 punti). Definire un analizzatore lessicale in `ANTLR` che accetta sequenze di token che a loro volta sono stringhe non vuote sull?alfabeto $\{\texttt{a}, \texttt{b}\}$ per cui non ci sono mai due occorrenze di `b` consecutive. Ad esempio `a abaa b aaaab` è un input riconosciuto.

Esercizio 2 (7 punti). Data la grammatica (le lettere minuscole sono simboli terminali, A è il simbolo iniziale)

$$\begin{array}{lllllll} S \rightarrow & Aa & | & bAc & |Bc & | & bBa \\ A \rightarrow & aA & | & \varepsilon & & & \\ B \rightarrow & bB & | & \varepsilon & & & \end{array}$$

Verificare se la grammatica è LL(1) costruendo l'opportuna tabella. Nel caso non lo sia, esiste un k per cui essa è LL(k)? Motivare la risposta.

Esercizio 3 (10 punti). Definire la funzione `code_gen` per il comando

```
for id := E to E' do S
```

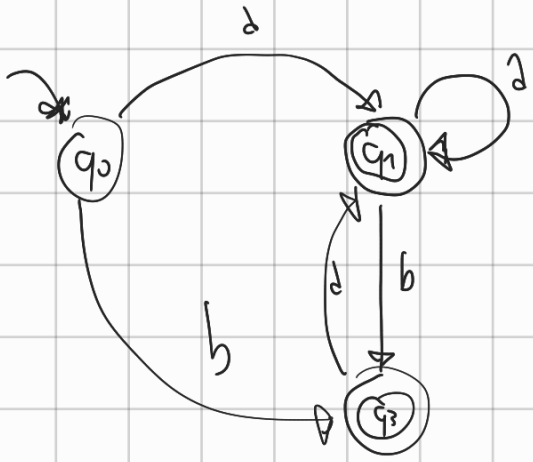
La semantica del for è: (1) si calcolano il valore delle espressioni `E` e `E'` e siano esse v e v'; (2) quindi si inizializza `id` a v e si esegue `S` se `id` $\leq v'$; (3) dopo l'esecuzione di `S`, si incrementa `id` e si riverifica se `id` $\leq v'$. L'iterazione termina quando `id` $> v'$.

Si applichi tale regola al comando

```
for x := y to z do z := x+1
```

assumendo che le variabili `x` e `y` si trovino nel record di attivazione corrente ad offset 8 e 12 del `$fp`, mentre `z` si trovi nell'ambiente statico immediatamente precedente a offset 8.

Esercizio n°①



q0: aq1 | bq2
q1: (aq1 | bq2)! | ε
q2: (aq1)! | ε

WS: (' ' | '\n' | '\t' | '\r') → skip
LC: '//' (~('\n' | '\r'))* → skip
BL: '/*' (~('/' | '*') | '/' ~'*' | '*' ~'/' BLOCKCOMMENSS)* '*/'
↓
skip

Esercizio n°2

S → Aa | bAc | Bc | bB
A → aA | ε
B → bB | ε

	a	b	c	$
S	S→Aa	S→bAc S→Bc S→bB	S→Bc	
A	A→aA A→ε		A→ε	
B	B→ε	B→bB	B→ε	

$S \to Aa$ FIRST(Aa): FIRST(A) $\setminus \{\varepsilon\} \cup$ FIRST(a) $= \{a\}$

NULLABLE(A): TRUE

FIRST(A) $= \{a, \varepsilon\}$

$S \to bAc$ FIRST(bAc): FIRST(b) $= \{b\}$

$S \to Bc$ FIRST(Bc): FIRST(B) $\setminus \{\varepsilon\} \cup \{c\} = \{b, c\}$

NULLABLE(B) = true

FIRST(B) $= \{b, \varepsilon\}$

$S \to bBa$ FIRST(bBa) $= \{b\}$

$A \to aA$ FIRST(aA) $= \{a\}$

$A \to \varepsilon$ FIRST(ε) $= \{\varepsilon\}$ FOLLOW(A): FOLLOW(A) $\cup$ FIRST(a) $\setminus \{\varepsilon\} \cup$ FIRST(c) $\setminus \{\varepsilon\} = \{a, c\}$

$B \to bB$ FIRST(bB) $= \{b\}$

$B \to \varepsilon$ FIRST(ε) $= \{\varepsilon\}$ FOLLOW(B): FOLLOW(B) $\cup$ FIRST(c) $\setminus \{\varepsilon\} \cup$ FIRST(a) $\setminus \{\varepsilon\} = \{c, a\}$

La grammatica, osservando non è LL(1). Inoltre non esiste alcun K per cui sia LL(K). Questo perché il formato delle stringhe generate è $a^* \mid ba^*c \mid b^*c \mid bb^*a$. Negli ultimi due casi, se abbiamo una sola b seguita da una a possiamo capire quale produzione sviluppare, mentre se abbiamo più b di fila non sappiamo quanti caratteri possiamo leggere per capire quale produzione sviluppare

Esercizio n°3

bm $t1 $t2 label

code_gen(stable, id := E to E' do S) =

```
        cgen(stable, E')
        push $a0
        cgen(stable, E)
inizio: $t1 <- top
        lw $a1 0($fp)
        For(i=0; i< nesting_level - lookup(stable, id).nesting_level ; i++)
            lw $a1 0($a1)
        $sw $a0 lookup(stable, id).offset($a1)
        bm $a0 $t1 Fine
        code_gen(stable, S)
        lw $a1 0($fp)
        For(i=0; i< nesting_level - lookup(stable, id).nesting_level ; i++)
            lw $a1 0($a1)
        lw $a0 lookup(stable, id).offset($a1)
        addi $a0 $a0 1
        b inizio
Fine:
        pop
```

Salta a label se $r1 > $r2

for x := y to z do z := x+1

```
    lw $a1 d($fp)
    lw $a1 d($a1)
    lw $a0 8($a1)
    push $a0
    lw $a1 0($fp)
    lw $a0 12($a1)
inizio:
    $t1 <- top
```

```
        lw $t1 0($fp)
        sw $s0 8($fp)
        bne $s0 $t1 Fine
        lw $s1 0($fp)
        lw $s0 8($s1)
        push $s0
        li $s0 1
        $t1 <- top
        add $s0 $t1 $s0
        pop
        lw $s1 0($fp)
        lw $s1 $s1
        sw $s0 8($s1)
        lw $t1 0($fp)
        lw $s0 8($fp)
        addi $s0 $s0 1
        b inizio
Fine:
pop
```

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.

Esercizio n°1

FORMATO GIUDIZI (e: espressioni , s: statement , d: dichiarazioni)

$\Gamma \vdash e : T, e'$ $\quad \Gamma \vdash s : s'$ $\quad \Gamma, n \vdash d : \Gamma', n'$ $\quad$ Formato ambiente $\Gamma = [id \mapsto T, n^*]$

$$\frac{n \in Integers}{\Gamma \vdash n : int, n}\ [int] \qquad \frac{d \in Doubles}{\Gamma \vdash d : double, d}\ [double]$$

$$\frac{\Gamma(x).type = T}{\Gamma \vdash x : T, x}\ [Id] \qquad \frac{\Gamma \vdash e_1 : int, e_1' \quad \Gamma \vdash e_2 : int, e_2'}{\Gamma \vdash e_1 + e_2 : int, e_1'\ int+\ e_2'}\ [addI]$$

$T = double \ || \ T_2 = double$

$$\frac{\Gamma \vdash e_1 : T, e_1' \quad \Gamma \vdash e_2 : T, e_2'}{\Gamma \vdash e_1 + e_2 : double, e_1'\ double+\ e_2'}\ [addD]$$

$$\frac{\Gamma \vdash e : T, e' \qquad x \in dom(\Gamma)}{\Gamma \vdash x = e : x = e'}\ [stm] \qquad \frac{\Gamma \vdash s : s' \quad \Gamma \vdash S : S'}{\Gamma \vdash s; S : s'S'}\ [Seq\ S]$$

$$\frac{x \notin dom(\Gamma)}{\Gamma, n \vdash Int\ x : \Gamma[x \mapsto int, n], n+4}\ [decI] \qquad \frac{x \notin dom(\Gamma)}{\Gamma, n \vdash double\ x : \Gamma[x \mapsto double, n], n+8}\ [decd]$$

$$\frac{\Gamma, n \vdash d, n' \quad \Gamma, n' \vdash D, n''}{\Gamma, n \vdash d; D; : \Gamma', n''}\ [seqD]$$

$$\frac{[], 0 \vdash D : \Gamma, n \quad \Gamma \vdash S : S'}{[] \vdash let\ D\ in\ S : \Gamma, S'}\ [prg]$$

Esercizio n°2

$\Gamma = [x \mapsto double, 0]$

$\Gamma' = [x \mapsto \text{double}_{,0} \; ; \; y \mapsto \text{int}_{,8}]$

$$\dfrac{\dfrac{x \notin dom([])}{[]_{,0} \vdash \text{double} x : \Gamma_{,8}}\,[decl] \quad \dfrac{y \notin dom(\Gamma)}{\Gamma_{,8} \vdash \text{int} y : \Gamma'_{,12}}\,[decl]}{[]_{,0} \vdash \text{double} x \; ; \text{int} y \; ; : \Gamma'_{,12}}\,[seqD] \quad \dfrac{\dfrac{}{\Gamma' \vdash 5.4 : \text{double}, 5.4}\,[double] \quad y \in dom(\Gamma')}{\Gamma' \vdash y = 5.4 : y = 5.4}\,[stm] \; (*)}{\Gamma' \vdash y = 5.4 \; ; x = 3+y : y = 5.4 \; ; x = 3 \; \text{int+} \; y \; ;}\,[seqS]$$

$$[]\ \text{let double } x \; ; \text{int } y \; ; \text{ in } y = 5.4 \; ; x = 3+y \; ; : \Gamma', y = 5.4 \; ; x = 3+y \; ;$$

$$\dfrac{\dfrac{}{\Gamma' \vdash 3 : \text{int}, 3}\,[int] \quad \dfrac{\Gamma(y).type = \text{int}}{\Gamma \vdash y : \text{int}, y}\,[Id]}{\dfrac{\Gamma' \vdash 3+y : \text{int}, 3 \; \text{int+} \; y \qquad x \in dom(\Gamma')}{\Gamma' \vdash x = 3+y : x = 3 \; \text{int+} \; y}\,[stm]}$$

(*)

Esercizio n°3

let int x ; double x ; in x = x + 1 ;

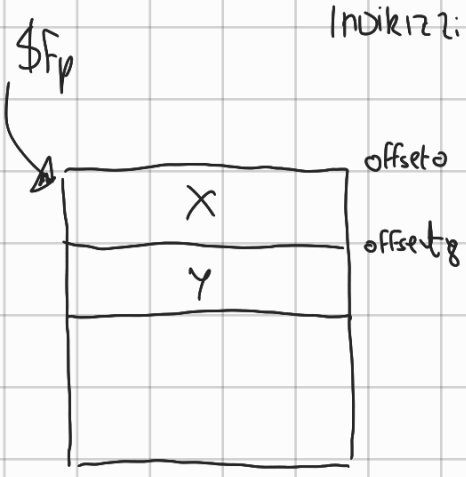
→ Doppia dichiarazione di x

Esercizio n°4

stable = Γ

```
cgen ( stable , e1 int+ e2 ):
        cgen (stable, e1)
        push $a0
        cgen (stable, e2)
        $t1 <- top
        iadd $t1 $a0 $a0
        pop
```

```
cgen( stable, e1 double+ e2 ) =
        cgen (stable, e1)
        push $a0
        cgen (stable, e2)
        $t1 ← top
        dadd $t1 $a0 $a0
        pop
```



INDIRIZZI: piccoli

INDIRIZZI: GRANDI

```
cgen ( stable, x:=e) :
        cgen(stablo, e)
        if( lookup ( stable, x) .type = "int")
            isw $a0 lookup(stable, x).offset ($fp)
        else
            sw $a0 lookup (stable, x) .offset ($fp)
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

20 Luglio 2020

Nota Bene. Quando avete terminato, 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 :  ('int' Id ';')+ ;
exp :  Integers | Id | exp '+' exp ;
stm :  (Id '=' exp ';')+
```

dove

- gli `Integers` sono sequenze non vuote di cifre prefissate dal segno `+` o `-`;
- gli `Id` sono gli identificatori (sequenze non vuote di caratteri);

Esercizi

1. (**punti 2**) completare l'input di ANTLR con le regole per l'analizzatore lessicale che riguardano `Integers` e `Id`;
2. (**punti 9**) dare tutte le regole di inferenza per verificare l'uso di identificatori non inizializzati. Ad esempio `let int x; int y; in x = 3 + y ;` è un programma erroneo secondo l'analisi semantica. L'analisi semantica ritorna anche informazioni sull'offset degli identificatori (vedi punto 4);
3. (**punti 4**) verificare, scrivendo l'albero di prova, che il programma seguente sia correttamente tipato:

   `let int x; int y; in y = 5 ; x = 3 + y ;`

4. (**punti 9**) definire il codice intermedio *per tutti i costrutti del linguaggio*, in particolare allocando lo spazio necessario sulla pila per memorizzare i valori degli identificatori (che occupano sempre 4 byte).

Esercizio n°1

FRAGMENT DIGIT : '0'..'9' ;

FRAGMENT CHAR: ('A'...'Z') | ('a'..'z') ;

Integers : '-'(DIGIT+) | '+'(DIGIT+) → ('-'|'+') DIGIT+

ID: CHAR (CHAR | DIGIT)*

Esercizio n°2

d = dichiarato i: inizializzato

FORMATO GIUDIZI effetti : [d, i]

$\Sigma \vdash e$ $\Sigma \vdash s : \Sigma'$ $\Sigma, n \vdash d : \Sigma', n'$

$$\frac{n \in Integers}{\Sigma \vdash n}\,[Int] \qquad \frac{\Sigma(x).effect = i}{\Sigma \vdash x}\,[id] \qquad \frac{\Sigma \vdash e_1 \quad \Sigma \vdash e_2}{\Sigma \vdash e_1 + e_2}$$

$$\frac{\Sigma \vdash e \qquad x \in dom(\Sigma)}{\Sigma \vdash x = e : \Sigma[x \mapsto i]}\,[stm] \qquad \frac{\Sigma \vdash s : \Sigma'' \quad \Sigma'' \vdash S : \Sigma'}{\Sigma \vdash s; S; : \Sigma'}\,[seqS]$$

$$\frac{x \notin dom(\Sigma)}{\Sigma, n \vdash int\ x : \Sigma[x \mapsto d, n], n+4}\,[dec] \qquad \frac{\Sigma, n \vdash d : \Sigma', n' \quad \Sigma', n' \vdash D : \Sigma'', n''}{\Sigma, n \vdash d; D; : \Sigma'', n''}\,[seqD]$$

$$\frac{[\,], 0 \vdash D : \Sigma, n \qquad \Sigma \vdash S : \Sigma'}{[\,] \vdash let\ D\ in\ S : \Sigma'}\,[prg]$$

Esercizio n° 4

stable = Σ

Gen (stable, n): li $sp n

cgen(stable, x) = lw $a0 lookup(stable, x).offset($sp)

cgen(stable, e1 + e2) = cgen(stable, e1)
push $a0
cgen(stable, e2)
$t1 ← top
add $a0 $t1 $a0
pop

cgen(stable, x := e) = cgen(stable, e)
sw $a0 lookup(stable, x).offset($fp)

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

18 Settembre 2020

Nota Bene. Quando avete terminato, 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` .

Esercizio 1 (punti 6) Gli identificatori di un linguaggio di programmazione devono iniziare e terminare con "_" e tra questi due caratteri ci possono essere solo lettere maiuscole e cifre (in qualunque ordine) con il vincolo che il numero di lettere e quello delle cifre sia sempre pari. Definire l'analizzatore lessicale per questi identificatori in `ANTLR`.

Esercizio 2 (punti 9) Si consideri la seguente grammatica:

```
S  ->  S B   |   y
B  ->  B x   |   A x
A  ->  z     |   z S y
```

1. verificare, costruendo la tabella che la grammatica non è LL(1);
2. modificare la grammatica per renderla LL(1) e dimostrarlo costruendo la tabella.

Esercizio 3 (punti 9) Si consideri il comando iterativo

$$\texttt{loop}\ k\ \{S\}$$

dove k è una costante intera. Quando $k > 0$, questo comando itera esattamente k volte il corpo S. Quando $k \leq 0$ il comando non fa niente.

1. Scrivere la `cgen` per questo comando;
2. generare il codice intermedio per

   ```
   loop 34 { x = x+1 ; loop 25 { y = x+y ;} }
   ```

   assumendo che `x` sia ad offset 4 del record di attivazione corrente e `y` sia ad offset 4 del record di attivazione dell'ambiente statico immediatamente esterno.

Esercizio n°3

loop K {S}

```
cgen ( stable, look K {S} ) =
    inizio = new_label()
    Fine = new_label()
    cgen(Stable, K)
    push $a0
    inizio:
     li $t1 0
    bne $a0 $t1 Fine
    cgen (stable, S)
    $a0 ← top
    subi $a0 $a0 1
    pop          } sw $a0 0($sp)
    push $a0     }
    b inizio
    Fine:
    pop
```

bne $t1 $t2 label
salta a label se $t1 ≠ $t2

loop 34 {x = x+1 ; loop 25 {y = x+y ;}}

```
 li $a0 34
 push $a0
inizio:
   li $t1 0
   bne $a0 $t1 Fine
   lw $a1 0($Fp)
   lw $a0 4($a1)
   push $a0
```

```
push $s0
li $s0 1
$t1 ← top
add $s0 $t1 $s0
pop
lw $s1 0($Fp)
sw $s0 4($s1)
li $s0 25
push $s0
inizio:
    li $t1 0
    bne $s0 $t1 Fine
    lw $s1 0($Fp)
    lw $s0 4($s1)
    push $s0
    lw $s1 0($Fp)
    lw $s1 0($s1)
    lw $s0 8($s1)
    $t1 ← top
    add $s0 $t1 $s0
    pop
    lw $s1 0($Fp)
    lw $s1 $s1
    sw $s0 8($Fp)
    $s0 ← top
    subi $s0 $s0 1
    pop
    push $s0
    b inizi
Fine:
    pop
$s0 ← top
```

```
        subbi $t0 $t0 1
        pop
        push $t0
        b inizio
    Fine
        pop
```

Esercizio n°2

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

28 Maggio 2021

Nota Bene. Quando avete terminato, 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` .

I programmi di un linguaggio di programmazione sono blocchi `Dec Stm` dove

- `Dec` sono sequenze di dichiarazioni di identificatori interi (int);
- `Stm` sono sequenze di comandi che possono essere
  - assegnamenti di una espressione `Exp` a una variabile;
  - iterazioni while (la guardia del condizionale è una espressione intera, la semantica è quella di `C`).
- `Exp` possono essere costanti intere, identificatori o espressioni con somma.

Esercizi

1. (**punti 6**) definire l'input *completo* di `ANTLR` per la grammatica del linguaggio di sopra;
2. (**punti 9**) dare tutte le regole di inferenza per verificare il corretto uso degli identificatori (identificatori non dichiarati o di dichiarazioni multiple) e per gestire gli offset nella generazione di codice.
3. (**punti 9**) definire il codice intermedio *per tutti i costrutti del linguaggio*, in particolare per il programma. Ricordate che la `cgen` prende come input anche l'ambiente/tabella dei simboli nei vari nodi dell'albero sintattico. Fate attenzione alla gestione degli accessi al record di attivazione.

   Generare il codice intermedio per il codice

```
int x; int z;
x = 4; z = x+5; while (z - 3){ z = z-x ; while (x){ x = x-1; } }
```

Andrea Citini 0000 984489

Esercizio n°1

Prg: Dec Stm ;
Stm: (Id '=' Exp ';' | While '(' e ')' '{' Stm '}')* ;
Dec: (int Id ';')* ;
Exp: Integer | Id | Id '=' Exp '+' Exp ;
Fragment CHAR: 'a'..'z' | 'A'..'Z' ;
Id: CHAR+ ;
Fragment DIGIT: '0'..'9' ;
Integer: ('+'|'-') DIGIT+ ;
WS: (' '|'\n'|'\r'|'\t') → skip ;
LN: ~~~ → skip ;
BL: ~~~ → skip ;

Esercizio n°2

FORMATI GIUDIZI

$\Gamma \vdash E$ $\quad \Gamma, n \vdash D : \Gamma', n' \quad \Gamma \vdash S$

FORMATO AMBIENTE $\Gamma = [Id \mapsto n, \dots]$ (dove n indica l'offset)

$$\frac{n \in Integers}{\Gamma \vdash n}\ [Int] \qquad \frac{x \in dom(\Gamma)}{\Gamma \vdash x}\ [Id] \qquad \frac{\Gamma \vdash e_1 \quad \Gamma \vdash e_1}{\Gamma \vdash e_1 + e_2}\ [Add]$$

$$\frac{x \notin dom(\Gamma)}{\Gamma, n \vdash int\ x : \Gamma[x \mapsto n], n+4}\ [Dec] \qquad \frac{\Gamma, n \vdash d : \Gamma', n' \quad \Gamma', n' \vdash D : \Gamma'', n''}{\Gamma, n \vdash d\,D ;: \Gamma'', n''}\ [SeqD]$$

$$\frac{x \in dom(\Gamma) \quad \Gamma \vdash e}{\Gamma \vdash x = e}\ [StmAs] \qquad \frac{\Gamma \vdash e \quad \Gamma \vdash S}{\Gamma \vdash While\ (e)\ \{S\}}\ [StmWh]$$

$$\frac{\Gamma \vdash s \quad \Gamma \vdash S}{\Gamma \vdash S ; S ;}\ [SecStm] \qquad \frac{[], 0 \vdash D : \Gamma, n \quad \Gamma \vdash S}{[] \vdash D\,S : h-4}\ [Prg]$$

Andrea Citina

Esercizio n° 3

cgen(stable, n) = li $a0 n cgen(stable, x) = lw $a0 lookup(stable, x).offset($Fp)

cgen(stable, e1 + e2) = cgen(stable, e1)
push $a0
cgen(stable, e2)
$t1 ← top
add $a0 $t1 $a0 ⇒ ($a0 = $t1 + $a0)
pop

cgen(stable, x=e) = cgen(stable, e)
sw $a0 lookup(stable, x).offset($Fp)

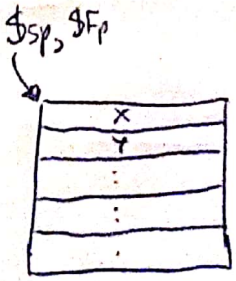
cgen(stable, while (e) {s}) =
inizio_loop = new_label(); Fine = new_label();
inizio_loop:
cgen(stable, e)
beq $a0 0 Fine
cgen(stable, s)
b inizio_loop
Fine:

beq $t1 n label
salta a label se il valore di $t1 è uguale a n
b label → salta a label incondizionatamente

cgen(stable, s; S;) = cgen(stable, s)
cgen(stable, S)

cgen([ ], P) = $sp ← P.offset
$Fp ← P.offset
cgen(P.declaration.stable, P.statement)

(vuoto)



P.declaration.stable = ambiente Γ creato con le regole precedenti, dove vengono indicati gli offset delle varie variabili

Generazione codice esercizio n°3

Andrea Grimaldi

cgen([], int x, int z; x=4, z=x+5; while(z<13){z=2*x;}; while(x){x=x+1;};) =

```
$sp <- 4
$fp <- 4
li $a0 4
sw $a0 0($fp)

lw $a0 0($fp)
push $a0
li $a0 5
$t1 <- top
add $a0 $t1 $a0
pop
sw $a0 4($fp)
inizio_loop:
        lw $a0 4($fp)
        push $a0
        li $a0 3
        $t1 <- top
        add $a0 $t1 $a0
        pop
        beq $a0 0 Fine
        lw $a0 4($fp)
        push $a0
        lw $a0 0($fp)
        $t1 <- top
        add $a0 $t1 $a0
        pop
        sw $a0 4($fp)
        inizio_loop:
            lw $a0 0($fp)
            beq $a0 0 Fine
            lw $a0 0($fp)
            push $a0 <------------ ** li $a0 1
            $t1 <- top
            add $a0 $t1 $a0
            pop
            sw $a0 0($fp)
            b inizio_loop
        Fine:
        b inizio_loop
fine:
```