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.

19-06-2019

giovedì 19 maggio 2022 13:36

ES. 1

$S \to Ab \mid Bc$

$A \to aA \mid \varepsilon$

$B \to acB \mid \varepsilon$

	a	b	c	$
S	S → Bc S → Ab			
A				
B				

Null(S) = Null(Ab) ∨ Null(Bc) = false ; Null(A) = true = Null(B)

First(S) = {a, b, c} First(A) = {a, ε} = First(B)

Follow(S) = {$} Follow(A) = {b} Follow(B) = {c}

1) NO LL(1) perché due produzioni in una cella

2) Esiste un **K** per cui essa è **LL(K)**?

Lasciando LL(1), quando il compilatore legge come prima lettera 'a' non sa quale produzione scegliere tra S->Ab e S->Bc, perché la prima lettera di entrambe le produzioni sarà 'a'.
Per differenziarle, in questo caso, bisogna leggere un carattere in più perché il secondo carattere di S->Bc è 'c', diverso dal secondo carattere di S->Ab che può essere 'a' o 'b'.

ES. 2

1) $\dfrac{\Gamma(x) = T_1 \quad \Gamma \vdash E : T_2 \quad T_2 <: T_1}{\Gamma \vdash x = E : void}$

```
public Node checkStat(){
    Node l = left.typecheck()
    Node r = right.typecheck()
    if(SimpleLab.isSubType(r,l)){
        return VoidTypeNode()
    }else{
        new Exception("Tipi diversi")
    }
}
```

2) x=y; y=z; z=new C() $\Gamma = [x \mapsto C_x, y \mapsto C_y, z \mapsto C_z]$

$$\dfrac{\dfrac{\Gamma(x) = C_x \quad \Gamma(y) = C_y \quad C_y <: C_x}{\Gamma \vdash x=y ; : \Gamma[x \mapsto C_y]} \quad \dfrac{\Gamma'(y) = C_y \quad \Gamma'(z) = C_z \quad C_z <: C_y}{\Gamma' \vdash y=z ; : \Gamma''[y \mapsto C_z]} \quad \dfrac{\Gamma''(z) = C_z \quad C <: C_z}{\Gamma'' \vdash z = new\ C() : \Gamma'''[z \mapsto C]}}{\Gamma \vdash x=y; y=z; z=new\ C() : \Gamma'''}$$

RELAZIONE: $C <: C_z <: C_y <: C_x$

ES. 3

```
cgen(stabla, interleave C and C' upto E times) =
        loop = newLabel();
        end = newLabel();
        cgen(stabla, E)
        push $a0
loop:   li $t1 0
        lw $a0 0($sp)
        bbq $a0 $t1 end     V ≤ 0
        cgen(stabla, C)
        $t1 ← top           // t1 = V
        pop
        subi $a0 $t1 1      // V = V-1
        push $a0            // push ultimo V
        li $t1 0
        bbq $a0 $t1 end
        cgen(stabla, C')
        $a0 ← top
        subi $a0 $a0 1
        pop
        push $a0
        b loop
end:    pop
```

```
cgen(stabla, interleave y = y+1 and x = x+1 upto x+y times) =
        loop = newLabel();
        end = newLabel();
        lw $al 0($fp)
        lw $al 0($al)
        lw $a0 8($al)       // a0 = y
        push $a0
        lw $t1 4($fp)       // t1 = x
        $a0 ← top
        pop
        add $a0 $t1 $a0     // x+y
        push $a0            // x+y
loop:   li $t1 0
        $a0 ← top
        bbq $a0 $t1 end     V ≤ 0
        lw $al 0($fp)
        lw $al 0($al)
        lw $al 0($al)
        lw $a0 8($al)       // a0 = y
        addi: $a0 $a0 1
        lw $al 0($fp)
        lw $al 0($al)
        lw $al 0($al)
        sw $a0 8($al)
        $t1 ← top           // t1 = V
        pop
        subi $a0 $t1 1      // V = V-1
        push $a0            // push ultimo V
        li $t1 0
        bbq $a0 $t1 end
        lw $al 0($fp)
        lw $al 0($al)
        lw $a0 4($al)
        subi $a0 $a0 1
        lw $a1 0($fp)
        lw $al 0($al)
        sw $a0 4($al)
        $a0 ← top
        subi $a0 $a0 1
        pop
        push $a0
        b loop
end:    pop
```

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) ;
}
```

19-02-2020

giovedì 19 maggio 2022 15:33

ES. 1

A -> BC
B -> aB | ε
C -> CbB
C -> c

NO RICORSIONE SINISTRA →

A -> BC
B -> aB | ε
C -> cD
D -> bBD | ε

First(A) = {a, c} First(B) = {a, ε}
First(C) = {c} First(D) = {b, ε}

Follow(A) = Follow(C) = Follow(D) = {$}
Follow(B) = {b, c, $}

	a	b	c	$
A	A -> BC		A -> BC	
B	B -> aB	B -> ε	B -> ε	B -> ε
C			C -> cD	
D		D -> bBD		D -> ε

E' LL(1)

ES. 2

$$\frac{\Gamma \vdash e : T' \quad \Gamma[x:T] \vdash e' : e'' : T'' \quad T' <: T}{\Gamma \vdash \text{let } T\ x = e \text{ in } e' ; : T''} \ [\text{let-In}]$$

esempio a - prima regola:
let int x = 5 in { int y = x } la x in e' non esiste

esempio b - prima regola: B <: A
let A x = new B(); Bx = new B() in print(x)
ridefinisce x ma dovrebbe dare errore

esempio a - seconda regola: B <: A
let A x = new B() in print(x)
inventati sottotipo e supertipo

esempio b - seconda regola:
let B x = new A() in print(x)
è tipabile per la regola ma non dovrebbe essere in generale perché B <: A

esempio a - terza regola
let A x = new B() in x = new A()
per la regola x nella ST ha tipo B quindi non posso fare new A()

esempio b - terza regola B <: A C <: B
let A x = new B() in x = new C()

ES. 3

```
gen(stable, void f(T1 x, T2 y){S}) =
    lookup(stable, f).label :
            move $fp $sp
            push $ra
            gen(stable, S)
            $ra <- top
            addi $sp $sp 12   // K = 4·n + 4
            $fp <- top
            pop
            jr $ra

gen(stable, f(e, e')) =
    push $fp
    gen(stable, e')
    push $a0
    gen(stable, e)
    push $a0
    lw $al 0($fp)
    for(i=0, i < nesting level - lookup(stable, f).nesting level; i++){
        lw $ al 0($al)
    }
    push $al
    jal lookup(stable, f).label

gen(stable, int x=1; void fact(int n, int z){ if(n==0) x=z; else fact(n-1, z*n)})
    lw $al 0($fp)
    li $a0 1
    sw $a0 lookup(stable, x).offset($al)
    fact_label :
        move $fp $sp
        push $ra
        gen(stable, if(n==0) x=z; else fact(n-1, z*n))
        $ra <- top
        li $t1 12   // K = 4·n + 4 = 4·2 + 4   ← PARAMETRI
        addu $sp $sp $t1
        $fp <- top
        pop
        jr $ra

gen(stable, if(n==0) x=z; else fact(n-1, z*n))
    if_label = new Label()
    lw $al 0($fp)
    lw $a0 lookup(stable, n).offset($al)
    push $a0
    li $a0 0
    $t1 <- top
    pop
    beq $t1 $a0 if_label
        push $fp
        gen(stable, z*n)
        push $a0
        gen(stable, n-1)
        push $a0
        lw $al 0($fp)
        for(i=0, i < nesting level - lookup(stable, f).nesting level; i++){
            lw $ al 0($al)
        }
        push $al
        jal lookup(stable, f).label
    if_label: lw $al 0($fp)
              lw $a0 lookup(stable, z).offset($al)
              sw $a0 lookup(stable, x).offset($al)

gen(stable, z*n) =
    lw $al 0($fp)
    lw $a0 lookup(stable, z).offset($al)
    push $a0
    lw $al 0($fp)
    lw $a0 lookup(stable, n).offset($al)
    $t1 <- top
    mult $a0 $t1 $a0
    pop

gen(stable, n-1)
    lw $al 0($fp)
    lw $a0 lookup(stable, n).offset($al)
    push $a0
    lw $al 0($fp)
    li $a0 1
    $t1 <- top
    sub $a0 $t1 $a0
    pop
```

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.

15-06-2020

giovedì 19 maggio 2022 09:40

ES. 1

```
// Definire un analizzatore lessicale in ANTLR che accetta sequenze
// di token che a loro volta sono stringhe non vuote sull'alfabeto {a, b}
// per cui non ci sono mai due occorrenze di b consecutive.
// Ad esempio a abba b aaaab 'e un input riconosciuto.

init : (TOKEN1)+;

TOKEN1 : 'a' | 'b' | 'a' TOKEN1 | 'b' TOKEN2;

TOKEN2 : 'a' TOKEN1;

WS : (' '|'\t'|'\n'|'\r')-> skip;
```

ES. 2

$S \to bBa \mid bAc \mid Bc \mid bBa$

$A \to aA \mid \varepsilon$

$B \to bB \mid \varepsilon$

$Null(A) = Null(aA) \vee Null(\varepsilon) = false \vee true = true$

$Null(B) = Null(bB) \vee Null(\varepsilon) = false \vee true = true$

$Null(S) = Null(bAc) \vee Null(Bc) \vee Null(bBa) = false$

$First(A) = \{a, \varepsilon\}$ $First(B) = \{b, \varepsilon\}$

$First(S) = \{a, b, c\}$

$Follow(A) = \{a, c\}$ $Follow(B) = \{a, c\}$ $Follow(S) = \{\$\}$

	a	$S \to bBa$	c	$
S	$S \to Aa$	$S \to bAc$	$S \to Bc$	
A	$A \to aA$		$A \to \varepsilon$	
B	$B \to \varepsilon$	$B \to bB$	$B \to \varepsilon$	

NO LL(1)

$S \to bBa \to bbBa \to bbbBa$

$S \to Bc \to bbBc \to bbbBc$

$S \to bAc \to baAc \to baaAc$

NON ESISTE ALCUN K PER CUI LA GRAMMATICA E' LL(K) perchè posso generare sia 'b' e non saprei quale produzione scegliere

ES. 3

```
gen(stable, for id := E to E' do S) =
        loop  = newLabel();
        end   = newLabel();
        gen(stable, E')
        push $a0
        gen(stable, E)
        lw $al 0($fp)
        for(int i=0; i<nestingLevel - lookup(stable, id).nestingLevel; i++){
            lw $al 0($al)
        }
        sw $a0 lookup(stable, id).offset($al)
loop:   lw $al 0($fp)
        for(int i=0; i<nestingLevel - lookup(stable, id).nestingLevel; i++){
            lw $al 0($al)
        }
        lw $t1 lookup(stable, id).offset($al)  // $t1 = id
        $a0 <- top
        bg $t1 $a0 end
        gen(stable, S)
        move $a0 $t1  // $a0 = id
        push $a0
        li $a0 1
        $t1 <- top    // $t1 = id
        add $a0 $t1 $a0   // id + 1
        pop
        lw $al 0($fp)
        for(int i=0; i<nestingLevel - lookup(stable, id).nestingLevel; i++){
            lw $al 0($al)
        }
        sw $a0 lookup(stable, id).offset($al)  // id = id + 1
        b loop
end : pop
```

```
gen(stable, for x := y to z do z := x+1)
    loop = newLabel();
    end = newLabel();
    lw $al 0($fp)
    lw $al 0($al)
    lw $a0 8($al)   // $a0 = z
    push $a0
    lw $al 0($fp)
    lw $a0 42($al)  // a0 = y
    lw $al 0($fp)
    sw $a0 8($al)
loop: lw $al 0($fp)
    lw $t1 8($fp)
    $a0 <- top
    bg $t1 $a0 end
    gen(stable, z=x+1) -->
    move $a0 $t1
    push $a0
    li $a0 1
    $t1 <- top
    add $a0 $t1 $a0
    pop
    lw $al 0($fp)
    sw $a0 8($al)
    b loop
end: pop
```

```
gen(stable, z=x+1) =
  li $a0 1
  push $a0
  lw $al 0($fp)
  lw $a0 8($al)
  $t1 <- top  // lw $t1 0($sp)
  add $a0 $a0 $t1
  pop
  lw $al 0($fp)
  lw $al 0($al)
  sw $a0 8($al)
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

19 Febbraio 2021

Esercizio 1 (7 punti) Sia L il linguaggio sull'alfabeto $\{a, b, c, d\}$ costituito da stringhe della forma $\alpha d \beta$ dove α è una qualunque stringa non vuota che contiene $\{b, c\}$ e β è una qualunque stringa non vuota che contiene $\{a, c\}$. Si definisca in `ANTLR` l'analizzatore lessicale per tokens in L.

Esercizio 2 (7 punti) Si verifichi che la grammatica

$$\begin{array}{lcl} S & \rightarrow & AaB \mid B \\ A & \rightarrow & bB \mid c \mid \varepsilon \\ B & \rightarrow & aA \end{array}$$

(ε è la stringa vuota). Verificare, scrivendo la tabella relativa, che

- la grammatica è LL(1) ;

Esercizio 3 (1) Definire la funzione `code_gen` che prende in input un termine

$$E \texttt{ \&\& } E'$$

e genera il codice intermedio (una espressione booleana ritorna 0, per falso, o 1, per vero). Il valore di ritorno si trova, come per tutte le espressioni, nel registro `$a0`.

(2) Come verifica, scrivere il codice di

```
(x && y) && z
```

Assumendo che le variabili `x`, `y` e `z` si trovano ad offset `+4`, `+8` del frame pointer `$fp`, mentre la variabile `z` si trova nell'ambiente statico immediatamente esterno all'ambiente corrente e a offset 0 (l'ambiente statico è accessibile attraverso il registro `$a1`).

19-02-2021

giovedì 19 maggio 2022 10:52

ES. 1

```
// Sia L il linguaggio sull'alfabeto {a, b, c, d} costituito da
// stringhe della forma AdB dove A è una qualunque stringa non vuota
// che contiene {b, c} e B è una qualunque stringa non vuota che contiene {a, c}.
// Si definisca in ANTLR l'analizzatore lessicale per tokens in L.

L : A 'd' B;

A : ('b' | 'c')+;

B : ('a' | 'c')+;

WS : (' '|'\t'|'\n'|'\r')-> skip;
```

ES. 2

$S \to A\ a\ B \mid B$

$A \to b\ B \mid c \mid \varepsilon$

$B \to a\ A$

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

$Follow(A) = \{a\} \cup Follow(B) =$

$= \{a\} \cup Follow(S) = \{a, \$\}$

	a	b	c	$
S	S → B S → AaB	S → AaB	S → AaB	S → AaB
A	A → ε	A → bB	A → c	A → ε
B	B → aA			

NO LL(1)

ES. 3

```
gen(stable, E && E') =
    label = newLabel();
    gen(stable, E)
    beq $a0 0 label
    gen(stable, E')
label:

gen(stable, (x && y) && z) =
    label = newLabel();
    gen(stable, x && y) ──→ gen(stable, x && y) =
                                label = newLabel();
                                lw $al 0($fp)
                                lw $a0 4($al)
                                beq $a0 0 label
                                lw $al 0($fp)
                                lw $a0 8($al)
                            label:
    beq $a0 0 label
    lw $al 0($fp)
    lw $al 0($al)
    lw $a0 0($al)
label:
```

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; } }
```

ESAME 28 MAGGIO 2021

ES. 1

→ PARSER

```
init : program ;

program : Dec* Stm* ;

Dec : 'int' ID ';' ;

Stm : ID '=' Exp ';'
    | 'while' '(' Exp ')' '{' Stm* '}' ;

Exp : DIGIT+          # constantExp
    | ID              # varExp
    | left=Exp '+' right=Exp   # sumExp ;
```

All'interno di ANTLR #constantExp è utile per la creazione di una funzione visitConstantExp(...) dentro l'interfaccia AssetLanVisitor al posto della funzione visitExp(...)

All'interno dell'implementazione dell'interfaccia AssetLanVisitor, dentro alla funzione visitSumExp(...) è possibile richiamare la visita su Exp utilizzando il contesto tramite il comando ctx.right

→ LEXER

```
DIGIT : '0'
      | '1'
      :
      | '9' ;

ID : CHAR (CHAR | DIGIT)* ;

CHAR : 'A'
     | 'a'
     :
     | 'z'
     | 'z' ;

WS : (' '
   | '\t'
   | '\n'
   | '\r') -> skip ;

LINE_COMMENTS : '//' (~('\n' | '\r'))* -> skip ;

BLOCK_COMMENTS : '/*' (~('/' | '*') | '/' ~'*' | '*' ~'/' | BLOCK_COMMENTS)*
                 '*/' -> skip ;
```

```
Exp : NUM | ... ;

NUM : DIGIT+ ;

fragment DIGIT : '0'..'9' ;
```

```
fragment CHAR : 'a'..'z' | 'A'..'Z' ;
```

Utilizzato in ANTLR per evitare l'analisi di ogni singolo carattere ma eseguire solo quella sull'intera stringa. In questo modo nell'albero sarà presente un nodo Foglia contenente la stringa e non l'insieme dei nodi Foglia contenenti un singolo carattere

Nelle espressioni regolari indica che non deve contenere il carattere '/'

$$\frac{}{\Gamma, N \vdash \text{NUM} : \text{int}}\ [\text{NUM}]$$

$$\frac{x \notin \text{dom}(\text{top}(\Gamma))}{\Gamma, N \vdash \text{int } x\ ; : \Gamma[x \mapsto (N, \text{int})], N+4}\ [\text{DEC}]$$

in x memorizziamo N e, solo dopo, restituiamo N modificato

$$\frac{\Gamma(x) = \text{int} \quad \Gamma, N \vdash \text{Exp} : \text{int}}{\Gamma, N \vdash x = \text{Exp}\ ; : \text{void}}\ [\text{ASGN}]$$

$$\frac{\Gamma(x) = \text{int}}{\Gamma, N \vdash x : \text{int}}\ [\text{ID}]$$

$$\frac{\Gamma, N \vdash \text{Exp}_1 : \text{int} \quad \Gamma, N \vdash \text{Exp}_2 : \text{int} \quad + : \text{int} \times \text{int} \to \text{int}}{\Gamma, N \vdash \text{Exp}_1 + \text{Exp}_2 : \text{int}}\ [\text{SUM}]$$

$$\frac{\Gamma, N \vdash \text{Exp} : \text{int} \quad \Gamma, N \vdash S : \text{void}}{\Gamma, N \vdash \text{while } (\text{Exp}) \{ S \} : \text{void}}\ [\text{WHILE}]$$

$$\frac{\Gamma, N \vdash d : \Gamma', N' \quad \Gamma', N' \vdash D : \Gamma'', N''}{\Gamma, N \vdash d\ D : \Gamma'', N''}\ [\text{SEQ D}]$$

$$\frac{\Gamma, N \vdash s : \text{void} \quad \Gamma, N \vdash S : \text{void}}{\Gamma, N \vdash s\ S : \text{void}}\ [\text{SEQ S}]$$

$$\frac{\Gamma[], 0 \vdash D : \Gamma', N' \quad \Gamma', N' \vdash S : \text{void}}{\Gamma[], 0 \vdash D\ S}\ [\text{PROG}]$$

NB: in questo esercizio di typecheck astratto le regole possono restituire tre diversi valori: ambiente + offset, tipo, void.
A livello implementativo per quanto riguarda l'ambiente viene modificato all'interno del corpo della funzione ma il valore restituito è nulla (void). Questo perché nella dichiarazione della funzione typecheck il tipo è ON type (che può essere effettivamente un tipo o null)

NB: la codegen delle dichiarazioni non esiste

cgen(stable, NULL) = li $a0 NULL

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

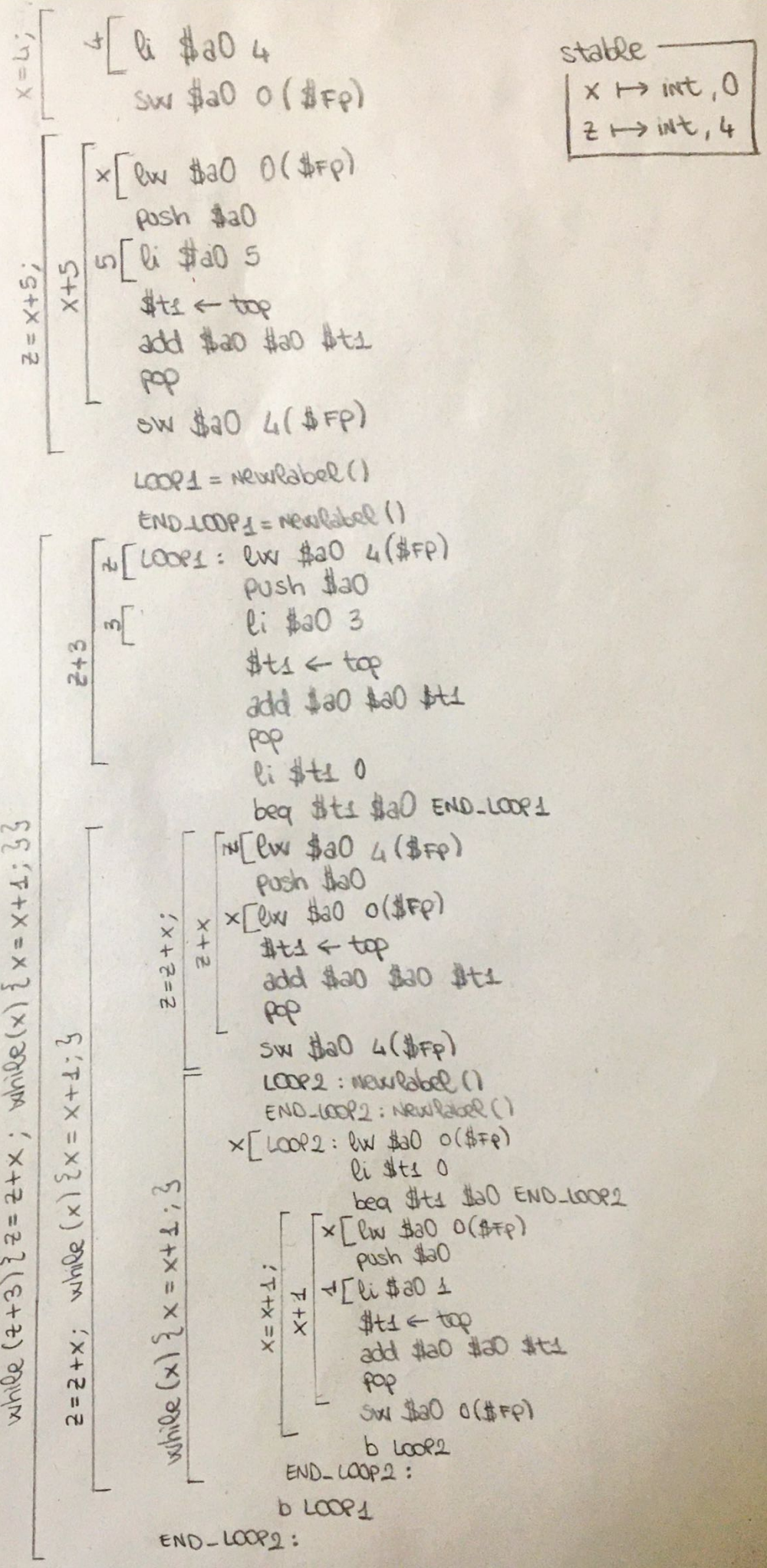
cgen(stable, e1 + e2) = cgen(stable, e1)
push $a0
cgen(stable, e2)
$t1 ← top
add $a0 $a0 $t1
pop

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

cgen(stable, while (e) {S}) = LOOP = newlabel()
END_LOOP = newlabel()
LOOP: cgen(stable, e)
li $t1 0
beq $t1 $a0 END_LOOP
cgen(stable, S)
b LOOP
END_LOOP:

cgen(stable, S S) = cgen(stable, S)
cgen(stable, S)

cgen(stable, D S) = cgen(stable, S)


stable
x ↦ int, 0
z ↦ int, 4
x=4;
4
li $a0 4
sw $a0 0($fp)
z=x+5;
x+5
x
lw $a0 0($fp)
push $a0
5
li $a0 5
$t1 ← top
add $a0 $a0 $t1
pop
sw $a0 4($fp)
LOOP1 = newlabel()
END_LOOP1 = newlabel()
while(z+3){z=z+x; while(x){x=x+1;}}
z+3
z
LOOP1: lw $a0 4($fp)
push $a0
3
li $a0 3
$t1 ← top
add $a0 $a0 $t1
pop
li $t1 0
beq $t1 $a0 END_LOOP1
z=z+x;
z+x
N
lw $a0 4($fp)
push $a0
x
lw $a0 0($fp)
$t1 ← top
add $a0 $a0 $t1
pop
sw $a0 4($fp)
LOOP2 : newlabel()
END_LOOP2 : newlabel()
z=x+1;
while(x){x=x+1;}
x
LOOP2: lw $a0 0($fp)
li $t1 0
beq $t1 $a0 END_LOOP2
x=x+1;
x+1
x
lw $a0 0($fp)
push $a0
1
li $a0 1
$t1 ← top
add $a0 $a0 $t1
pop
sw $a0 0($fp)
b LOOP2
END_LOOP2:
b LOOP1
END_LOOP1:

28-05-2021

giovedì 19 maggio 2022 11:25

ES. 1

```
grammar BOTELLA;

/*
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.
*/

block : '{' dec=dec* stm=stm* '}';

dec : "int" ID ";";

stm : ID "=" exp ";"
    | "while" '(' exp ')' block;

exp : NUMBER
    | ID
    | exp '+' exp;

fragment CHAR : 'a'..'z' | 'A'..'Z';
ID : CHAR (CHAR | NUMBER)+;

fragment DIGIT : '0'..'9';
NUMBER : DIGIT+;

WS              : (' '|'\t'|'\n'|'\r')+ -> skip;
```

ES. 2

NON DICHIARATI O DICHIARAZIONI MULTIPLE

$$\frac{\Gamma,0 \vdash dec : \Gamma_1,m \quad \Gamma_1,m \vdash stm : \Gamma,m}{\Gamma,0 \vdash \{dec^*\ stm^*\} : void}\ [block] \qquad \frac{ID \notin dom(top(\Gamma))}{\Gamma,m \vdash int\ ID : \Gamma[ID \mapsto int],m+4}\ [dec]$$

$$\frac{\Gamma,m \vdash d : \Gamma',m' \quad \Gamma',m' \vdash D : \Gamma'',m''}{\Gamma,m \vdash d\ D : \Gamma'',m''}\ [seq\text{-}D] \qquad \frac{}{\Gamma,m \vdash NUMBER : int}\ [exp]$$

$$\frac{\Gamma,m \vdash s : void \quad \Gamma,m \vdash S : void}{\Gamma,m \vdash s\ S : void}\ [seq\text{-}S] \qquad \frac{ID \in dom(\Gamma) \quad \Gamma(ID) = T \quad T = int}{\Gamma,m \vdash ID : T}\ [ID]$$

$$\frac{\Gamma,m \vdash exp : T \quad \Gamma(ID) = T_1 \quad T_1 : T}{\Gamma,m \vdash ID = exp\ ; : void}\ [Stm]$$

$$+ : int \times int \to int$$

$$\frac{\Gamma,m \vdash exp_1 : T_1 \quad \Gamma,m \vdash exp_2 : T_2 \quad T_1 = T_2 = int}{\Gamma,m \vdash exp_1 + exp_2 : T_1}\ [sum]$$

$$\frac{\Gamma,m \vdash exp : int \quad \Gamma,m \vdash block : void}{\Gamma,m \vdash while\ (exp)\ block : void}\ [stm\text{-}2]$$

ES. 3

block : { Dec* Stm* }

Dec : int ID ;

Stm : ID = exp ; | while (exp) block

exp : Number | ID | exp + exp ;

```
gen(stabb, {dec* stm*}):
    gen(stabb', stm)   // stabb ⊢ dec : stabb'
```

```
gen(stabb, ID = exp)=
    gen(stabb, exp)
    lw $al 0($fp)
    for(int i=0; i<nesting_level - lookup(stabb, ID).nesting_level; i++){
    lw $al 0($al)
    }
    sw $a0 lookup(stabb, ID).offset($al)
```

```
gen(stabb, while (exp) block) =
    loop = newLabel();
    end = newLabel();
loop: gen(stabb, exp)
    li $t1
    beq $a0 $t1 end
    gen(stabb, block)
    b loop
end:
```

```
gen(stabb, NUMBER) =
    li $a0 NUMBER
```

```
gen(stabb, exp1 + exp2) =
    gen(stabb, exp1)
    push $a0
    gen(stabb, exp2)
    $t1 <- top
    add $a0 $a0 $t1
    pop
```

```
gen(stabb, ID) =
    lw $al 0($fp)
    for(int i=0; i<nesting_level - lookup(stabb, ID).nesting_level; i++){
    lw $al 0($al)
    }
    lw $a0 lookup(stabb, ID).offset($al)
```

```
gen(stabb, int x; int z; x=4; z=x+5; while(z+3){z=z+x; while(x){x=x+z;}})
    loop = newLabel();
    end = newLabel();
    li $t1 4
    lw $al 0($fp)
    sw $t1 0($al)
    lw $al 0($fp)
    lw $a0 0($al)   // a0 = x
    push $a0
    li $t1 5
    $a0 <- top
    pop
    add $a0 $a0 $t1   // x+5
    lw $al 0($fp)
    sw $a0 4($al)   // z = x+5
    lw $al 0($fp)
    lw $a0 4($al)   // a0 = z
    push $a0
    li $t1 3
    $a0 <- top
    pop
loop: add $a0 $a0 $t1   // z+3
    li $t1 0
    beq $a0 $t1 end
    lw $al 0($fp)
    lw $a0 0($al)   // a0 = x
    push $a0
    lw $al 0($fp)
    lw $t1 4($al)
    $a0 <- top
    pop
    add $a0 $a0 $t1   // z+x
    lw $al 0($fp)
    sw $a0 4($al)   // z = z+x
    gen(stabb, while(x){x=x+1;}) ->
    b loop
end:
```

STABLE
x ↦ int, 0
z ↦ int, 4

```
gen(stabb, while(x){x=x+1;}) =
    loop = newLabel();
    end = newLabel();
    lw $al 0($fp)
loop: lw $a0 0($al)
    li $t1 0
    beq $a0 $t1 end
    lw $al 0($fp)
    lw $a0 0($al)
    push $a0
    li $t1 1
    $a0 <- top
    pop
    add $a0 $a0 $t1
    lw $al 0($fp)
    sw $a0 0($al)
    b loop
end:
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

17 Settembre 2021

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 : stm ;
stm : (Id '=' exp ';' | '{' dec stm '}')+ ;
dec : (type Id ';')+ ;
type: 'int' | 'bool' ;
exp : Int | Bool | Id | exp '+' exp | exp '-' exp | exp '>' exp | exp '==' exp
    | exp '||' exp | exp '&&' exp ;
```

dove

- gli `Int` sono sequenze non vuote di cifre senza segno oppure prefissate dal segno `+` o `-`;
- i `Bool` sono i valori "`true`" e "`false`";
- gli `Id` sono gli identificatori (sequenze non vuote di caratteri);
- le operazioni di somma "`+`", sottrazione "`-`" e maggiore "`>`" si applicano a espressioni `Int`, le operazioni di or "`||`" e and "`&&`" si applicano a espressioni `Bool`, mentre l'operazione di uguaglianza si applica a espressioni dello stesso tipo.

Esercizi

1. dare tutte le regole di inferenza per la verifica dei tipi del linguaggio di sopra (attenzione che il linguaggio ammette annidamento di ambienti);
2. verificare, scrivendo l'albero di prova, che il programma seguente sia correttamente tipato:

   `{ int x; int y; x = 5; { bool z; z = (x > 5)||false; { int z; y = 6+x; z = 3 + y; }}}`
3. definire il codice intermedio per tutti i costrutti del linguaggio. In particolare, si ricordi che un booleano occupa 1 byte mentre un intero occupa 4 byte. Inoltre, per quanto riguarda le operazioni `||` e `&&`, si implementi la cosiddetta *lazy evaluation*: il secondo argomento non viene valutato se inutile (nell' `||`, il secondo argomento non viene valutato se il primo è `true`, nell' `&&` il secondo argomento non viene valutato se il primo è `false`).
4. scrivere il codice generato per l'esempio al punto 2.

17-09-2021

giovedì 19 maggio 2022 16:29

```
prg : stm ;
stm : (Id '=' exp ';' | '{' dec stm '}')+ ;
dec : (type Id ';')+ ;
type: 'int' | 'bool' ;
exp : Int | Bool | Id | exp '+' exp | exp '-' exp | exp '>' exp | exp '==' exp
    | exp '||' exp | exp '&&' exp ;
```

ES. 1

$$\frac{\Gamma(ID) = T \quad \Gamma \vdash exp : T' \quad T = T'}{\Gamma \vdash ID = exp\ ; : void}\ [stm\ 1]$$

$$\frac{\Gamma \cdot [\,] \vdash dec : \Gamma' \qquad \Gamma' \vdash stm : \Gamma''}{\Gamma \vdash \{ dec\ stm \} : \Gamma''}\ [stm\ 2]$$

$$\frac{ID \notin dom(\Gamma)}{\Gamma \vdash type\ ID}\ [dec] \qquad \frac{\Gamma \vdash exp_1 : T_1 \quad \Gamma \vdash exp_2 : \overbrace{T_2 \quad (-|+) : Int \times Int \to Int}^{T_1 = T_2 = Int}}{\Gamma \vdash exp_1 (+|-) exp_2 : Int}$$

$$\frac{\Gamma \vdash exp_1 : T_1 \quad \Gamma \vdash exp_2 : \overbrace{T_2 \quad > : Int \times Int \to Bool}^{T_1 = T_2 = Int}}{\Gamma \vdash exp_1 > exp_2 : Bool}$$

$$\frac{\Gamma \vdash exp_1 : T_1 \quad \Gamma \vdash exp_2 : \overbrace{T_2 \quad == : T_1 \times T_2 \to Bool}^{T_1 = T_2}}{\Gamma \vdash exp_1 == exp_2 : Bool}$$

$$\frac{\Gamma \vdash exp_1 : T_1 \quad \Gamma \vdash exp_2 : \overbrace{T_2 \quad \&\&|\| : Bool \times Bool \to Bool}^{T_1 = T_2 = Bool}}{\Gamma \vdash exp_1 (\&\&|\|) exp_2 : Bool}$$

ES. 2

$$\frac{z \in dom(\Gamma''') \quad \Gamma'''(x) = T_1 \quad 5 : T_2 \quad T_1 = T_2 = Int}{\Gamma'''(z) : bool \quad \Gamma''' \vdash (x > 5) : bool \quad false : bool \quad \| : bool \times bool \to bool}{z = (x > 5) \| false : void}$$

$$\frac{\Gamma_4(y) = T_6 \quad \Gamma_4 + 6 : T_5 \quad \Gamma_4(x) = T_6 \quad T_6 = T_5 = T_6 = int}{\Gamma_4 \vdash y = 6 + x : void}$$

$$\frac{\Gamma \cdot [\,] \vdash int\ x : \Gamma' \quad \overbrace{\Gamma' \vdash int\ y : \Gamma''}^{y \notin dom(\Gamma')} \quad \Gamma''(x) = T = Int = 5 \quad \Gamma'' \cdot [\,] \vdash bool\ z : \overbrace{\Gamma'' \cdot [z \mapsto bool]}^{\Gamma'''} \quad \Gamma''' \cdot [\,] \vdash int\ z : \overbrace{\Gamma''' \cdot [z \mapsto int]}^{\Gamma_4} \quad \frac{\Gamma_4(z) = T_6 \quad \Gamma_4 + 3 = T_7 \quad \Gamma_4(y) = T_8 \quad T_6 = T_7 = T_8 = int}{\Gamma_4 \vdash z = 3 + y : void}}{\Gamma \cdot [\,] \vdash \{ int\ x;\ int\ y;\ x = 5;\ \{ bool\ z;\ z = (x > 5) \| false;\ \{ int\ z;\ y = 6 + x;\ z = 3 + y; \} \} \}}$$

ES. 3

```
prg : stm ;
stm : (ID = exp ; | { dec stm })+ ;
dec : (type ID ;)+ ;
type : int | bool ;
exp : Int | Bool | ID | exp + exp | exp - exp | exp > exp
      | exp == exp | exp || exp | exp && exp

gen(stable, prg) =
    gen(stable, stm)

gen(stable, ID = exp) =
    gen(exp)
    lw $a1 0($fp)
    for (i=0; i < nestinglevel - lookup(stable, ID).nestinglevel; i++){
        lw $a1 0($a1)
    }
    sw $a0 lookup(stable, ID).offset($a1)
gen(stable, { dec stm }) =
    gen(stable', stm)   // stable ⊢ dec : stable'

gen(stable, Int) =          gen(stable, Bool) =
    lw $a0 Int                  lw $a0 Bool

gen(stable, ID) =
  lw $a1 0($fp)
  for (i=0; i < nestinglevel - lookup(stable, ID).nestinglevel; i++){
    lw $a1 0($a1)
  }
  lw $a0 lookup(stable, ID).offset($a1)
```

```
gen(stable, exp1 + exp2) =
    gen(stable, exp1)
    push $a0
    gen(stable, exp2)
    $t1 <- top
    add $a0 $t1 $a0
    pop
```

```
gen(stable, exp1 - exp2) =
    gen(stable, exp1)
    push $a0
    gen(stable, exp2)
    $t1 <- top
    sub $a0 $t1 $a0
    pop
```

```
gen(stable, exp1 > exp2)
    label = newLabel();
    gen(stable, exp1)
    push $a0
    gen(stable, exp2)
    $t1 <- top
    bleq $t1 $a0 label
    li $a0 1
label: li $a0 0
```

```
gen(stable, exp1 == exp2)
    label = newLabel();
    gen(stable, exp1)
    push $a0
    gen(stable, exp2)
    $t1 <- top
    beq $t1 $a0 label
    li $a0 0
label: li $a0 1
```

```
gen(stable, exp1 || exp2) =
    label = newLabel();
    gen(stable, exp1)
    li $t1 1
    beq $a0 $t1 label
    gen(stable, exp2)
label:
```

```
gen(stable, exp1 && exp2) =
    label = newLabel();
    gen(stable, exp1)
    li $t1 0
    beq $a0 $t1 label
    gen(stable, exp2)
label:
```

```
gen (stable, {dec stm}) =
    foreac d in dec {
      if (lookup(stable, d).type()) == 'int') {
        li $t1 4
        add $fp $sp $t1
    }else{
        li $t1 1
        add $fp $sp $t1
    }
  }
  gen(stable', stm) // stable ⊢ Dec : stable'
  lw $fp 0($fp)
```

ESAME 17 SETTEMBRE 2021

ES 1

$$\frac{}{\Gamma \vdash \text{NUM} : \text{int}}\ [\text{NUM}] \qquad \frac{}{\Gamma \vdash \text{bool} : \text{bool}}\ [\text{BOOL}] \qquad \frac{\Gamma(\text{id}) = T}{\Gamma \vdash \text{id} : T}\ [\text{ID}]$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \pm e_2 : \text{int}}\ [\text{OPA}] \qquad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 > e_2 : \text{bool}}\ [\text{MAG}] \qquad \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \,\&\&\, e_2 : \text{bool}}\ [\text{AND}]$$

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2 \quad T_1 = T_2}{\Gamma \vdash e_1 == e_2 : \text{bool}}\ [\text{UG}] \qquad \frac{\text{id} \notin \text{dom}(\text{top}(\Gamma))}{\Gamma \vdash T\ \text{id};\ : \Gamma[\text{id} \mapsto T]}\ [\text{DEC}]$$

$$\frac{\Gamma \vdash d : \Gamma' \quad \Gamma' \vdash D : \Gamma''}{\Gamma \vdash d\,D : \Gamma''}\ [\text{SEQ D}] \qquad \frac{\Gamma \vdash e : T \quad \Gamma(\text{id}) = T' \quad T = T'}{\Gamma \vdash \text{id} = e\,; : \text{void}}\ [\text{ASGN}]$$

$$\frac{\Gamma \cup [] \vdash D : \Gamma' \quad \Gamma' \vdash S : \text{void}}{\Gamma \vdash \{D\ S\} : \text{void}}\ [\text{PROG}] \qquad \frac{\Gamma \vdash s : \text{void} \quad \Gamma \vdash S : \text{void}}{\Gamma \vdash s\,S : \text{void}}\ [\text{SEQ S}] \qquad \frac{\Gamma \vdash \text{exp} : T}{\Gamma \vdash (\text{exp}) : T}\ [\text{EXP}]$$

$\Gamma_1 : [x \mapsto int, y \mapsto int]$

$\Gamma_2 : [x \mapsto int]$

$\Gamma_3 : [x \mapsto int, y \mapsto int] \cdot [z \mapsto bool]$

$\Gamma_4 : [x \mapsto int, y \mapsto int] \cdot [z \mapsto bool] \cdot [z \mapsto int]$

$$\dfrac{\dfrac{\dfrac{}{\Gamma_4 \vdash 6 : int}[NUM] \quad \dfrac{\Gamma_4(x) = int}{\Gamma_4 \vdash x : int}[ID]}{\Gamma_4 \vdash 6+x : int}[SUM] \quad \dfrac{\Gamma_4(y) = int \quad int = int}{\Gamma_4 \vdash y = 6+x; : void}[ASGN]}{\Gamma_4 \vdash y = 6+x; : void \quad \Gamma_4 \vdash z = 3+y; : void}[ASGN]$$

$$[SEQ\ S]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_3(x) = int}{\Gamma_3 \vdash x : int}[ID] \quad \dfrac{}{\Gamma_3 \vdash 5 : int}[NUM]}{\Gamma_3 \vdash (x>5) : bool}[BAG] \quad \dfrac{}{\Gamma_3 \vdash false : bool}[BOOL]}{\Gamma_3 \vdash (x>5) \,||\, false : bool}[AND] \quad \dfrac{\Gamma_3(z) = bool \quad bool = bool}{\Gamma_3 \vdash z = (x>5) \,||\, false; : void}[ASGN]}{\Gamma_3 \vdash \{int\ z; y = 6+x; z = 3+y; \} : void}[SEQ\ S]$$

$$\dfrac{z \notin dom(top(\Gamma_3[\,])) \quad \Gamma_3 : [\,] \cdot [int\ z;] : \Gamma_4 \quad \Gamma_4 \vdash y = 6+x; z = 3+y; : void}{\Gamma_3 \vdash \{int\ z; y = 6+x; z = 3+y; \} : void}[DEC]\ [PROG]$$

$$\dfrac{\dfrac{x \notin dom(top(\Gamma_0)) \quad \Gamma_0 \cdot [\,] \vdash int\ x; : \Gamma_2 \quad y \notin dom(top(\Gamma_2)) \quad \Gamma_2 \vdash int\ y; : \Gamma_1}{\Gamma_0 \cdot [\,] \vdash int\ x; int\ y; : \Gamma_1}[SEQ\ D] \quad \dfrac{\dfrac{}{\Gamma_1 \vdash 5 : int} \quad \Gamma(x) : int \quad int = int}{\Gamma_1 \vdash x = 5; : void}[ASGN] \quad \dfrac{z \notin dom(top(\Gamma_1[\,])) \quad \Gamma_1 \cdot [\,] \vdash bool\ z; : \Gamma_3 \quad \Gamma_3 \vdash z = (x>5) \,||\, false; \{int\ z; y = 6+x; z = 3+y; \} : void}{\Gamma_1 \vdash \{bool\ z; z = (x>5) \,||\, false; \{int\ z; y = 6+x; z = 3+y; \}\} : void}[DEC]\ [PROG]}{\Gamma_1 \vdash x = 5; \{bool\ z; z = (x>5) \,||\, false; \{int\ z; y = 6+x; z = 3+y; \}\} : void}[SEQ\ S]$$

$$\Gamma \vdash \{int\ x; int\ y; x = 5; \{bool\ z; z = (x>5) \,||\, false; \{int\ z; y = 6+x; z = 3+y; \}\}\} : void \quad [PROG]$$

```
codegen(stable, N) = li $a0 N

codegen(stable, true) = li $a0 1

codegen(stable, false) = li $a0 0

codegen(stable, x) = move $al $FP
                     for (i=0; i<nesting_level - lookup(stable,x).nesting_level; i++)
                        lw $al 0($al)
                     lw $a0 lookup(stable,x).offset($al)

codegen(stable, e1+e2) = cgen(stable, e1)
                         push $a0
                         cgen(stable, e2)
                         $t1 ← top
                         add $a0 $t1 $a0   → sub $a0 $t1 $a0 con '-'
                         pop

cgen(stable, e1>e2) = cgen(stable, e1)
                      push $a0
                      cgen(stable, e2)
                      $t1 ← top
                      MAG = newlabel()
                      END = newlabel()
                      bgt $t1 $a0 MAG
                      cgen(stable, false)
                      b END
                      MAG: cgen(stable, true)
                      END: pop

cgen(stable, e1==e2) = cgen(stable, e1)
                       push $a0
                       cgen(stable, e2)
                       $t1 ← top
                       EQ = newlabel()
                       END = newlabel()
                       beq $t1 $a0 EQ
                       cgen(stable, false)
                       b END
                       EQ: cgen(stable, true)
                       END: pop
```

```
cgen(stable, e1 || e2) = cgen(stable, e1)
                         LAZY = newlabel()
                         END = newlabel()
                         li $t1 1
                         beq $a0 $t1 LAZY
                         cgen(stable, e2)
                         li $t1 1
                         beq $a0 $t1 LAZY
                         cgen(stable, false)
                         b END
                         LAZY : cgen(stable, true)
                         END:

cgen(stable, e1 && e2) = cgen(stable, e1)
                         LAZY = newlabel()
                         END = newlabel()
                         li $t1 0
                         beq $a0 $t1 LAZY
                         cgen(stable, e2)
                         li $t1 0
                         beq $a0 $t1 LAZY
                         cgen(stable, true)
                         b END
                         LAZY : cgen(stable, false)
                         END:

cgen(stable, (e)) = cgen(stable, e)
cgen(stable, x=e) = cgen(stable, e)
                    move $al $FP
                    for(i=0; i<nesting_level - lookup(stable, x).nesting_level; i++)
                        lw $al 0($al)
                    sw $a0 lookup(stable, x).offset($al)

cgen(stable, {D S}) = cgen(stable, S)
cgen(stable, s S) = cgen(stable, s)
                    cgen(stable, S)
```

ES 4

stable

x	↦	int	4, 0
y	↦	int	8, 0
z	↦	bool	1, 1
z	↦	int	4, 2

```
{int x; int y; x=5; {bool z; z=(x>5)||false; {int z; y=6+x; z=3+y;}3}
x=5;                                                                  li $a0 5
                                                                      move $al $fp
                                                                      sw $a0 4($al)
{bool z; z=(x>5)||false; {int z; y=6+x; z=3+y;}3}
z=(x>5)||false; {int z; y=6+x; z=3+y;}                                move $al $fp
                                                                      lw $al 0($al)
                                                                      lw $a0 4($al)
                                                                      push $a0
(x>5)||false                                                          li $a0 5
                                                                      $t1 ← top
                                                                      MAG1 = newlabel()
                                                                      END1 = newlabel()
                                                                      bgt $t1 $a0 MAG1
                                                                      li $a0 0
                                                                      b END1
                                                                      MAG1: li $a0 1
                                                                      END1: pop
                                                                      LAZ42: newlabel()
                                                                      END2: newlabel()
                                                                      li $t1 1
                                                                      beq $a0 $t1 LAZ42
                                                                      li $a0 0
                                                                      li $t1 1
                                                                      beq $a0 $t1 LAZ42
                                                                      li $a0 0
                                                                      b END2
                                                                      LAZ42: li $a0 1
                                                                      END2:
                                                                      move $al $fp

                                                                      sw $a0 1($al)
{int z; y=6+x; z=3+y;}3                                               li $a0 6
                                                                      push $a0
                                                                      move $al $fp
                                                                      lw $al 0($al)
                                                                      lw $al 0($al)
                                                                      lw $a0 4($al)
                                                                      $t1 ← top
                                                                      add $a0 $t1 $a0
                                                                      pop
                                                                      move $al $fp
                                                                      sw $a0 8($al)
y=6+x; z=3+y;                                                         li $a0 3
                                                                      push $a0
                                                                      move $al $fp
                                                                      lw $a0 8($al)
                                                                      $t1 ← top
                                                                      add $a0 $t1 $a0
                                                                      pop
                                                                      move $al $fp
                                                                      sw $a0 4($al)
```

```
lw $al 0($al)
lw $al 0($al)

lw $al 0($al)
lw $al 0($al)
```

Corso di Laurea Magistrale in Informatica

Compito di Compilatori e Interpreti

17 Febbraio 2022

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` ed a `adele.veschetti2@unibo.it`.

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

- `Dec` sono sequenze di dichiarazioni di identificatori interi (`int`) con inizializzazione (tipo `int x = E`);
- `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),
   - per verificare *se un identificatore cambia valore* (all'interno di `Stm` c'e' un assegnamento con l'identificatore che compare a sinistra)
   - 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 = 4; int z = x+5;
while (z + 3){ z = z+x ; while (x){ x = x+1; } }
```

ESAME 17 FEBBRAIO 2022

ES 1

```
INIT : program
program : Dec Stm ;
Dec : ('int' ID '=' Exp ';')* ;
Stm : (ID '=' Exp ';'
     |'while' '(' Exp ')' '{' Stm '}' )* ;

Exp : NUM
    | ID
    | Exp '+' Exp ;

NUM : DIGIT+ ;
ID : CHAR+ ( CHAR | DIGIT )* ;
fragment DIGIT : '0'..'9' ;
fragment CHAR : 'a'..'z' | 'A'..'Z' ;

WS : (' '|'\n'|'\t'|'\r') -> skip ;
```

ES 2

$\beta = \{0, 1\}$ con 0 (valore assegnato), 1 (valore calcolato)

$\Gamma : id \to (type, offset, effect)$

$$\frac{}{\Gamma, N \vdash NUM : int}\ [NUM] \qquad \frac{\Gamma(x) = (int, N', \beta)}{\Gamma, N \vdash x : int}\ [ID] \qquad \frac{\Gamma, N \vdash e_1 : int \quad \Gamma, N \vdash e_2 : int \quad + : int \times int \to int}{\Gamma, N \vdash e_1 + e_2 : int}\ [SUM]$$

$$\frac{\Gamma, N \vdash e : int \quad \Gamma(x) : (int, N', \beta)}{\Gamma, N \vdash x = e ; : \Gamma[x \mapsto (int, N', 1)]}\ [ASGN] \qquad \frac{\Gamma, N \vdash e : int \quad \Gamma \cdot [\,], N \vdash S : \Gamma'}{\Gamma, N \vdash while\,(e)\,\{S\} : \Gamma'}\ [WHILE] \qquad \frac{\Gamma, N \vdash s : \Gamma' \quad \Gamma, N \vdash S : \Gamma''}{\Gamma, N \vdash s\,S : \Gamma''}\ [SEQ S]$$

$$\frac{\Gamma, N \vdash e : int \quad x \notin dom(top(\Gamma))}{\Gamma, N \vdash int\ x = e ; : \Gamma[x \mapsto (int, N, \emptyset)], N+4}\ [DEC] \qquad \frac{\Gamma, N \vdash d : \Gamma', N' \quad \Gamma', N' \vdash D : \Gamma'', N''}{\Gamma, N \vdash d\,D : \Gamma'', N''}\ [SEQ D]$$

$$\frac{\Gamma \cdot [\,], 0 \vdash D : \Gamma', N \quad \Gamma', N \vdash S : \Gamma''}{\Gamma, 0 \vdash D\,S : \Gamma''}\ [PROG]$$

ES 3

cgen(stable, N) = li $a0 N

cgen(stable, x) = move $al $FP
for(i=0; i<nesting_level - lookup(stable,x).nesting_level; i++)
lw $al 0($al)
lw $a0 lookup(stable,x).offset($al)

↳ questo ciclo FOR serve all'interno del nuovo AR del while

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

cgen(stable, x = e) = cgen(stable, e)
move $al $FP
for(i=0; i<nesting_level - lookup(stable, x).nesting_level; i++)
lw $al 0($al)
sw $a0 lookup(stable, x).offset($al)

cgen(stable, while (e) {S}) =
Loop = newlabel()
END = new_label()
LOOP: cgen(stable, e)
li $t1 0
beq $a0 $t1 END
cgen(stable, S)
b Loop
END:

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

Di solito la cgen delle dichiarazioni non viene fatta, ma in questo caso viene scritta perché è presente anche un assegnamento

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

cgen(stable, d D) = cgen(stable, d)
cgen(stable, D)

cgen(stable, D S) = cgen(stable, D)
cgen(stable, S)

stable

x ↦ int, 4, 0
z ↦ int, 8, 0

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

int x=4;
    4 [ li $a0 4
        sw $a0 4($fp)
int z=x+5;
    x [ move $al $fp
        lw $a0 4($al)
        push $a0
    5 [ li $a0 5
        $t1 ← top
        add $a0 $a0 $t1
        pop
        sw $a0 8($fp)
while(z+3){z=z+x; while(x){x=x+1;}}
LOOP1: move $al $fp
    z [ lw $a0 8($al)
        push $a0
    3 [ li $a0 3
        $t1 ← top
        add $a0 $a0 $t1
        pop
        li $t1 0
        beq $a0 $t1 END1
z=z+x;
        move $al $fp
    z [ lw $al 0($al)
        lw $a0 8($al)
        push $a0
        move $al $fp
    x [ lw $al 0($al)
        lw $a0 4($al)
        $t1 ← top
        add $a0 $a0 $t1
        pop
        move $al $fp
        lw $al 0($al)
        sw $a0 8($al)
while(x){x=x+1;}
LOOP2: move $al $fp
    x [ lw $al 0($al)
        lw $a0 4($al)
        li $t1 0
        beq $a0 $t1 END2
x=x+1;
        move $al $fp
    x [ lw $al 0($al)
        lw $al 0($al)
        lw $a0 4($al)
        push $a0
    1 [ li $a0 1
        $t1 ← top
        add $a0 $a0 $t1
        pop
        move $al $fp
        lw $al 0($al)
        lw $al 0($al)
        sw $a0 4($al)
        b LOOP2
END2:   b LOOP1
END1:
```

ESAME
17/02/2022

(1)

```
prg : { Dec Stm } ;
Dec : ('int' Id ';' E)* ;
Stm : ( Id '=' Exp | while '(' E ')' '{' Stm '}' )* ;
Exp : E | Id | Exp '+' Exp ;

E : DIGIT+ ;
Id : LETTER+ ;
DIGIT : '0' .. '9' ;
LETTER : 'a' .. 'z' | 'A' .. 'Z' ;
WS : (' ' | '\t' | '\n' | '\r') -> skip ;
LINECOMMENT : '//' (~('\n' | '\r'))* -> skip ;
BLOCKCOMMENT : '/*' (~('/' | '*') | '/' ~'*' | '*' ~'/' | BLOCKCOMMENT)* '*/' -> skip
```

(2)

$$\frac{\Gamma \vdash d : \Gamma' \quad \Gamma' \vdash D : \Gamma''}{\Gamma \vdash d\ D : \Gamma''}\ [\text{seqD}] \qquad \frac{\Gamma \vdash s : \text{void} \quad \Gamma \vdash S : \text{void}}{\Gamma \vdash s\ S : \text{void}}\ [\text{seqS}]$$

$$\frac{\Gamma \vdash e : T' \quad T = T' \quad \begin{array}{l} \text{offset} = \Gamma.\text{length} \\ x \notin \text{dom}(\text{top}(\Gamma)) \end{array}}{\Gamma \vdash T\ x = e ; : \Gamma[x \mapsto \text{offset}]}\ [\text{varD}]$$

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2 \quad T_1 = \text{int} = T_2 \quad + : \text{int} \times \text{int} \to \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}\ [\text{Plus}]$$

$$\frac{\Gamma(\text{id}) = T}{\Gamma(\text{id}) : T}\ [\text{Var}] \qquad \frac{}{\Gamma \vdash \text{num} : \text{int}}\ [\text{Num}] \qquad \frac{\Gamma \vdash E : T_1 \quad T_1 = \text{int} \quad \Gamma \vdash \text{Stm} : T_2 \quad T_2 : \text{void}}{\Gamma \vdash \text{while } E \text{ } \{\text{Stm}\} : \text{void}}\ [\text{while}]$$

$$\frac{\Gamma \vdash \text{Dec} : \Gamma' \quad \Gamma' \vdash \text{Stm} : \text{void}}{\Gamma \vdash \{\text{Dec Stm}\} : \text{void}}\ [\text{Block}]$$

(3) cgen(table) ($e_1 + e_2$) = cgen(table) (e_1)
push $t0
cgen(table) (e_2)
$t1 ← top
add $t0 $t0 $t1
pop

cgen(table, while (e) {stm}):
start_while:
cgen (e)
push $t0
beq $t0 0 end_label
cgen (stm)
subi $t0 $t0 1
b start_while
end_label:

cgen(table, $x = e$) = cgen(table, e)
sw $t0 (offset($sp))
Lookup(table, x).offset

cgen(table, x) = lw $t0 offset($x)($sp)

cgen(table, n) = li $t0 $n

cgen(table, Decs) =
subi $sp $sp (4 * table.height)
$fp ← $sp

Variabili e codice:

int x = 4; int z = x+5;

while (z+3) { z = z+x ; while (x) { x = x+z; } }

```
li $a0 4
sw $a0 lookup(stable, x).offset($fp)
lw $a0 lookup(stable, x).offset($fp)
push $a0
li $a0 5
$t1 <- top
add $a0 $a0 $t1
pop
start_while:
lw $a0 lookup(stable, z).offset($fp)
push $a0
li $a0 3
$t1 <- top
add $a0 $a0 $t1
pop
push $a0
beq $a0 0 end_label
lw $a0 lookup(stable, z).offset($fp)
push $a0
lw $a0 lookup(stable, x).offset($fp)
$t1 <- top
add $a0 $a0 $t1
pop
sw $a0 lookup(stable, z).offset($fp)
start_while_2:
lw $a0 lookup(stable, x).offset($fp)
push $a0 0 end_label_2
lw $a0 lookup(stable, x).offset($fp)
push $a0
li $a0 1
add $a0 $a0 1
```

pop
sw $a0, lookup(symbl, x) offset($fp)

subi $a0, $a0, 1
b start_while_2
end_label_2:
subi $a0, $a0, 1
b start_while
end_label: