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)

$$
\begin{array}{rlcl}
S \rightarrow & Ab & \mid & Bc \\
A \rightarrow & aA & \mid & \varepsilon \\
B \rightarrow & acB & \mid & \varepsilon
\end{array}
$$

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 1

$S \to Ab \mid Bc$

$A \to aA \mid \varepsilon$

$B \to acB \mid \varepsilon$

Una grammatica è LL(1) quando per ogni NON TERMINALE e TOKEN in input ci può essere al massimo una (1) produzione.

La grammatica deve essere FATTORIZZATA → questa è già fattorizzata → una grammatica è fattorizzata quando non una regola non comincia producendo lo stesso non terminale più volte

NULLABLE : se esiste una derivazione che mi porta ad avere ε

$$\text{NULLABLE}(\gamma) = \begin{cases} T & \text{se } \gamma \Rightarrow^* \varepsilon \\ F & \text{altrimenti} \end{cases}$$

$\text{NULL}(S) = \text{NULL}(Ab) \vee \text{NULL}(Bc) = F \vee F = F$

$\text{NULL}(Ab) = \text{NULL}(A) \wedge \text{NULL}(b) = \text{NULL}(A) \wedge F = F$

$\text{NULL}(Bc) = \text{NULL}(B) \wedge \text{NULL}(c) = F$

$\text{NULL}(A) = \text{NULL}(aA) \vee \text{NULL}(\varepsilon) = F \vee T = T$

$\text{NULL}(aA) = \text{NULL}(a) \wedge \text{NULL}(A) = F \wedge \text{NULL}(A) = F$

$\text{NULL}(B) = \text{NULL}(acB) \vee \text{NULL}(\varepsilon) = \text{NULL}(acB) \vee T = T$

FIRST : primo terminale derivabile da una stringa

1. $\text{FIRST}(\varepsilon) = \varepsilon$
2. $\text{FIRST}(t) = t$
3. $\text{FIRST}(\alpha\gamma) = \begin{cases} \text{FIRST}(\alpha) & \text{se } \text{NULL}(\alpha) = F \quad (3a) \\ \text{FIRST}(\alpha) \setminus \{\varepsilon\} \cup \text{FIRST}(\gamma) & \text{se } \text{NULL}(\alpha) = T \quad (3b) \end{cases}$
4. $\text{FIRST}(X) = \bigcup_{X \to \gamma \text{ in } G} \text{FIRST}(\gamma)$

$Fi(S) = Fi(Ab) \cup Fi(Bc) = \{a, b\} \cup \{a, c\} = \{a, b, c\}$ verica che S non può produrre ε

$Fi(Ab) = Fi(A) \setminus \{\varepsilon\} \cup Fi(b) = \{a\} \cup \{b\} = \{a, b\}$ 3b

$Fi(A) = Fi(aA) \cup Fi(\varepsilon) = \{a\} \cup \{\varepsilon\} = \{a, \varepsilon\}$

$Fi(aA) = Fi(a) = a$ perché a essendo terminale è NULLABLE 3a

$Fi(Bc) = Fi(B) \setminus \{\varepsilon\} \cup Fi(c) = \{a\} \cup \{c\} = \{a, c\}$

$Fi(B) = Fi(acB) \cup Fi(\varepsilon) = \{a\} \cup \{\varepsilon\} = \{a, \varepsilon\}$

FOLLOW : simbolo terminale che trovo dopo un non terminale nullable ; potrebbe anche essere l' EOF ($)

→ MINIMO PUNTO FISSO

1. $Follow(S) = \{\$\}$

2. $Follow(X) = \bigcup_{Z \to \delta X \gamma \text{ in } G} FIRST(\gamma) \setminus \{\varepsilon\}$ 2a il FIRST della parte di regola "dopo di me"

$\bigcup_{Z \to \delta X \gamma \text{ in } G,\ NULL(\gamma) = T} Follow(Z)$ 2b quando γ è nullable devo il mio follow è il follow della regola dove sono uscito

$FO(S) = \{\$\}$ 1

$FO(A)$: guardiamo tutte le regole dove c'è A

$= \underbrace{Fi(b) \setminus \{\varepsilon\}}_{2a} \cup \underbrace{FO(A)}_{2b} = \{b\} \cup FO(A) = \{b\} \cup \{b\} = \{b\}$

→ CASO RICORSIVO

TROVATO CON IL METODO DEL PUNTO FISSO

$FO(B) = \underbrace{Fi(c) \setminus \{\varepsilon\}}_{2a} \cup \underbrace{FO(B)}_{2b} = \{c\} \cup FO(B) = \{c\} \cup \{c\} = \{c\}$

→ CASO RICORSIVO

TROVATO CON IL METODO DEL PUNTO FISSO

CASO RICORSIVO ⇒ METODO DEL PUNTO FISSO

	CASO BASE	1	2
FO(A)	∅	$\{b\} \cup \emptyset = \{b\}$	$\{b\} \cup \{b\} = \{b\}$
FO(B)	∅	$\{c\} \cup \emptyset = \{c\}$	$\{c\} \cup \{c\} = \{c\}$

→ per due iterazioni non è cambiato quindi $FO(A) = \{b\}$

→ per due iterazioni non è cambiato quindi $FO(B) = \{c\}$

inizio dall'insieme vuoto

applico FO(...) con il valore dell'iterazione precedente

TROVATO COME DEFINIZIONE RICORSIVA

	a	b	c	ε
S	$S \to Ab$, $S \to Bc$	$S \to Ab$	$S \to Bc$	
A	$A \to aA$	$A \to \varepsilon$		
B	$B \to acB$		$B \to \varepsilon$	

Per ogni produzione $A \to \alpha$:

1: Per ogni terminale "a" in FIRST(α):
$LL^1_G[A, a] = A \to \alpha$

$i=0$: $S \to Ab$
$Fi(Ab) = \{a, b\}$
$LL^1_G[S, a] = S \to Ab$
$LL^1_G[S, b] = S \to Ab$

$i=1$: $S \to Bc$
$Fi(Bc) = \{a, c\}$
$LL^1_G[S, a] = S \to Bc$
$LL^1_G[S, c] = S \to Bc$

2: Se $\varepsilon \in Fi(\alpha)$ allora per ogni $b \in Fo(A)$:
$LL^1_G(A, b) = A \to \alpha$

$i=3$ $A \to \varepsilon$
$Fo(A) = \{b\}$
$LL^1_G(A, b) = A \to \varepsilon$

$i=5$ $B \to \varepsilon$
$Fo(B) = \{c\}$
$LL^1_G(B, c) = B \to \varepsilon$

TABELLA COMPLETA

La grammatica NON è LL(1) perché $LL^1_G[S, a]$ è sia $S \to Ab$ che $S \to Bc$

ESISTE UN k per cui essa è LL(k)?

$S \Rightarrow Ab \Rightarrow aAb \Rightarrow ab$
$S \Rightarrow Bc \Rightarrow acc$

$\Rightarrow$ devo poter guardare avanti un carattere in più quindi G è LL(2)

$\mathcal{L}(G) = \{b, c, \underline{ab}, \underline{acc}, \dots\}$ per poter differenziare queste due produzioni devo leggere 2 caratteri, non posso avere altre produzioni che si confondono perché A e B producono sequenze di terminali differenti

Ricapitolando

$\forall$ produzione in G $X \to \gamma$ ci sono da fare 2 cose:

- $\forall\, t$ in FIRST(γ): $LL^1_G[X, t] = X \to \gamma$
- Se $\varepsilon \in$ FIRST(γ): $LL^1_G[X, t] = X \to \gamma$

- Se $\varepsilon \in$ FIRST(γ) && $\$ \in$ FOLLOW(X) : $LL^1_G[X, \$] = X \to \gamma$

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); RELATIVA ALLA GENERAZIONE DEL BYTECODE
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).

Es 2

Dare la regola di inferenza per quanto riguarda le assegnazioni

$$\frac{\overset{3}{x \notin dom(\Sigma)}}{\Sigma \vdash \underset{1}{int\ x\ ;}\ :\ \Sigma[\underset{2}{x \mapsto N\_init}]}\ [\text{dichiarazioni}]$$

$$\frac{\overset{2}{\Sigma \vdash d : \Sigma'} \qquad \overset{3}{\Sigma' \vdash D : \Sigma''}}{\Sigma \vdash \underset{1}{d;D}\ ;\ :\ \Sigma''}\ [\text{sequenza di dichiarazioni}]$$

$$\frac{\overset{2}{x \in dom(\Sigma)} \qquad \overset{2}{\Sigma \vdash e : int}}{\Sigma \vdash \underset{1}{x = e\ ;}\ :\ \Sigma[x \mapsto init}\ [\text{assegnazione}]$$

$$\frac{Integer \in \mathbb{N}}{\Sigma \vdash Integer}\ [\text{exp integer}] \qquad \frac{\Sigma \vdash e \qquad \Sigma \vdash e'}{\Sigma \vdash e + e'}\ [\text{exp +}] \qquad \frac{\Sigma(x) = init}{\Sigma \vdash \underset{id}{x}}\ [\text{exp id}]$$

$$\frac{\Sigma \vdash s : \Sigma'' \qquad \Sigma \vdash S : \Sigma'}{\Sigma \vdash s\ S\ :\ \Sigma'}\ [\text{seq di stm}]$$

$$\frac{[\text{seq dec}]\ \dfrac{}{\Sigma \vdash dec : \Sigma'} \qquad \dfrac{}{\Sigma' \vdash stm : \Sigma''}\ [\text{seq di stm}]}{\Sigma \vdash let\ \ sec\ \ in\ \ stm : \Sigma''}\ [\text{prog}]$$

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

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

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.

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)

$$\begin{array}{rl} A \rightarrow & BC \\ B \rightarrow & aB \quad | \quad \varepsilon \\ C \rightarrow & CbB \quad | \quad c \end{array}$$

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

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

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

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) ;
}
```

Es 1

$A \to BC$

$B \to aB \mid \varepsilon$

$C \to cT$

$T \to bBT \mid \varepsilon$

	a	b	c	\$
A	$A \to BC$		$A \to BC$	
B	$B \to aB$			$B \to \varepsilon$
C			$C \to cT$	
T		$T \to bBT$		$T \to \varepsilon$

- $A \to BC$

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

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

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

- $B \to aB$

$FIRST(aB) = \{a\}$

- $B \to \varepsilon$

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

$FOLLOW(B) = \underbrace{FIRST(C)}_{\{\#\}} \cup FOLLOW(B) \cup \underbrace{FOLLOW(T)}_{\left(FOLLOW(C) \cup FOLLOW(T)\right) = \$}$

$= \{\$\} \cup FOLLOW(B) = \{\$\}$

$$\mathrm{FOLLOW}(T) = \underbrace{\mathrm{FOLLOW}(C)} \cup \mathrm{FOLLOW}(T)$$

$$= \mathrm{FOLLOW}(A) \cup \mathrm{FOLLOW}(T)$$

$$= \{\$\} \cup \mathrm{FOLLOW}(T) = \{\$\}$$

	0	1	2
$\mathrm{FOLLOW}(T)$	$\emptyset$	$\{\$\} \cup \emptyset$	$\$$

- $C \to cT$

  $\mathrm{FIRST}(cT) = \{c\}$

- $T \to bBT$

  $\mathrm{FIRST}(bBT) = \{b\}$

- $T \to \varepsilon$

  $\mathrm{FOLLOW}(T) = \{\$\}$

Esame del 2020 - 07 - 03

Es ①

$$\frac{id \notin dom(\Gamma)}{\Gamma \vdash t\ id\ ;\ : \Gamma[id \mapsto t]}\ [dec] \qquad \frac{\Gamma \vdash d\ ; : \Gamma'' \quad \Gamma'' \vdash D\ ; : \Gamma'}{\Gamma \vdash d; D;\ : \Gamma'}\ [seq\ D]$$

$$\frac{Integers \in \mathbb{Z}}{\Gamma \vdash Integers\ : \Gamma}\ [EXP-INT] \qquad \frac{Double \in \mathbb{R}}{\Gamma \vdash Double\ : \Gamma}\ [EXP-DOUBLE]$$

$$\frac{id \in dom(\Gamma)}{\Gamma \vdash id\ : \Gamma}\ [exp-id]$$

TYPE CHECK

$$\frac{\Gamma \vdash e_1 : Int \quad \Gamma \vdash e_2 : Int \quad + : int \times int \to int}{\Gamma \vdash e_1 + e_2\ : \Gamma}\ [exp-+-int]$$

$$\frac{\Gamma \vdash e_1 : Double \quad \Gamma \vdash e_2 : Double \quad + : Double \times Double \to Double}{\Gamma \vdash e_1 + e_2\ : \Gamma}\ [exp-+-D]$$

$$\frac{\Gamma \vdash e_1 : Integer \quad \Gamma \vdash e_2 : Double \quad + : int \times double \to Double}{\Gamma \vdash e_1 + e_2\ : \Gamma}\ [exp-+-D_1]$$

$$\frac{\Gamma \vdash e_1 : Double \quad \Gamma \vdash e_2 : Integer \quad + : Double \times Int \to Double}{\Gamma \vdash e_1 + e_2\ : \Gamma}\ [exp-+-D_2]$$

TYPE INFERENCE

$T_1 = int = T_2$

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2 \quad + : T_1 \times T_2 \to int}{\Gamma \vdash e_1 + e_2\ : \Gamma}\ [exp-+-int]$$

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2 \quad + : T_1 \times T_2 \to Double}{\Gamma \vdash e_1 + e_2\ : \Gamma}\ [exp-+-int]$$

$$\frac{T_1 = \Gamma(id) \quad \Gamma \vdash exp : T_2 \quad T_1 = T_2}{\Gamma \vdash id\ =\ exp;\ : \Gamma}\ [statement] \qquad \frac{\Gamma \vdash s; : \Gamma \quad \Gamma \vdash S; : \Gamma}{\Gamma \vdash s; S;\ \vdash \Gamma}\ [seq\ statement]$$

$$\dfrac{\Gamma \vdash D : \Gamma' \qquad \Gamma' \vdash S : \Gamma'}{\underset{\emptyset}{\underset{\shortparallel}{\Gamma}} \vdash \mathrm{let}\ D\ \mathrm{in}\ S\ ;\ : \Gamma'}\ [\mathrm{prog}]$$

3-07-2020

Avendo override dell'operatore +
- $e_1 + e_2$ con e_1 ed e_2 di tipo INT ritorna un int
- $e_1 + e_2$ con e_1 o e_2 di tipo DOUBLE ritorna un DOUBLE

$$\frac{\Gamma \vdash e_1 : int, e_1' \qquad \Gamma \vdash e_2 : int, e_2'}{\Gamma \vdash e_1 + e_2 : int, e_1' +_{int} e_2'}$$

→ il tipo riporta anche l'espressione, indica che sono due nodi dell'AST differenti

$$\frac{\Gamma \vdash e_1 : T_1, e_1' \qquad \Gamma \vdash e_2 : T_2, e_2' \quad T_1 = double \text{ OR } T_2 = double}{\Gamma \vdash e_1 + e_2 : int, \underbrace{e_1' +_{double} e_2'}_{}}$$

↳ codice + espressivo

$$\frac{\Gamma \vdash e : T, e' \quad \Gamma(x) = T}{\Gamma \vdash x = e : x = e'}$$

NB: il formato dei giudizi che deve essere mantenuto in tutto l'esercizio

$\Gamma \vdash$ Espressione : Tipo, Espressione'

↳ espressione modificata $\Gamma \vdash e_1 + e_2 : int, e_1' +_{int} e_2'$

$\Gamma \vdash$ Statement : Statement'

$\Gamma, n \vdash$ Declaration : Γ', n'

↳ indirizzo dell'offset della prossima variabile da scrivere

$\Gamma \vdash$ Program : Γ', Statement'

NB2: Come è fatto Γ (la symbol table)

Una semplice hash map ID → TIPO, OFFSET perché non ho ambienti annidati

$$\frac{}{\Gamma, n \vdash int\ x : \Gamma[x \mapsto int], n+4}\ [dec\_int]$$

$$\frac{}{\Gamma, n \vdash double\ x : \Gamma[x \mapsto double], n+8}\ [dec\_double]$$

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

sequenza di dichiarazioni dello scope (l'unico che c'è)

$$\frac{\emptyset, 0 \vdash D : \Gamma, n \qquad \Gamma \vdash S : S'}{\Gamma \vdash let\ D\ in\ S : \Gamma', S'}\ [Prog]$$

20-07-2020

$$\frac{\emptyset, 0 \vdash D : \Sigma'' \qquad \Sigma \vdash S : \Sigma'}{\emptyset \vdash \text{let } D \text{ in } S : \Sigma'}\ [\text{prog}]$$

$$\frac{id \notin dom(\Sigma)}{\Sigma, m \vdash \text{int } id; : \Sigma[id \mapsto dec], m+4}\ [\text{dec}]$$

$$\frac{\Sigma, m \vdash d : \Sigma'', m'' \qquad \Sigma'', m'' \vdash D : \Sigma', m'}{\Sigma, m \vdash d; D : \Sigma', m'}\ [\text{seqDec}]$$

$$\frac{Int \in Integers}{\Sigma \vdash Int : \Sigma}\ [\text{exp-Int}]$$

$$\frac{id \in dom(\Sigma) \quad \Sigma(id) = init}{\Sigma \vdash id : \Sigma}\ [\text{exp int}]$$

$$\frac{\Sigma \vdash e_1 : \Sigma \quad \Sigma \vdash e_2 : \Sigma}{\Sigma \vdash e_1 + e_2 : \Sigma}\ [\text{exp+}]$$

$$\frac{\begin{array}{l} id \in dom(\Sigma) \\ \Sigma(id) = dec \qquad \Sigma \vdash exp : \Sigma \end{array}}{\Sigma \vdash id = exp; : \Sigma[id \mapsto init]}\ [\text{stm}]$$

$$\frac{\Sigma \vdash s; : \Sigma'' \qquad \Sigma' \vdash S : \Sigma'}{\Sigma \vdash s; S : \Sigma'}\ [\text{seqStm}]$$

$$\frac{x \notin dom(\emptyset)}{\emptyset, 0 \vdash \text{int } x : \emptyset[x \mapsto dec], 4}\ [\text{dec}]$$

$$\frac{y \notin dom([x \mapsto dec])}{[x \mapsto dec], 4 \vdash \text{int } y : [x \mapsto dec][y \mapsto dec], 8}$$

$$\frac{\emptyset, 0 \vdash \text{int } x : \emptyset[x \mapsto dec], 4 \qquad [x \mapsto dec], 4 \vdash \text{int } y : [x \mapsto dec][y \mapsto dec], 8}{\emptyset, 0 \vdash \text{int } x; \text{ int } y : \Sigma', m = [x \mapsto dec, y \mapsto dec], 8}\ [\text{seq dec}]$$

$$\frac{y \in dom(\Sigma') \quad \Sigma'(y) = dec \qquad \dfrac{5 \in Integers}{\Sigma' \vdash 5 : \Sigma'}}{\Sigma' \vdash y = 5; : \Sigma'[y \mapsto init]}\ [\text{stm}]$$

$$\frac{x \in dom(\Sigma'') \quad \Sigma(x) = dec \qquad \dfrac{}{\Sigma'' \vdash 3+y}}{\Sigma'' = \begin{bmatrix} x \mapsto dec \\ y \mapsto init \end{bmatrix} \vdash x = 3 + y : \Sigma''}$$

$$\frac{\Sigma' \vdash y = 5; : \Sigma'[y \mapsto init] \qquad \Sigma'' = \begin{bmatrix} x \mapsto dec \\ y \mapsto init \end{bmatrix} \vdash x = 3 + y : \Sigma''}{\Sigma' \vdash y = 5; x = 3 + y : \Sigma}\ [\text{seqStm}]$$

$$\frac{\emptyset, 0 \vdash \text{int } x; \text{ int } y : \Sigma', m = [x \mapsto dec, y \mapsto dec], 8 \qquad \Sigma' \vdash y = 5; x = 3+y : \Sigma}{\emptyset \vdash \text{int } x; \text{ int } y; \text{ in } y = 6; x = 3 + y; : \Sigma = [x \mapsto init, y \mapsto init]}\ [\text{prog}]$$

```
cgen(Σ, let decs in stms)
<——————————— COSTRUZIONE FRAME   ⚠ DA RIVEDERE
for (dec in decs) {
   cgen(Σ, dec)
}
for (stm in stms) {
   cgen(Σ, stm)
}

cgen(T, let D in S) =

   addi $sp $sp -4*(D.length) ; creo lo spazio nello STACK le le variabili dichiarate in D
   for (int i=0, i < S.length(); i++)   } per ogni statement chiamo cgen
      cgen(T, s)
   addi $sp $p +4*(D.length)   } libero lo stack delle variabili dichiarate e inizializzate

cgen(T, iD = exp)
   cgen(T, exp)
   sw $a0   T(iD).offset($sp)

cgen(T, number)
   li $a0 number
cgen(T, iD)
   lw $a0 T(iD).offset($sp)
cgen(T, e1 + e2)
   cgen(T, e1)
   push $a0
```

cgen(Γ, e_2)

lw \$t1 top_stack

addi \$a0 \$a0 \$t1

pop

$\Gamma \vdash$ class A { ... } : Γ'

ESTENDO L'AMBIENTE CHE USO MA NON LO RESTITUISCO

$\Gamma' = \Gamma[A \mapsto [f:T,\ m:T'' \to T'],\ f \mapsto T]$

$$\frac{A \notin dom(\Gamma) \quad \Gamma[A\mapsto[f:T,\ m:T''\to T']] \vdash T\ f : \Gamma' \quad \Gamma'[m\mapsto T''\to T',\ y\mapsto T'']\vdash s : T''' \quad T''' <: T'}{\Gamma \vdash \text{class } A\ \{T\ f;\ T'\ m(T''\ y)\ \{s\}\} : \Gamma[A\mapsto[f:T,\ m:T''\to T']]}$$

CASO SENZA OVERRIDE

$$\frac{\begin{array}{lll} A \in dom(\Gamma) \quad f, \boxed{m \notin dom(\Gamma.A)} & & \Gamma[B\mapsto \Gamma.A[f:T,\ m:T''\to T']] \vdash T\ f : \Gamma' \\ B \notin dom(\Gamma) & & \Gamma'[m\mapsto T''\to T',\ y\mapsto T'']\vdash s : T''' \quad T''' <: T' \end{array}}{\Gamma \vdash \text{class } B \text{ extends } A\ \{T\ f;\ T'\ m(T''\ y)\ \{s\}\} : \Gamma[B\mapsto \Gamma.A[f:T,\ m:T''\to T']]}$$

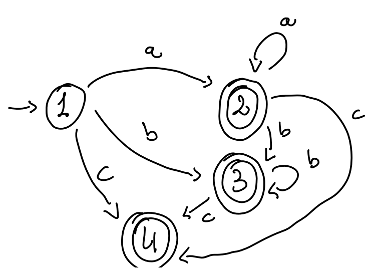
CASO CON OVERRIDE

in caso di override i tipi delle funzioni di B devono essere sottotipi delle funzioni di A

$$\frac{\begin{array}{lll} A \in dom(\Gamma) \quad \boxed{m \in dom(\Gamma.A)} & & \Gamma[B\mapsto \Gamma.A[f:T,\ m:T''\to T']] \vdash T\ f : \Gamma' \\ B \notin dom(\Gamma) & & \Gamma'[m\mapsto T''\to T',\ y\mapsto T'']\vdash s : T''' \quad T''' <: T' \quad T''\to T' <: \Gamma.A(m) \end{array}}{\Gamma \vdash \text{class } B \text{ extends } A\ \{T\ f;\ T'\ m(T''\ y)\ \{s\}\} : \Gamma[B\mapsto \Gamma.A[f:T,\ m:T''\to T']]}$$

B È ASSOCIATO AD UN AGGIORNAMENTO DI UNA COPIA DELL'AMBIENTE DI A

copio A e lo modifico



INPUT DI ANTLR FUNZIONANTE

Q0 : (S)*

S : 'a'A | 'b'B | 'c'C

A : 'a'A | 'b'B | 'c'C

B : 'b'B | 'c'C

C : 'c'C

regole per whitespace, ...

$\breve{U}_c$

19 - 12 - 2019

Esercizio 1:

$$\text{NULLABLE}(\gamma) = \begin{cases} \text{TRUE} & \Gamma \Rightarrow^* \varepsilon \\ \text{FALSE} & \text{altrimenti} \end{cases}$$

$$\text{NULLABLE}(\varepsilon) = \text{TRUE}$$

$$\text{NULLABLE}(a) = \text{FALSE} \qquad a \in T$$

$$\text{NULLABLE}(\alpha\gamma) = \text{NULLABLE}(\alpha) \wedge \text{NULLABLE}(\gamma)$$

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

$$\text{FIRST}: (N \cup T)^* \to \mathcal{P}(T \cup \{\varepsilon\})$$

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

$$\text{FIRST}(a) = \{a\} \quad \text{con} \quad a \in T$$

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

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

$$\text{FOLLOW}: N \to \mathcal{P}(T \cup \{\$\})$$

$$\text{FOLLOW}(S) = \{\$\} \quad \text{con } S \text{ produzione iniziale}$$

$$\underset{X \in N}{\text{FOLLOW}(X)} = \bigcup_{\substack{Y \to \alpha X \gamma \\ \text{NULLABLE}(\gamma) = \text{FALSE}}} \text{FIRST}(\gamma) \cup \bigcup_{\substack{Y \to \alpha X \gamma \\ \text{NULLABLE}(\gamma) = \text{TRUE}}} \text{FOLLOW}(Y)$$

Esercizio 2

Le regole di inferenza sono definite

$$\Gamma \vdash \text{Prg} : \Gamma'$$
$$\Gamma \vdash \text{Fun} : \Gamma'$$
$$\Gamma \vdash \text{Stm} : \text{Tipo}$$

Γ è definito come insieme di coppie $\text{ID} \mapsto \text{TIPO}$

$$\emptyset \Vdash Fun^* : \Gamma''$$

$$\frac{\Gamma'' \vdash Fun^* : \Gamma' \qquad \Gamma' \vdash Stm : \Gamma}{\emptyset \vdash Fun^*\ Stm \ : \Gamma}$$

Se Stm modifica l'ambiente potendo definire variabili allora restituisce un Γ' modificato. Altrimenti restituisce Γ'

$$\frac{\emptyset \Vdash Fun : \Gamma' \qquad \Gamma' \Vdash Fun^* : \Gamma}{\emptyset \Vdash Fun\ Fun^* \ : \ \Gamma} \text{[seq Fun pre visit]}$$

$$\frac{ID \notin dom(\Gamma)}{\Gamma \Vdash TIPO\ ID\ (FPAR) : \Gamma[ID \mapsto FPAR \to TIPO]} \text{[Fun pre visit]}$$

$$\frac{\Gamma \vdash Fun : \Gamma'' \qquad \Gamma'' \vdash Fun^* : \Gamma'}{\Gamma \vdash_{Fun} Fun^* \ : \ \Gamma'} \text{[seq Fun]}$$

$$\frac{\Gamma \bullet [\,] \vdash FPAR : \overbrace{\Gamma \bullet [ID_1 \mapsto T_1, \dots, ID_n \mapsto T_n]}^{\Gamma''} \qquad \Gamma'' \vdash STM : T' \qquad T' = T}{\Gamma \vdash T\ ID\ (FPAR) = STM : \Gamma'} \text{[fun]}$$

```
Env CheckProg ( Env Γ , Prg p )   prg
   for f in p.Funs {
      if Γ(f.id) ≠ null then throw Exception // doppia definizione di f.id
      Γ = Γ.update(f.id , f.type) // Γ[ f.id ↦ f.FPAR → f.type ]
   }
   for f in p.Funs {
      Γ = Γ.push()
      Γ = Γ.update( f.FPAR )  // Γ • [ id1 ↦ T1 , ... , idn ↦ Tn ]
      RetType = checkStm ( Γ , f.Stm )
      if RetType ≠ f.Type then throw Exception
      Γ.pop()
   }
   Γ = checkStm ( Γ, p.Stm )
   return Γ
```

$$\frac{*}{\emptyset \Vdash int\ f(int\ x) = \dots ;\ int\ g(int\ w, int\ v) = \dots \ : \emptyset[f \mapsto int \to int,\ g \mapsto int \times int \to int] = \Gamma'} \text{[seq Fun First pass]}$$

$$\frac{\dfrac{**}{\Gamma' \vdash int\ f(int\ x) = return\ (g(x,x)+1);\ int\ g(int\ w, int\ v) = return\ (f(w+v));\ : \Gamma'} \text{[seq Fun]}}{\emptyset \vdash int\ f(int\ x) = return\ (g(x,x)+1);\ int\ g(int\ w, int\ v) = return\ (f(w+v));\ print\ (f(1) + g(2,3));\ : \Gamma}$$

$$\dfrac{\dfrac{f \notin dom(\emptyset)}{\emptyset \Vdash int\ f(int\ x) = \ldots\ :\ \emptyset[f \mapsto int \to int] = \Gamma''}\,[\text{fun pre init}] \qquad \dfrac{g \notin dom([f \mapsto int \to int])}{\Gamma'' \vdash int\ g(int\ w,\ int\ v) = \ldots\ :\ \Gamma''[g \mapsto int \times int \to int] = \Gamma'}\,[\text{fun pre init}]}{\emptyset \Vdash int\ f(int\ x) = \ldots\ ;\ int\ g(int\ w,\ int\ v) = \ldots\ :\ \emptyset[f \mapsto int \to int,\ g \mapsto int \times int \to int] = \Gamma'}\,[\text{seq Fun first pass}]$$

$$\dfrac{\dfrac{\Gamma' \cdot [\,] \vdash int\ x : \Gamma' \cdot [\,][x \mapsto int] = \Gamma'' \qquad \dfrac{}{\Gamma'' \vdash return(g(x,x)+1)\ :\ T(=int)}\,[\text{stm}] \qquad T = int}{\Gamma' \vdash int\ f(int\ x) = return(g(x,x)+1) : \Gamma'} \qquad ***}{\Gamma' \vdash int\ f(int\ x) = return(g(x,x)+1)\ ;\ int\ g(int\ w,\ int\ v) = return(f(w+v))\ ;\ :\ \Gamma'}\,[\text{seq Fun}]$$

$$***\quad \Gamma' \cdot [\,] \vdash int\ w,\ int\ v : \Gamma' \cdot [\,][w \mapsto int,\ v \mapsto int] = \Gamma'' \qquad \dfrac{}{\Gamma'' \vdash return(f(w+v))\ ;\ :\ T(=int)}\,[\text{stm}] \qquad T = int$$

Esercizio 3

cgen $(\Gamma$, do S while $E)$

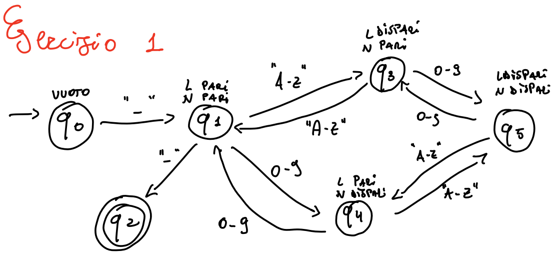
```
loopLabel = generateLabel() // genera una label univoca per il ciclo
loopLabel:
   cgen (Γ, S)

   cgen (Γ, E)   // $a0 é vero <=> $a0 = 1
                    $a0 é falso <=> $a0 = 0

   li $t1  1

   beq $a0 $t1 loopLabel


loop1:
loop2:
lw   $a0  4($fp) // cgen (Γ, x)

addi $a0  $a0  1 // cgen (Γ, x+1)

sw   $a0  4($fp) // cgen (Γ, x := x+1)
```

```
lw $a0 8($fp) // cgen(T, y)
push $a0
lw $a0 4($fp) // cgen(T, x)
lw $t1 0($sp)                  cgen(T, x+y)
add $a0 $a0 $t1
pop
sw $a0 8($fp)
lw $a0 4($fp) // cgen(T, x)
push $a0
lw $a0 8($fp) // cgen(T, y)
lw $t1 0($sp)
bg $a0 $t1 trueBranchloop2 ; bg $r1 $r2 Label jump to Label if $r1 > $r2
li $a0 0
trueBranchloop2:
li $a0 1
li $t1 1
beq $a0 $t1 loop2
lw $a0 4($fp)
push $a0
lw $a0 12($fp)
lw $t1 0($sp)
pop
add $a0 $a0 $t1
push $a0
lw $a0 8($fp)
lw $t1 0($sp)
```

```
bl $a0 $t1 TrueBranch Loop 1

li $a0 0
True Branch Loop1 :
li $a0 1

li $t1 1

beq $a0 $t1 loop 1
```

18-09-2020

Esercizio 1




```
q0 : '_' q1 ;

q1 : LETTERA q3 | DIGIT q4 | '_' ;

q3 : LETTERA q1 | DIGIT q5 ;

q4 : DIGIT q1 | LETTERA q5 ;

q5 : DIGIT q3 | LETTERA q4 ;

fragment LETTERA : 'A'..'Z' ;

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

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

Esercizio 2

$S \to SB \mid y$

$B \to Bx \mid Ax$

$A \to z \mid zSy$

	x	y	z	$
S		$S \to SB$ $S \to y$		
B			$B \to Bx$ $B \to Ax$	
A			$A \to z$ $A \to zSy$	

- $S \to SB$, $FIRST(SB) = FIRST(S) = FIRST(SB) \cup FIRST(y) = FIRST(SB) \cup \{y\} = \{y\}$

	caso base	1	2
$\text{FIRST}(SB)$:	$\emptyset$	$\emptyset \cup \{y\}$	$\{y\} \cup \{y\}$

- $S \rightarrow y$, $\text{FIRST}(y) = \{y\}$
- $B \rightarrow Bx$, $\text{FIRST}(Bx) = \text{FIRST}(B) = \text{FIRST}(Bx) \cup \text{FIRST}(Ax) = \text{FIRST}(Bx) \cup \{z\} = \{z\}$

$\text{FIRST}(Ax) = \text{FIRST}(A) = \text{FIRST}(z) \cup \text{FIRST}(zSy) = \{z\}$

	caso base	1	2
$\text{FIRST}(Bx)$:	$\emptyset$	$\emptyset \cup \{z\}$	$\{z\} \cup \{z\}$

- $B \rightarrow Ax$, $\text{FIRST}(Ax) = \{z\}$
- $A \rightarrow z$, $\text{FIRST}(z) = \{z\}$
- $A \rightarrow zSy$, $\text{FIRST}(zSy) = \{z\}$

	x	y	z	\$
S		$S \rightarrow yS'$		
S'		$S' \rightarrow \varepsilon$	$S' \rightarrow BS'$	
B			$B \rightarrow AxB'$	
B'	$B' \rightarrow xB'$		$B' \rightarrow \varepsilon$	
A			$A \rightarrow zA'$	
A'	$A' \rightarrow \varepsilon$	$A' \rightarrow Sy$		

$S \rightarrow y S'$

$S' \rightarrow B S' \mid \varepsilon$

$B \rightarrow A x B'$

$B' \rightarrow x B' \mid \varepsilon$

$A \rightarrow z A'$

$A' \rightarrow \varepsilon \mid Sy$

- $S \rightarrow yS'$, $\text{FIRST}(yS') = \text{FIRST}(y) = \{y\}$
- $S' \rightarrow BS'$, $\text{FIRST}(BS') = \text{FIRST}(B) = \text{FIRST}(AxB') = \text{FIRST}(A) = \text{FIRST}(zA') = \{z\}$

$\text{NULLABLE}(B) = \text{NULLABLE}(AxB') = \text{NULLABLE}(A) \wedge \text{NULLABLE}(x) \wedge \text{NULLABLE}(B')$

$= \dots \wedge \text{FALSE} \wedge \dots = \text{FALSE}$

$\text{NULLABLE}(A) = \text{NULLABLE}(zA') = \{z\}$

- $S' \rightarrow \varepsilon$, $\text{FIRST}(\varepsilon) = \{\varepsilon\}$

$\text{FOLLOW}(S') = \text{FOLLOW}(S) \cup \text{FOLLOW}(S') = \{y\} \cup \text{FOLLOW}(S') = \{y\}$

	0	1	2
$\text{FOLLOW}(S) =$	$\emptyset$	$\{y\} \cup \emptyset$	$\{y\} \cup \{y\}$

- $B \rightarrow AxB'$, $\text{FIRST}(A) = \text{FIRST}(zA') = \{z\}$

- $B' \to x B'$, $\text{FIRST}(x B') = \{x\}$
- $B' \to \varepsilon$, $\text{FIRST}(\varepsilon) = \{\varepsilon\}$

$$\text{FOLLOW}(B') \overset{!}{=} \text{FOLLOW}(B) \cup \text{FOLLOW}(B') = \text{FIRST}(S') \cup \text{FOLLOW}(B') = \{z\} \cup \text{FOLLOW}(B')$$
$$= \{z\}$$

- $A \to z A'$, $\text{FIRST}(z A') = \{z\}$
- $A' \to \varepsilon$, $\text{FIRST}(\varepsilon) = \{\varepsilon\}$

$$\text{FOLLOW}(A') = \text{FOLLOW}(A) = \text{FIRST}(x B') = \{x\}$$

- $A' \to S y$, $\text{FIRST}(S) = \text{FIRST}(y S') = \{y\}$

Esercizio 3

```
cgen(T, loop k{s})

loopLabel = generateLabel()

li $a0 k
push $a0
loopLabel :
li $t1 0
beq $a0 $t1   endLoop + loopLabel
cgen(T, s)
lw $a0 0($sp)
sobbi $a0 $a0 1
pop
push $a0
b loopLabel

endLoop + loopLabel :
pop


li $a0 34
push $a0
loop 1 :
li $t1 0
beq $a0 $t1 endloop1

lw $al 0($fp)
lw $a0 4($al)
addi $a0 $a0 1

lw $al 0($fp)
sw $a0 4($al)

li $a0 25
```

```
push $a0
loop2:
li $t1 0
beq $a0 $t1 endloop2

lw $al 0($fp)
lw $a0 4($al)
push $a0

lw $al 0($fp)
lw $al 0($al)
lw $a0 4($al)

lw $t1 0($sp)
pop

add $a0 $a0 $t1

lw $al 0($fp)
lw $al 0($al)
sw $a0 4($al)

lw $a0 0($sp)
subi $a0 $a0 1
pop
push $a0
b loop2
end loop2:
pop

lw $a0 0($sp)
subi $a0 $a0 1
pop
push $a0
b loop1
end loop1:
pop
```

Analisi degli Effetti

modifiche di variabili

$D \; ; \; S$

$D ::= \text{int } id = E \; ; \; D$

$S ::= id = E \; ; \; S$

$\Gamma = id \mapsto (\text{effect}, \text{stato})$

$\Gamma, n \vdash D : \Gamma', n'$

$\Gamma \vdash S : \Gamma'$

$$\frac{x \notin dom(\Gamma) \qquad \Gamma \vdash E \qquad \Gamma[x \mapsto (\bot, n)],\, n+4 \vdash D : \Gamma', n'}{\Gamma, n \vdash \text{int } x = E ; D : \Gamma', n'}$$

$$\frac{x \in dom(\Gamma) \qquad \Gamma \vdash E \qquad \Gamma[x.\text{effect} \to T] \vdash S : \Gamma'}{\Gamma \vdash x = E ; S : \Gamma'}$$

19-02-20

Es 1

$A \to BC$

$B \to aB \mid \varepsilon$

$C \to CbB \mid c$

Rimozione ric a sx

$A \to BC$

$B \to aB \mid \varepsilon$

$C \to cC'$

$C' \to bBC' \mid \varepsilon$

	a	b	c	$
A	$A \to BC$		$A \to BC$	
B	$B \to aB$		$B \to \varepsilon$	$B \to \varepsilon$
C			$C \to cC'$	
C'		$C' \to bBC'$		$C \to \varepsilon$

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

$\quad NULLABLE(B) = NULLABLE(aB) \lor NULLABLE(\varepsilon) = TRUE$

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

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

$\quad NULLABLE(C') = \quad = T$

$\quad FOLLOW(B) = FIRST(C) \cup FOLLOW(B) \cup FOLLOW(C')$

$\quad NULLABLE(C) = NULLABLE(C) \lor NULLABLE(C') = F \lor NULLABLE(C') = F$

$\quad = \{c\} \cup FOLLOW(B) \cup FOLLOW(C') = \{c,\$\} \cup FOLLOW(B) = \{c,\$\}$

$\quad FOLLOW(C') \overset{!}{=} FOLLOW(C) \cup FOLLOW(C') = FOLLOW(A) \cup FOLLOW(C') = \{\$\} \cup FOLLOW(C')$

$\quad = \{\$$

$\quad FOLLOW(C') = \overset{0}{\emptyset} \quad \overset{1}{\{\$\} \cup \emptyset} \quad \overset{2}{\{\$\} \cup \{\$\}}$

$\quad FOLLOW(B) = \overset{0}{\emptyset} \quad \overset{1}{\{c,\$\} \cup \emptyset} \quad \overset{2}{\{c,\$\} \cup \{c,\$\}}$

$C \to cC'$, $\quad FIRST(cC') = \{c\}$

$C' \to bBC'$, $\quad FIRST(bBC') = \{b\}$

$C' \to \varepsilon$, $\quad FIRST(\varepsilon) = \{\varepsilon\}$

$\quad FOLLOW(C') = \{\$\}$

La grammatica è LL(1) perché la tabella non ha entry con più di una produzione

Es 2

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

T = LIST
T' = ARRAY LIST

È sbagliata perché manca l'aggiornamento di Γ con $\Gamma[x \mapsto T]$

a) CODICE TIPABILE MA NON LO È: con $B <: A$

let A x = new B in print(x)

b) CODICE NON TIPABILE MA CHE TIPA: con $B <: A$

let A x = new B ; B x = new B in Print(x)

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

Il tipo dichiarato non può essere un sottotipo del tipo argomento

a) CODICE TIPABILE MA CHE NON DOVREBBE: con $B <: A$

let B x = new A in print(x)

b) CODICE NON TIPABILE MA CHE DOVREBBE con $B <: A$

let A x = new B in Print(x)

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

a) CODICE TIPABILE MA CHE NON DOVREBBE: con $B <: A$

A :: m
B : m , B : f

let A x = new B in x.f()

b) CODICE NON TIPABILE MA CHE DOVREBBE con $B <: A$

let A x = new B in x = new A

regola corretta: $\dfrac{\Gamma \vdash e : T' \quad \Gamma[x \mapsto T] \vdash e' : T'' \quad T' <: T}{\Gamma \vdash \text{let } T\ x = e \text{ in } e' : T''}$

Esercizio 3

cgen(Γ, void f(T1 x, T2 y) {s}) =

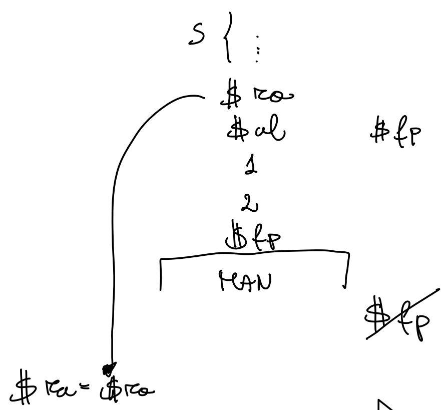
```
Γ(f).label:
    move $fp $sp      ; $fp <- $sp
    push $ra          ; l' $ra é gestito automaticamente
    cgen(Γ, s)
    lw $ra 0($sp)
    addi $sp $sp 8
    lw $fp 0($sp)
    pop
    jr $ra
```

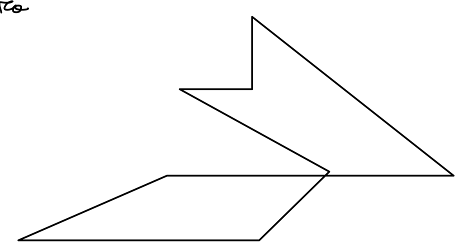
$ra = prossima istruzione

cgen(Γ, f(e, e'))

```
push $fp
cgen(Γ, e')
push $a0
cgen(Γ, e)
push $a0
lw $al 0($fp)
for (int i=0 ; Γ.nestingLevel - Γ(f).nestingLevel ; i++)
    lw $al 0($al)
push $al
jal Γ(f).label
```

{ f(1,2); }





```
li $fp MAXMEM
li $a0 1
push $a0
fact_label:
  move $fp $sp
  push $ra
... cgen (corpo funzione) ...
  lw $al 0($fp)
  sw $a0 0($al)
  push $a0
  li $a0 0
  lw $t1 0($sp)
  pop
  beq $t1 $a0 truebranclabel_1
  push $fp
  lw $al 0($fp)
  sw $a0 4($al)
```

```
push $a0
lw $al 0($fp)
sw $a0 0($al)
lw $t1 0($sp)
mult $a0 $a0 $t1
pop
push $a0
lw $al 0($fp)
sw $a0 0($al)
push $a0
li $a0 1
lw $t1 0($sp)
pop
sub $a0 $t1 $a0
push $a0
lw $al 0($fp)
push $al
jal fact_label
true_branch_1
lw $al 0($fp)
sw $a0 0($al)
lw $al 0($fp)
lw $al 0($al)
sw $a0 0($al)
lw $ra 0($sp)
```

```
addi $sp $sp 16
lw $fp 0($fp)
jr $ra
```

Esercizio Completo

prg : vdec* fdec* stm* ;

vdec : type id ;

fdec : type id (T_1 id_1, ..., T_n id_n) { stm* ; return exp }

stm : id = exp ; | if (exp) then { stm* } (elseif (exp) { stm* })* else { stm* } |

type : int | long int | bool

exp : Integers | Long Integers | Booleans | exp + exp | exp * exp | exp == exp | id |
max (exp , exp) | exp > exp | exp < exp | id (exp_1, ..., exp_n)

Γ : hash map $\langle$ String , (TIPO , STATO , OFFSET) $\rangle$

Definizione dei giudizi

$\Gamma \vdash prg : \Gamma'$

$$\frac{\emptyset \vdash D : \Gamma' \quad \Gamma' \Vdash F : \Gamma'' \quad \Gamma'' \vdash F : \Gamma''' \quad \Gamma''' \vdash S : T}{\emptyset \vdash D ; F ; S ; : T}\,[prg]$$

$\Gamma, n \vdash vdec : \Gamma', n'$

$$\frac{x \notin dom(\Gamma) \qquad T = bool}{\Gamma \vdash T\ x ; : \Gamma[x \mapsto (T, \perp, n)],\ n+1}\,[vdec\ bool]$$

$$\frac{x \notin dom(\Gamma) \qquad T = int}{\Gamma \vdash T\ x ; : \Gamma[x \mapsto (T, \perp, n)],\ n+4}\,[vdec\ int]$$

$$\frac{x \notin dom(\Gamma) \qquad T = longint}{\Gamma \vdash T\ x ; : \Gamma[x \mapsto (T, \perp, n)],\ n+8}\,[vdec\ longint]$$

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

$\Gamma \Vdash fdec : \Gamma'$

$$\frac{f \notin dom(\Gamma)}{\Gamma \Vdash T \; f(T_1 \; id_1, \dots T_n \; id_n) : \Gamma[f \mapsto T_1 \times \dots \times T_n \to T]} \; [\text{fdec - pre}]$$

$$\frac{\Gamma \Vdash f : \Gamma'' \quad \Gamma'' \Vdash F : \Gamma'}{\Gamma \Vdash f \,;\, F : \Gamma'}$$

$\Gamma, n \vdash fdec : \Gamma', n$

$$[\text{fdec}] \; \frac{[\text{seqVdec}]\,\dfrac{}{\Gamma \cdot [\,], n \vdash T_1 \; id_1, \dots, T_n \; id_n : \Gamma \cdot [id_1 \mapsto (T_1, \bot, n), \dots id_n \mapsto (T_n, \bot, n^*)] = \Gamma''} \quad [\text{seqStm}]\,\dfrac{}{\Gamma'' \vdash S : \Gamma'} \quad [\text{exp}]\,\dfrac{}{\Gamma' \vdash exp : T'} \quad T' <: T}{\Gamma, n \vdash T \; f(T_1 \; id_1, \dots, T_n \; id_n)\{S \,;\, \text{return } exp\} : \Gamma, n}$$

$$\frac{\Gamma, n \vdash f : \Gamma'', n \quad \Gamma'', n \vdash F : \Gamma', n}{\Gamma, n \vdash f \,;\, F : \Gamma', n} \; [\text{seq fDec}]$$

$\Gamma \vdash stm : \Gamma'$

$$\frac{x \in dom(\Gamma) \quad \Gamma \vdash e : T \quad T <: \Gamma(x).type}{\Gamma \vdash x = e : \Gamma[x \mapsto (x.type, T, x.offset)]} \; [\text{asgn}]$$

$$\frac{\Gamma \vdash e_1 : T_{e_1} \quad T_{e_1} <: bool \quad \Gamma \vdash S_1 : \Gamma'' \quad \Gamma \vdash S_2 : \Gamma''' \quad \Gamma \vdash seqElseIf : \Gamma^{IV}}{\Gamma \vdash \text{if } e_1 \text{ then } S_1 \; seqElseIf \; \text{else } S_2 : max(\Gamma'', \Gamma''', \Gamma^{IV})}$$

$$\frac{\Gamma \vdash e : T_e \quad T_e <: bool \quad \Gamma \vdash S : \Gamma'}{\Gamma \vdash \text{else if } e \text{ then } S : \Gamma'} \; [\text{elseif}]$$

$$\frac{\Gamma \vdash elif : \Gamma' \quad \Gamma \vdash ELIF : \Gamma''}{\Gamma \vdash elif \; ELIF : max(\Gamma', \Gamma'')} \; [\text{seq elseif}]$$

$\Gamma \vdash exp : T$

$$\frac{e \in Integer \cup LongI}{\Gamma \vdash e : Integer} \; [\text{exp-int}] \qquad \frac{e \in}{\Gamma \vdash e : LongInteger}$$

19-06-2019

$$\frac{x \in dom(\Gamma) \quad \Gamma \vdash E : T_E \quad T_E <: \Gamma(x).type}{\Gamma \vdash x := E : env}$$

```
CHECK STAT ( Γ , x := E )

  if not lookup(Γ, x) the {
     error
     exit
  }

  xType = lookup (Γ, x).type

  EType = E.typecheck()   // E è un nodo di AST che implementa il metodo
                          typecheck che restituisce il tipo di un nodo controllando
                          che non ci siano errori

  if EType is not subtype of xType then {
     errore
     exit
  }
```

is not subtype of ↳ in JAVA sarebbe !(EType instanceof xType)

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

$$\frac{\dfrac{x \in dom(\Gamma) \quad \Gamma \vdash y : T_y \quad T_y <: \Gamma(x).type}{\Gamma \vdash x := y} \qquad \dfrac{y \in dom(\Gamma) \quad \Gamma \vdash z : T_z \quad T_z <: \Gamma(y).type}{\Gamma \vdash y := z} \qquad \dfrac{z \in dom(\Gamma) \quad \Gamma \vdash new\ C : C \quad C <: \Gamma(z).type}{\Gamma \vdash z = new\ C}}{\Gamma \vdash x := y\ ;\ y := z\ ;\ z = new\ C}$$

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

Esercizio 3

interleave C and c' upto E times

cgen(T, interleave C and C' upto E times)

```
cgen(T, E)
push $a0
li $a0 0
push $a0
loop1:
lw $t1 4($sp) ; $t1 <- v
lw $t2 0($sp) ; $t2 <- i
beq $t1 $a0 exit loop 1
modi $a0 $t2 2
beqi $a0 0 execute C
cgen (T, c')
b update_iteration 1
execute C
cgen (T, c)
update_iteration 1
lw $t2 0($sp)
addi $t2 $t2 1
sw $t2 0($sp)
b loop 1
exit loop 1
addi $sp $sp 8
```

~

```
lw $al 0($fp)
lw $a0 4($al)
push $a0
lw $al 0($fp)
lw $al 0($al)
lw $a0 8($al)
lw $t1 0($sp)
pop
add $a0 $a0 $t1
push $a0
li $a0 0
push $a0
loop1:
lw $t1 4($sp)
lw $t2 0($sp)
beq $t1 $t2 exit loop1
modi $a0 $t2 2
beqi $a0 0 execute C
lw $al 0($fp)
lw $a0 4($al)
subi $a0 $a0 1
                    * push $a0
lw $al 0($fp)
                      lw $t1 0($sp)
--> sw $t1 4($al)
                    pop
b update_loop1
execute C :
lw $al 0($fp)
```

```
lw $a1 0($a1)
lw $a0 8($a1)
addi $a0 $a0 1          * push $a0
lw $a1 0($fp)
lw $a1 0($a1)           lw $t1 0($sp)
→ sw $t1 8($a1)         pop
update loop 1:
lw $t2 0($sp)
addi $t2 $t2 1
sw $t2 0($sp)
b loop 1
exit loop:
addi $sp $sp 8
```

Esercizio

Esercizio 3 Dato il codice

```
void f(int a, int b) {
    int x = 1;
    while (x>a) { int x = a, y =b ; a = a-y ; // (@)
        }
}
void g() {
    f(1,5);
}                                        // (*)
```

Scrivere la tabella dei simboli nei punti (@) e (*)

1. con il metodo delle hashtable di liste
2. con il metodo delle liste di hashtable.

1 @

f : $\boxed{\text{int} \times \text{int} \to \text{void}, 0}$
a : $\boxed{\text{int}, 1}$
b : $\boxed{\text{int}, 1}$
x : $\boxed{\text{int}, 1} \to \boxed{\text{int}, 2}$
y : $\boxed{\text{int}, 2}$

* $f : int \times int \to void$
$g : () \to void$

HEAD DELLA LISTA

② @ $\boxed{f : int \times int \to void} \to \boxed{\begin{array}{l} a : int \\ b : int \\ x : int \end{array}} \to \boxed{\begin{array}{l} x : int \\ y : int \end{array}}$

* $\boxed{\begin{array}{l} f : int \times int \to void \\ g : () \to void \end{array}}$

Esercizio

$\Gamma(x) = [\, m \mapsto C_m \to T_m \,]$

$$\frac{x \in dom(\Gamma) \quad m \in dom(\Gamma(x)) \quad \Gamma \vdash y : T_y \quad T_y <: C_m}{\Gamma \vdash x.m(y) : T_m}$$

$\Gamma = [\, a \mapsto T_1 \,,\, z \mapsto T_2 \,,\, y \mapsto T_3 \,]$

$a = x.m(y)$ TIPA

$T_3 <: C_m$

$T_1 <: T_m$