

UNIVERSITÀ DEGLI STUDI DI UDINE

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Specialistica in Informatica

Relazione di Linguaggi di Programmazione 2

GRUPPO 3

Componenti del gruppo:

ALESSIO ONZA
STEFANO BORTOLOTTI
DANTE DEGL'INNOCENTI
DARIO DE NART

ANNO ACCADEMICO 2011-2012

Indice

| | |
|---|-----------|
| 1 Primo esercizio | 1 |
| 1.1 Consegnna | 1 |
| 1.2 Svolgimento | 1 |
| 2 Secondo esercizio | 3 |
| 2.1 Consegnna | 3 |
| 2.2 Svolgimento | 3 |
| 3 Terzo esercizio | 7 |
| 3.1 Consegnna | 7 |
| 3.2 Svolgimento | 8 |
| 3.2.1 Calcolo termini t_1, \dots, t_4 | 8 |
| 4 Quarto esercizio | 11 |
| 4.1 Consegnna | 11 |
| 4.2 Svolgimento | 12 |
| 4.2.1 Vautazione | 12 |
| 4.2.2 Query con variabili libere | 12 |
| 4.2.3 Risultati uguali | 13 |
| 5 Quinto esercizio | 15 |
| 5.1 Consegnna | 15 |
| 5.2 Svolgimento | 16 |
| 5.2.1 Predicato smallParents | 16 |
| 5.2.2 Esecuzione | 16 |
| 6 Sesto esercizio | 17 |
| 6.1 Consegnna | 17 |
| 6.2 Svolgimento | 18 |
| 6.2.1 Soluzione | 19 |

| | |
|---|-----------|
| 7 Settimo esercizio | 21 |
| 7.1 Consegnna | 21 |
| 7.2 Svolgimento | 22 |
| 7.2.1 Costruzione del parser top-down | 22 |
| 7.2.2 Esecuzione del parser | 23 |
| 8 Ottavo esercizio | 27 |
| 8.1 Consegnna | 27 |
| 8.2 Svolgimento | 27 |
| 8.2.1 Costruzione del parser SLR | 27 |
| 8.2.2 Esecuzione del parser SLR | 31 |
| 8.2.3 Costruzione ed esecuzione del parser LALR | 34 |
| 9 Nono esercizio | 39 |
| 9.1 Consegnna | 39 |
| 9.2 Svolgimento | 40 |
| 9.2.1 Grammatica | 40 |
| 9.2.2 Type System | 44 |
| 9.2.3 Descrizione della soluzione | 52 |
| 9.2.4 Grammatica | 52 |
| 9.2.5 Type Checker | 52 |

Capitolo 1

Primo esercizio

1.1 Consegnna

Si stabilisca se il seguente programma è lineare sx/dx, duplicante, eliminante, collassante, se è constructor-based (specificando anche quali funzioni lo sono) e se è ortogonale. Si specifichi quali regole sono in overlap (mostrandone un testimone).

$$\begin{aligned} f [2 , y , -] y &= K y 4 \\ f z (K z) &= z \\ g (f [x] -) &= A x x x \\ g (f x [x]) &= f [] x \end{aligned}$$

1.2 Svolgimento

Assegnamo un'etichetta R_i ad ogni regola del programma per indicare l'i-esima regola a cui ci si riferisce nella costruzione della tabella di classificazione. Ripresentiamo quindi il programma “decorato” con le etichette delle regole:

$$\begin{aligned} R_1 : f [2 , y , -] y &= K y 4 \\ R_2 : f z (K z) &= z \\ R_3 : g (f [x] -) &= A x x x \\ R_4 : g (f x [x]) &= f [] x \end{aligned}$$

Riportiamo quindi in tabella 1.1 la classificazione delle regole e del TRS.

| | Lineare SX | Lineare DX | Duplicante | Eliminante | Collassante |
|------------|------------|------------|------------|------------|-------------|
| R_1 | NO | SI | NO | SI | NO |
| R_2 | NO | SI | NO | NO | SI |
| R_3 | SI | NO | SI | SI | NO |
| R_4 | NO | SI | NO | NO | NO |
| TRS | NO | NO | SI | SI | SI |

Figura 1.1: Tabella classificazione TRS

Il TSR non è Constructor Based, in particolare R_1 e R_2 lo sono, ma R_3 ed R_4 no.

Il TSR non è Ortogonale, in quanto non è lineare sinistro.

Sono presenti inoltre delle regole in overlap e per queste viene mostrato un testimone della sovrapposizione:

- $R_1 - R_3$:

$$g(f [2, y, a] y)$$

in quanto si può applicare la regola R_1 ad f o la regola R_3 a g .

- $R_2 - R_3$:

$$g(f [a] K[a])$$

in quanto si può applicare la regola R_2 ad f o la regola R_3 a g .

- $R_3 - R_4$:

$$g(f [a] [[a]])$$

in quanto sono in overlap in radice.

Capitolo 2

Secondo esercizio

2.1 Consegnna

Si rappresenti il codice della macchina di Warren relativo al seguente programma in Prolog.

```
gf ( [X] , [ ] , X ).  
gf ( [X] , [Y | Q] , R ) :- rv ( [X, Y | Q] , [ - | P] ) , gf ( P, [ ] , R ).  
gf ( [X, -] , [ ] , X ).  
gf ( [X, -] , [Y | Q] , R ) :- rv ( [X, Y | Q] , P ) , gf ( P, [ ] , R ).  
gf ( [X, - , Y | P] , Q , R ) :- gf ( [Y | P] , [X | Q] , R ).  
hanoi ( [B] , S , D , m(B,S,D) ).  
hanoi ( [B|H] , S , D , T , h(m(B,S,D),L,R) ) :-  
    hanoi ( H , S , T , D , L ) , hanoi ( H , T , D , S , R ).
```

2.2 Svolgimento

```
gf/3 :  try_me_else  gf2  
        get_str      cons/2  A1  //  [X]  
        uni_var      X4  
        uni_var      X5  
        get_str      null/0  X5  
        get_str      null/0  A2  //  []  
        get_var      A3  //X  
        get_val      X4  
        proceed
```

```

gf2 : retry_me_else gf3
      alloc          2
      get_str        cons/2 A1    // [X]
      uni_var        X4
      uni_var        X5
      get_str        null/0 X5
      get_str        cons/2 A2    // [Y|Q]
      uni_var        X6
      uni_var        X7
      get_var        A3    // R
      uni_var        Y1
      put_str        cons/2 A1
      set_val        X4
      set_str        cons/2
      uni_val        X6
      uni_val        X7
      put_str        cons/2 A2
      set_var        // -
      set_var        Y2    // P
      call           rev/2
      put_val        Y2    A1
      put_str        null/0 A2
      put_val        Y1    A3
      call           gf/3
      dealloc

gf3 : retry_me_else gf4
      get_str        cons/2 A1    // [X,-]
      uni_var        X4    // X
      uni_var        X5    // -
      get_str        null/0 A2    // []
      get_var        A3    // X
      uni_val        X4
      proceed

```

```

gf4 : retry_me_else gf5
    alloc           2
    get_str        cons/2 A1      // [X, _]
    uni_var        X4
    uni_var        X5
    get_str        cons/2 A2      // [Y|Q]
    uni_var        X6
    uni_var        X7
    get_var        A3      // R
    uni_var        Y1
    put_str        cons/2 A1
    set_val        X4
    set_str        cons/2
    set_val        X6
    set_val        X7
    put_var        Y2      A2      // P
    call           rev/2
    put_val        Y2      A1
    put_str        null/0 A2
    put_val        Y1      A3
    call           gf/3
    dealloc

gf5 : trust_me
    get_str        cons/2 A1      // [X, _, Y|P]
    uni_var        X4      // X
    uni_var        X5
    get_str        cons/2 X5
    uni_var        X6      // _
    uni_var        X7
    get_str        cons/2 X7
    uni_var        X8      // Y
    uni_var        X9      // P
    get_var        A2      // Q
    uni_var        X10
    get_var        A3      // R
    uni_var        X11
    put_str        cons/2 A1      // [Y|P]
    set_val        X8
    set_val        X9
    put_str        cons/2 A2      // [X|Q]
    set_val        X4
    set_val        X10
    put_val        X11     A3
    call           gf/3

```

```
hanoi/5 : try_me_else hanoi'
           get_str      cons/2 A1
           uni_var      X6
           uni_var      X7
           get_str      null/0 X7
           get_var      X8   A2
           get_var      X9   A3
           get_var      X10  A4
           get_str      m/3  A5
           uni_var      X6
           uni_var      X8
           uni_var      X9
           proceed

hanoi' : trust_me
         alloc          5
         get_str      cons/2 A1
         uni_var      X6
         uni_var      Y1
         get_var      Y2   A2
         get_var      Y3   A3
         get_var      Y4   A4
         get_str      h/3  A5
         uni_var      X7
         uni_var      X8
         uni_var      Y5
         get_str      m/3  X7
         uni_val      X6
         uni_val      Y2
         uni_val      Y3
         put_val      Y1   A1
         put_val      Y2   A2
         put_val      Y4   A3
         put_val      Y3   A4
         put_val      X8   A5
         call          hanoi/5
         put_val      Y1   A1
         put_val      Y5   A2
         put_val      Y3   A3
         put_val      Y2   A4
         put_val      Y5   A5
         call          hanoi/5
         dealloc
```

Capitolo 3

Terzo esercizio

3.1 Consegnna

Si determinino opportuni termini t_1, \dots, t_4 affinchè il seguente programma Curry sia correttamente tipato ed inoltre le funzioni f e g siano induttivamente sequenziali.

data $W a b = D b a | E a | G b$

$$g(D y _, x : xs, _) = D x y$$

$$g(t_1, t_2, t_3) = t_1$$

$$g(_, t_4, t_4) = G t_4$$

$$f x (E xs) = (xs, x)$$

$$f x @ (E _) (G y) = (y, x)$$

Si calcoli l'albero di *needed narrowing* per il termine

f (**g** (**x**,**_,_**)) **x** **where** **x** **free**.

Per i rami terminati si scriva la risposta calcolata.

3.2 Svolgimento

3.2.1 Calcolo termini t₁, ..., t₄

I termini t₁, ..., t₄ affinchè il programma Curry risulti correttamente tipato e per cui le funzioni f e g siano induttivamente sequenziali sono:

- t₁ = E y
- t₂ = x:xs
- t₃ = _
- t₄ = []

Il programma Curry risultante dalla sostituzione dei termini t₁, ..., t₄ con i rispettivi valori è:

data W a b = D b a | E a | G b

$$\begin{aligned} g(D y _, x : xs, _) &= D x y \\ g(E y, x : xs, _) &= t1 \\ g(_, [], []) &= G t4 \end{aligned}$$

$$\begin{aligned} f x (E xs) &= (xs, x) \\ f x @ (E _) (G y) &= (y, x) \end{aligned}$$

Il tipo del programma è: g :: (W a a, [[b]], [c]) → W a [b]

In figura 3.1 ed in figura 3.2 vengono presentati i Definitional Tree (DT) rispettivamente per la funzione f e per la funzione g. Questi vengono costruiti a compile time.

In figura 3.3 viene riportato l'albero di *needed narrowing* per il termine

f (g (x, _, _)) x where x free

in cui sono presenti le risposte calcolate, per i rami terminanti.

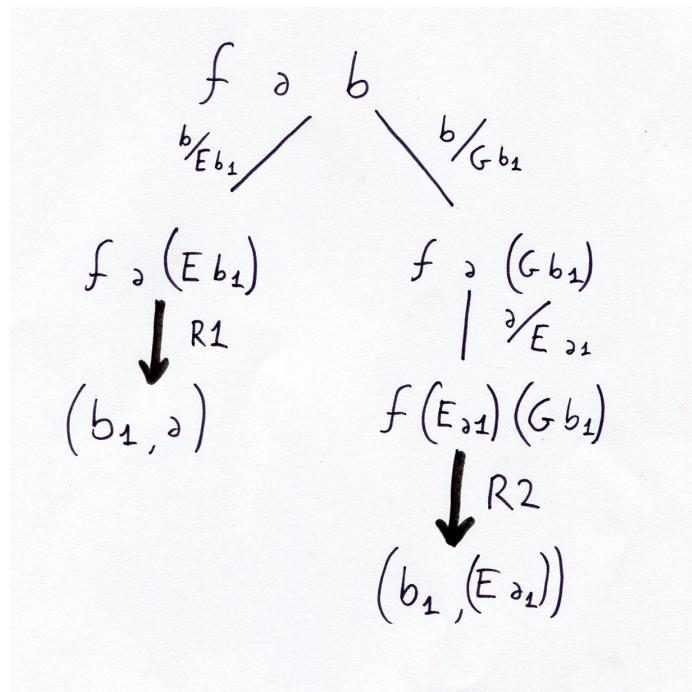


Figura 3.1: Definitional Tree funzione f

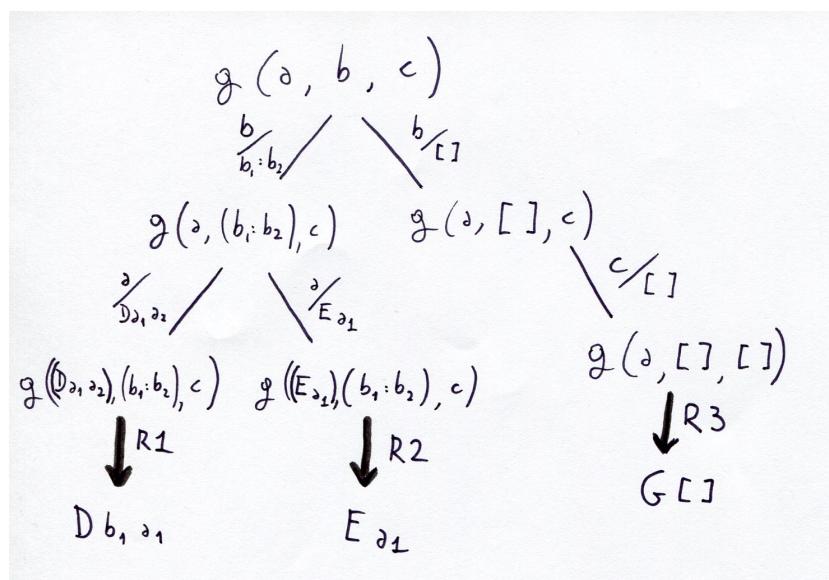


Figura 3.2: Definitional Tree funzione g

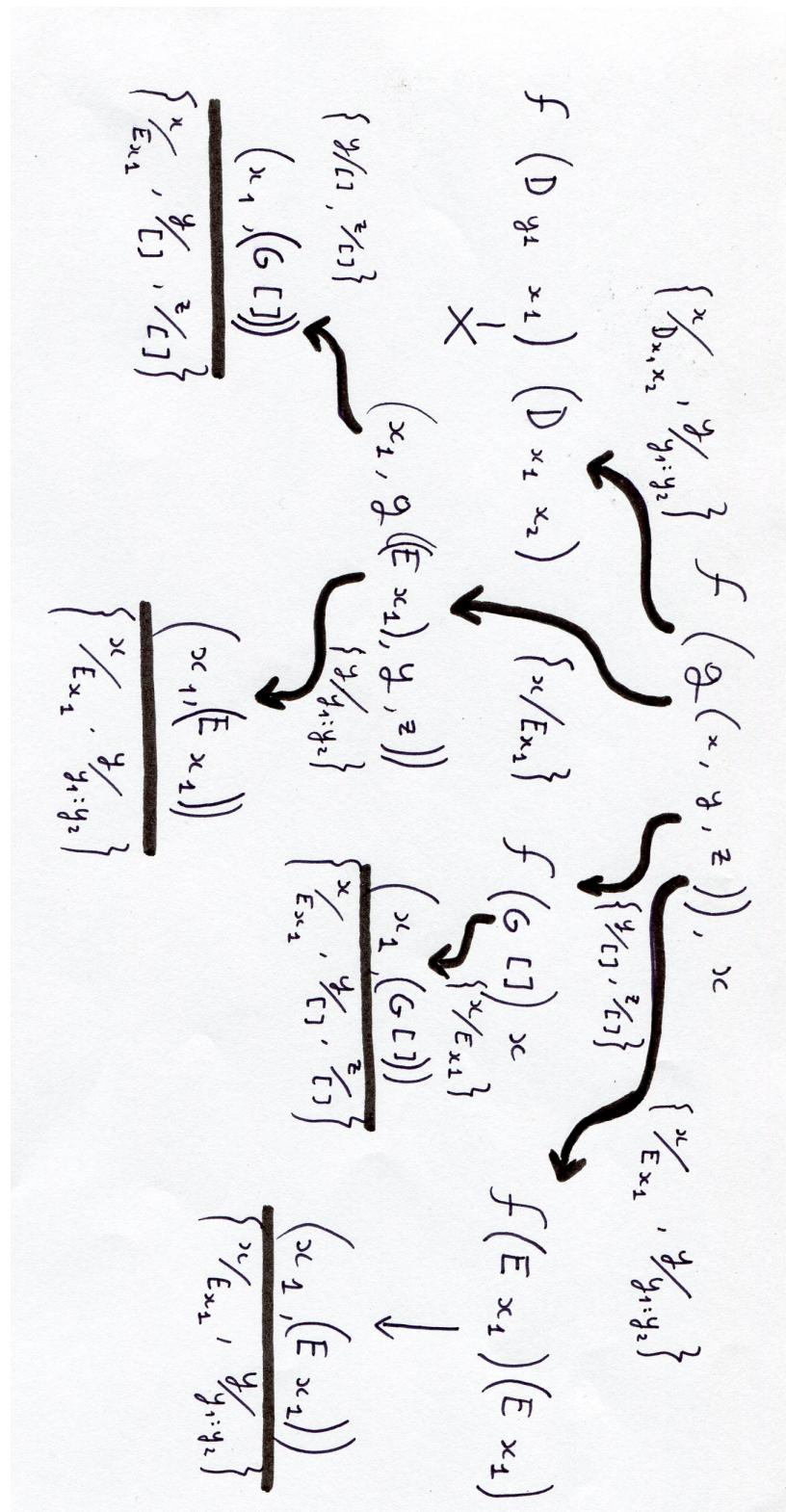


Figura 3.3: Albero di Needed Narrowing

Capitolo 4

Quarto esercizio

4.1 Consegnna

Si consideri il seguente frammento di codice sintatticamente ammissibile sia in Haskell che in Curry.

```
data T a b = V b | N a ( T a b ) ( T a b )
```

```
h _ ( V y ) ( Right z ) = y == z
h x ( N y _ _ ) ( Left z ) = 3 * z == 2 + y
h x ( N y _ _ ) ( Left z ) = ( z < y ) == True & z == x * y & y == 2 + x
```

Si assuma di aver esteso il Prelude di Haskell con le definizioni $(==:)= (==)$ e **True** & x = x.

Preso t = N 1 (V 'a') (V 'b'), si descrivano le valutazioni nei due linguaggi delle queries h (-1) t (Left (-1)) e h 0 t (Left 1).

Si mostri una rappresentazione dello spazio di ricerca di Curry per la query h n x y con n ground e x, y variabili libere.

Se ha senso si riscriva la versione di un linguaggio per ottenere gli stessi risultati di quella dell'altro (su queries ammissibili da entrambi i linguaggi) o si spieghi perché ciò non sia possibile. Oppure si riscriva se avesse un qualche effetto pratico per queries ammissibili solo in un linguaggio.

4.2 Svolgimento

4.2.1 Vautazione

Date le query:

- Q1: $h \ (-1) \ (N \ 1 \ (V \ 'a') \ (V \ 'b')) \ (\text{Left} \ (-1))$
- Q2: $h \ 0 \ (N \ 1 \ (V \ 'a') \ (V \ 'b')) \ (\text{Left} \ 1)$

Le valutazioni delle queries nel linguaggio Curry risultano:

- Q1 → Success
- Q2 → Success

Nel linguaggio Haskell invece risultano:

- Q1 → False
- Q2 → True

4.2.2 Query con variabili libere

Per quanto riguarda la query `h n x y where x, y free ed n ground`, questa restituirebbe, in prima istanza, il risultato Success con il seguente binding delle variabili:

`x/V a, y/Right a`

Nel caso in cui l'utente richiedesse altri risultati, però, l'esecuzione verrebbe sospesa perché le operazioni aritmetiche (rigid) presenti nelle regole 2 e 3 non permettono ulteriori passi di narrowing.

4.2.3 Risultati uguali

Premesso che è sempre possibile valutare variabili libere e funzioni non deterministiche in Haskell a patto di implementare anche apposite strutture che le gestiscano (Curry, in fin dei conti, è un'estensione di Haskell), mostreremo come, con modifiche minimali al programma, i due linguaggi possano produrre i medesimi risultati su termini chiusi.

- per far restituire a Curry gli stessi risultati di Haskell è sufficiente cancellare la terza regola
- per far restituire ad Haskell gli stessi risultati di Curry occorre riscrivere la seconda regola in modo che non “oscuri” la terza. Una possibile riscrittura potrebbe essere:

```
h x (N y _ _) Left z | 3*z == 2+y = True
```


Capitolo 5

Quinto esercizio

5.1 Consegnna

Rappresentando Alberi “generici” con i costruttori `void/0` e `node/2`, dove `node` mantiene il dato di un nodo e la lista dei figli, si scriva un predicato `smallParents` che restituisce la lista dei (valori dei) nodi che sono genitori *ma non nonni* di qualche altro nodo. La lista deve essere prodotta rispettando l’ordine di comparizione nell’albero.

Ad esempio

`smallParents(node(1, [node(2, []), node(3, [node(5, [])]), node(4, [])]), Xs)`

restituisce `Xs = [3]`.

Si discuta cosa potrebbe essere fatto (se possibile) per far funzionare il precedente predicato su termini non necessariamente ground.

5.2 Svolgimento

5.2.1 Predicato smallParents

```
%stabilisce se un nodo è foglia
isLeaf( node( _ , [ ] ) ).
```

```
%stabilisce se una lista di nodi è una lista di foglie
otherLeaves( [ ] ).
otherLeaves( [ N | NS ] ) :- isLeaf( N ), otherLeaves( NS ).
```

```
areLeaves( [ N | NS ] ) :- isLeaf( N ), otherLeaves( NS ).
```

```
childList( [ ] , [ ] ).
childList( [ N | NS ] , SP ) :- smallParents( N, S1 ),
                                childList( NS, S2 ), append( S1, S2, SP ).
```

```
%lista dei genitori delle foglie
smallParents( void, [ ] ).
smallParents( node( _ , [ ] ), [ ] ).
smallParents( node( X, XS ) , SP ) :-  
    areLeaves( XS ) -> append( [ X ], [ ], SP );
                                childList( XS, SP ).
```

5.2.2 Esecuzione

Il programma funziona correttamente su termini ground.

Per quanto riguarda, invece, argomenti non ground, esso non ha problemi a generare la lista dei genitori di basso rango (nodi che sono genitori ma non nonni di qualche altro nodo) dato un albero, mentre non può (chiaramente) fare il contrario, dal momento che data una lista di etichette è possibile costruire infiniti alberi tali da soddisfare il predicato smallParents rispetto a tale lista.

Il programma è presente nella cartella `Esercizio5\esercizio5.pl`.

Capitolo 6

Sesto esercizio

6.1 Consegnna

Si vuole risolvere il seguente gioco matematico. Data una matrice M di numeri interi $n \times 5$ ed un numero intero “goal” v_g , si deve restituire un’espressione aritmetica “sequenziale” della forma

$$(((v_1 \ o_1 \ v_2) \ o_2 \ v_3) \ o_3 \ v_4) \ o_4 \ v_5)$$

dove $\{o1, \dots, o4\} = \{+, -, \cdot, \div\}$ mentre i v_i sono 5 numeri ognuno scelto da una colonna *distinta* di M (cioè non devono esistere $i \neq j, k, p \neq q$ per cui $v_i = M_{pk}$ e $v_j = M_{qk}$).

L’espressione cercata deve essere quella, fra tutte le espressioni sequenziali, il cui valore si avvicina maggiormente, in valore assoluto, al goal v_g (in caso di più espressioni con medesimo valore una qualsiasi).

L’operazione \div va calcolata in virgola fissa, arrotondando al secondo decimale (e quindi tutti i valori intermedi dell’espressione son pure da considerare in virgola fissa).

Si faccia attenzione al fatto che l’espressione $e = (167 \div 331) \cdot 374 - 379 + 248$, a causa di questa regola, non valuta a $57.6948\dots$ ma a 56 e quindi e è una soluzione esatta per il goal 56 e la matrice

$$\begin{pmatrix} 36 & 49 & 79 & 20 & 22 \\ 174 & 85 & 189 & \mathbf{167} & 187 \\ \mathbf{248} & 210 & 234 & 293 & 298 \\ 314 & \mathbf{374} & \mathbf{379} & 382 & \mathbf{331} \end{pmatrix}$$

Codificando le espressioni aritmetiche mediante il tipo di dato

```
data Expr = N Int — O Op Expr Int
data Op = Add — Mul — Sub — Div
```

si scriva una funzione Curry che dato un intero v_g e la matrice M, codificata come si ritiene più opportuno, restituisce la soluzione (o una qualunque tra quelle equivalenti).

6.2 Svolgimento

Il programma si appoggia sulla funzione “`buildCandidateSol`” che partendo dalla matrice di input (espressa come lista di colonne, ogni colonna è lista di interi), grazie al non determinismo, genera tutte le espressioni (Expr) possibili, permutando le colonne, le quattro operazioni aritmetiche e scegliendo un termine per colonna.

Dal momento che lo spazio di ricerca delle soluzioni è estremamente vasto ($n^5 \cdot 4! \cdot 5!$) si è resa necessaria la creazione di una funzione (`candidateSolList`) che navighi tale spazio, suddividendolo in classi di equivalenza rispetto alla distanza in valore assoluto dal goal.

Durante la ricerca le classi di equivalenza più distanti dall’obiettivo, rispetto a quelle già incontrate, non vengono aggiunte alla lista. La lista prodotta non sarà, quindi, la lista di tutte le classi di equivalenza presenti nello spazio delle soluzioni ma un sottoinsieme di classi “vicine” ordinate in maniera decrescente (l’ultima è la più vicina).

Le classi di equivalenza restituite dipendono, quindi, dalla prima classe incontrata: verranno aggiunte in coda solo quelle via via più vicine al goal.

6.2.1 Soluzione

La soluzione restituita dal programma (“`solve`”) è un rappresentante della classe di equivalenza più vicina, in valore assoluto, all’obiettivo (ossia l’ultimo elemento della lista generata da `candidateSolList`).

In tal modo, data la matrice fornita come esempio nel testo d’esame

$[[36,174,248,314],[49,85,210,374],[79,189,234,379],[20,167,293,382],[22,187,298,331]]$

e goal pari a 56, il programma ritorna per soluzione l’espressione:

$$ODiv(OSub(OMul(OAdd(N293)298)36)49)379$$

che valutata in virgola fissa equivale proprio a 56.

Utilizzando la stessa matrice, ma richiedendo come obiettivo l’intero negativo -1001, il programma ritorna l’espressione

$$OAdd(OMul(ODiv(OSub(N174)234)22)374)20$$

che valutata equivale a -1001,02.

Il programma è presente nella cartella `Esercizio6\esercizio6.curry`.

Capitolo 7

Settimo esercizio

7.1 Consegnna

Data la grammatica

$$\begin{aligned} S &\rightarrow (\ L \) \mid E \\ L &\rightarrow E \mid L , \ S \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T * \ a \mid a \end{aligned}$$

1. Costruire il parser top-down.
2. Mostrare l'esecuzione del suddetto parsers sull'input $(\ a \ + \ a \ , \ + \)$, utilizzando opportuni passi di error recovery.

7.2 Svolgimento

7.2.1 Costruzione del parser top-down

Questa grammatica non è ambigua, non necessita di fattorizzazione sinistra ma presenta ricorsioni sinistre; pertanto ripresentiamo la grammatica modificata:

$$\begin{aligned} S &\rightarrow (L) \mid E \\ L &\rightarrow E L' \\ L' &\rightarrow, S \mid \epsilon \\ E &\rightarrow T E' \\ E' &\rightarrow +T \mid \epsilon \\ T &\rightarrow a T' \\ T' &\rightarrow * a \mid \epsilon \end{aligned}$$

A questo punto calcoliamo gli insiemi FIRST e FOLLOW.

$$\begin{array}{ll} FIRST(S) = \{ (a \} & FOLLOW(S) = \{ \$ \} \\ FIRST(L) = \{ a \} & FOLLOW(L) = \{) \} \\ FIRST(L') = \{ , \epsilon \} & FOLLOW(L') = \{) \} \\ FIRST(E) = \{ a \} & FOLLOW(E) = \{ \$ \} , \} \\ FIRST(E') = \{ + \epsilon \} & FOLLOW(E') = \{ \$ \} , \} \\ FIRST(T) = \{ a \} & FOLLOW(T) = \{ + \$ \} , \} \\ FIRST(T') = \{ * \epsilon \} & FOLLOW(T') = \{ + \$ \} , \} \end{array}$$

Ora, sfruttando le informazioni fornite dagli insiemi FIRST e FOLLOW appena calcolati, procediamo con la costruzione della tabella di parsing predittivo, presente in figura 7.1.

| Non terminale | Simbolo d'ingresso | | | | | | |
|---------------|---------------------|---------------------------|-------------------------|---------------------------|---------------------------|----------------------|---------------------------|
| | (|) | * | + | , | a | \$ |
| S | $S \rightarrow (L)$ | | | | | $S \rightarrow E$ | |
| L | | | | | | $L \rightarrow E L'$ | |
| L' | | $L' \rightarrow \epsilon$ | | | $L' \rightarrow , S L'$ | | |
| E | | | | | | $E \rightarrow T E'$ | |
| E' | | $E' \rightarrow \epsilon$ | $E' \rightarrow + T E'$ | $E' \rightarrow \epsilon$ | | | $E' \rightarrow \epsilon$ |
| T | | | | $T \rightarrow \epsilon$ | $T \rightarrow \epsilon$ | $T \rightarrow a T'$ | |
| T' | | $T' \rightarrow \epsilon$ | $T' \rightarrow * a T'$ | $T' \rightarrow \epsilon$ | $T' \rightarrow \epsilon$ | | $T' \rightarrow \epsilon$ |

Figura 7.1: Tabella di parsing predittivo

7.2.2 Esecuzione del parser

Considerando l'input ($a + a$, $+$) andiamo ora a effettuarne la valutazione. In fig.7.2 sono mostrate le mosse del parser predittivo.

| Riconosciuta | Stack | Input | Azione |
|--------------|---------------|-----------------|----------------------------------|
| | $S \$$ | $(a + a, +) \$$ | output $S \rightarrow (L)$ |
| | $(L) \$$ | $(a + a, +) \$$ | consuma (|
| (| $L) \$$ | $a + a, +) \$$ | output $L \rightarrow EL'$ |
| (| $EL') \$$ | $a + a, +) \$$ | output $E \rightarrow TE'$ |
| (| $TE'L') \$$ | $a + a, +) \$$ | output $T \rightarrow aT'$ |
| (a | $aT'E'L') \$$ | $a + a, +) \$$ | consuma a |
| (a | $T'E'L') \$$ | $+ a, +) \$$ | output $T' \rightarrow \epsilon$ |
| (a | $E'L') \$$ | $+ a, +) \$$ | output $E' \rightarrow +TE'$ |
| (a+ | $+TE'L') \$$ | $+ a, +) \$$ | consuma + |
| (a+ | $TE'L') \$$ | $a, +) \$$ | $T \rightarrow aT'$ |
| (a+a | $aT'E'L') \$$ | $a, +) \$$ | consuma a |
| (a+a | $T'E'L') \$$ | $, +) \$$ | output $T' \rightarrow \epsilon$ |
| (a+a | $E'L') \$$ | $, +) \$$ | output $E' \rightarrow \epsilon$ |
| (a+a | $L') \$$ | $, +) \$$ | $L' \rightarrow , SL'$ |
| (a+a, | $, SL') \$$ | $, +) \$$ | consuma , |
| (a+a, | $SL') \$$ | $+) \$$ | errore |

Figura 7.2: Tabella di esecuzione del parser

Osserviamo che si è verificato un errore durante la valutazione. Per poter effettuare comunque il parsing sull'input è stato deciso di gestire gli errori, utilizzando una strategia di error recovery basata su routines. Aggiorniamo a tale scopo la tabella di parsing predittivo aggiungendo dei simboli en,

che rappresentano l' n -esima routine definita dove necessario per risolvere la situazione.

La tabella avrà quindi la struttura visibile in figura 7.3.

| Non terminale | Simbolo d'ingresso | | | | | | |
|---------------|-----------------------|---------------------------|-------------------------|--------------------------|---------------------------|---------------------------|----|
| | (|) | * | + | , | a | \$ |
| S | $S \rightarrow (L)$ | e2 | | e1 | | $S \rightarrow E$ | |
| L | | | | | | $L \rightarrow E L'$ | |
| L' | | $L' \rightarrow \epsilon$ | | | $L' \rightarrow , S L'$ | | |
| E | | | | | | $E \rightarrow T E'$ | |
| E' | | $E' \rightarrow \epsilon$ | | $E' \rightarrow + T E'$ | $E' \rightarrow \epsilon$ | $E' \rightarrow \epsilon$ | |
| T | | | | $T \rightarrow \epsilon$ | $T \rightarrow \epsilon$ | $T \rightarrow a T'$ | |
| T' | | $T' \rightarrow \epsilon$ | $T' \rightarrow * a T'$ | | $T' \rightarrow \epsilon$ | $T' \rightarrow \epsilon$ | |

Figura 7.3: Tabella di parsing predittivo, con gestione degli errori

Definiamo ora le routines di cui abbiamo inserito la sigla nella tabella di parsing predittivo.

e1: ignora il token.

e2: effettua un inserimento del simbolo a, produce il messaggio “simbolo a mancante” e rimuove S dallo stack.

A questo punto è possibile effettuare di nuovo le stesse mosse di parsing, risolvendo il precedente errore tramite le routine di error recovery appena introdotte.

La strategia qui adottata nella gestione degli errori, ha cercato di eliminare il meno possibile di quanto sia già stato riconosciuto dal parser, assumendo quindi l’errato inserimento del simbolo + invece della simbolo a.

La valutazione in questo modo può perciò procedere fino al termine (vedi fig.7.4).

| Riconosciuta | Stack | Input | Azione |
|---------------|-----------------|------------------|------------------|
| | \$ \$ | (a + a , +) \$ | output S → (L) |
| | (L) \$ | (a + a , +) \$ | consuma (|
| (| L) \$ | a + a , +) \$ | output L → EL' |
| (| E L') \$ | a + a , +) \$ | output E → TE' |
| (| T E' L') \$ | a + a , +) \$ | output T → aT' |
| (a | a T' E' L') \$ | a + a , +) \$ | consuma a |
| (a | T' E' L') \$ | + a , +) \$ | output T' → ε |
| (a | E' L') \$ | + a , +) \$ | output E' → +TE' |
| (a + | + T E' L') \$ | + a , +) \$ | consuma + |
| (a + | T E' L') \$ | a , +) \$ | T → aT' |
| (a + a | a T' E' L') \$ | a , +) \$ | consuma a |
| (a + a | T' E' L') \$ | , +) \$ | output T' → ε |
| (a + a | E' L') \$ | , +) \$ | output E' → ε |
| (a + a | L') \$ | , +) \$ | L' → , \$ L' |
| (a + a , | , \$ L') \$ | , +) \$ | consuma , |
| (a + a , | \$ L') \$ | +) \$ | e1 |
| (a + a , | \$ L') \$ |) \$ | e2 |
| (a + a , a | L') \$ |) \$ | L' → ε |
| (a + a , a) |) \$ |) \$ | consuma) |
| (a + a , a) | \$ | \$ | accept |

Figura 7.4: Tabella di esecuzione del parser, con gestione degli errori

Capitolo 8

Ottavo esercizio

8.1 Consegnna

Data la grammatica

$$\begin{aligned} S &\rightarrow S + C \mid C \\ C &\rightarrow C . K \mid K \\ K &\rightarrow (S) \mid K * a \mid b \mid c \end{aligned}$$

1. Costruire i parsers SLR e LALR.
2. Mostrare l'esecuzione di suddetti parsers sull'input

c . + (b * + c

utilizzando opportuni passi di error recovery.

8.2 Svolgimento

8.2.1 Costruzione del parser SLR

Ci occupiamo per prima cosa di costruire il parser SLR. Per poter fare ciò, la prima operazione da effettuare è quella di costruire una grammatica aumentata G' , sulla base di G data nella consegna.

Aggiungiamo quindi la produzione

$$S' \rightarrow S$$

con S' nuovo simbolo iniziale: indica al parser quando terminare.

L'insieme finale di produzioni della grammatica G' è quindi il seguente:

1. $S' \rightarrow S$
2. $S \rightarrow S + C$
3. $S \rightarrow C$
4. $C \rightarrow C . K$
5. $C \rightarrow K$
6. $K \rightarrow (S)$
7. $K \rightarrow K *$
8. $K \rightarrow a$
9. $K \rightarrow b$
10. $K \rightarrow c$

Calcoliamo a questo punto gli insiemi FIRST e FOLLOW. In particolare gli insiemi FOLLOW ci serviranno nella costruzione della tabella SLR.

$$\begin{array}{ll} FIRST(S') = \{ (, b , c \} & FOLLOW(S') = \{ \$ \} \\ FIRST(S) = \{ (, b , c \} & FOLLOW(S) = \{ \$, + ,) \} \\ FIRST(C) = \{ (, b , c \} & FOLLOW(C) = \{ \$, + ,) , . \} \\ FIRST(K) = \{ (, b , c \} & FOLLOW(K) = \{ \$, + ,) , . , * \} \end{array}$$

A partire dall'insieme di produzioni sopra elencate, procediamo quindi al calcolo della collezione $C = \{I_0, I_1, \dots, I_n\}$ degli insiemi di item $LR(0)$ di G' .

- $I_0 = \text{Closure}(S' \rightarrow \cdot S) =$
 $S' \rightarrow \cdot S$
 $S \rightarrow \cdot S + C$
 $S \rightarrow \cdot C$
 $C \rightarrow \cdot C . K$
 $C \rightarrow \cdot K$
 $K \rightarrow \cdot (S)$
 $K \rightarrow \cdot K * a$
 $K \rightarrow \cdot b$
 $K \rightarrow \cdot c$
- $I_1 = \text{Goto}(I_0, S) =$
 $S' \rightarrow S \cdot$
 $S \rightarrow S \cdot + C$
- $I_2 = \text{Goto}(I_0, C) =$
 $S \rightarrow C \cdot$
 $C \rightarrow C \cdot . K$
- $I_3 = \text{Goto}(I_0, K) =$
 $C \rightarrow K \cdot$
 $K \rightarrow K \cdot * a$
- $I_4 = \text{Goto}(I_0, ()) =$
 $K \rightarrow (\cdot S)$
 $S \rightarrow \cdot S + C$
 $S \rightarrow \cdot C$
 $C \rightarrow \cdot C . K$
 $C \rightarrow \cdot K$
 $K \rightarrow \cdot (S)$
 $K \rightarrow \cdot K * a$
 $K \rightarrow \cdot b$
 $K \rightarrow \cdot c$
- $I_5 = \text{Goto}(I_0, b) =$
 $K \rightarrow b \cdot$
- $I_6 = \text{Goto}(I_0, c) =$
 $K \rightarrow c \cdot$

- $I_7 = Goto(I_1, +) =$

$$S \rightarrow S + \cdot C$$

$$C \rightarrow \cdot C . K$$

$$C \rightarrow \cdot K$$

$$K \rightarrow \cdot (S)$$

$$K \rightarrow \cdot K * a$$

$$K \rightarrow \cdot b$$

$$K \rightarrow \cdot c$$

- $I_8 = Goto(I_2, .) =$

$$C \rightarrow C . \cdot K$$

$$K \rightarrow \cdot (S)$$

$$K \rightarrow \cdot K * a$$

$$K \rightarrow \cdot b$$

$$K \rightarrow \cdot c$$

- $I_9 = Goto(I_3, *) =$

$$K \rightarrow K * \cdot a$$

- $I_{10} = Goto(I_4, S) =$

$$K \rightarrow (S \cdot)$$

$$S \rightarrow S \cdot + C$$

- $I_{11} = Goto(I_7, C) =$

$$S \rightarrow S + C \cdot$$

$$C \rightarrow C \cdot . K$$

- $I_{12} = Goto(I_8, K) =$

$$C \rightarrow C . K \cdot$$

$$K \rightarrow K \cdot * a$$

- $I_{13} = Goto(I_9, a) =$

$$K \rightarrow K * a \cdot$$

- $I_{14} = Goto(I_{10}, ()) =$

$$K \rightarrow (S) \cdot$$

Sulla base di questa collezione, siamo in grado ora di creare la tabella di parsing SLR relativa alla grammatica G' , presentata in figura 8.1.

| Stato | ACTION | | | | | | | | GOTO | | | |
|-------|--------|-----|----|-----|----|----|----|--------|------|----|----|---|
| | (| * | + | . | a | b | c | \$ | S' | S | C | K |
| 0 | s4 | | | | | s5 | s6 | | | 1 | 2 | 3 |
| 1 | | | s7 | | | | | accept | | | | |
| 2 | | r2 | | r2 | s8 | | | | r2 | | | |
| 3 | | r4 | s9 | r4 | r4 | | | | r4 | | | |
| 4 | s4 | | | | | s5 | s6 | | | 10 | 2 | 3 |
| 5 | | r7 | r7 | r7 | r7 | | | | r7 | | | |
| 6 | | r8 | r8 | r8 | r8 | | | | r8 | | | |
| 7 | s4 | | | | | s5 | s6 | | | 11 | 3 | |
| 8 | s4 | | | | | s5 | s6 | | | | 12 | |
| 9 | | | | s13 | | | | r6 | | | | |
| 10 | | s14 | | s7 | | | | | | | | |
| 11 | | r1 | | r1 | s8 | | | | r1 | | | |
| 12 | | r3 | s9 | r3 | r3 | | | | r3 | | | |
| 13 | | r6 | r6 | r6 | r6 | | | | r6 | | | |
| 14 | | r5 | r5 | r5 | r5 | | | | r5 | | | |

Figura 8.1: Tabella di parsing SLR

8.2.2 Esecuzione del parser SLR

Considerando l'input

c . + (b * + c

andiamo ora a effettuarne la valutazione.

Nella tabella in figura 8.2 sono mostrate le mosse del parser SLR.

| Passo | Stack | Simboli | Input | Azione |
|-------|-------|---------|--------------------|--------------|
| (1) | 0 | | c . + (b * + c \$ | Shift |
| (2) | 0 6 | c | . + (b * + c \$ | Reduce K → c |
| (3) | 0 3 | K | . + (b * + c \$ | Reduce C → K |
| (4) | 0 2 | C | . + (b * + c \$ | Shift |
| (5) | 0 2 8 | C. | + (b * + c \$ | errore |

Figura 8.2: Tabella di esecuzione del parser SLR

Osserviamo dalla precedente tabella il fallimento nel parsing dell'input fornito. E' necessario perciò gestire tali errori, utilizzando una strategia di error recovery basata su routines. Aggiorniamo a tale scopo la tabella ACTION aggiungendo dei simboli en , che rappresentano l' n -esima routine definita.

| Stato | ACTION | | | | | | | GOTO | | | | |
|-------|--------|-----|----|----|----|-----|----|------|--------|----|----|----|
| | (|) | * | + | . | a | b | c | \$ | S' | S | C |
| 0 | s4 | | | | | | s5 | s6 | | 1 | 2 | 3 |
| 1 | | | | s7 | | | | | accept | | | |
| 2 | | r2 | | r2 | s8 | | | | | r2 | | |
| 3 | | r4 | s9 | r4 | r4 | | | | | r4 | | |
| 4 | s4 | | | | | | s5 | s6 | | 10 | 2 | 3 |
| 5 | | r7 | r7 | r7 | r7 | | | | | r7 | | |
| 6 | | r8 | r8 | r8 | r8 | | | | | r8 | | |
| 7 | s4 | | | | | | s5 | s6 | | | 11 | 3 |
| 8 | s4 | | | e1 | | | s5 | s6 | | | | 12 |
| 9 | | | | e2 | | s13 | | | | r6 | | |
| 10 | | s14 | | s7 | | | | | | e3 | | |
| 11 | | r1 | | r1 | s8 | | | | | r1 | | |
| 12 | | r3 | s9 | r3 | r3 | | | | | r3 | | |
| 13 | | r6 | r6 | r6 | r6 | | | | | r6 | | |
| 14 | | r5 | r5 | r5 | r5 | | | | | r5 | | |

Figura 8.3: Tabella di parsing SLR, con gestione degli errori

Definiamo ora le routines di cui abbiamo inserito la sigla nella tabella ACTION.

e1: elimina . dai simboli riconosciuti dal parser e effettua un $pop(8)$.

e2: effettua un inserimento del simbolo a, produce il messaggio “simbolo a mancante” ed effettua uno $shift\ 13$.

e3: effettua un inserimento del simbolo), produce il messaggio “Parentesi tonda mancante” ed effettua uno $shift\ 14$.

A questo punto, rivalutiamo l'input gestendo gli errori come appena descritto. Nella tabella in figura 8.4 sono mostrate le mosse del parser SLR, comprensive di gestione degli errori. La strategia qui adottata nella gestione degli errori, ha cercato di eliminare il meno possibile di quanto sia già stato riconosciuto dal parser, assumendo quindi la mancanza di alcuni simboli nell'input.

| Passo | Stack | Simboli | Input | Azione |
|-------|-------------------|-------------|--------------------|----------------|
| (1) | 0 | | c . + (b * + c \$ | Shift |
| (2) | 0 6 | c | + (b * + c \$ | Reduce K → c |
| (3) | 0 3 | K | + (b * + c \$ | Reduce C → K |
| (4) | 0 2 | C | + (b * + c \$ | Shift |
| (5) | 0 2 8 | C . | + (b * + c \$ | e1 |
| (6) | 0 2 | C | + (b * + c \$ | Reduce S → C |
| (7) | 0 1 | S | + (b * + c \$ | Shift |
| (8) | 0 1 7 | S + | (b * + c \$ | Shift |
| (9) | 0 1 7 4 | S + (| b * + c \$ | Shift |
| (10) | 0 1 7 4 5 | S + (b | * + c \$ | Reduce K → b |
| (11) | 0 1 7 4 5 | S + (K | * + c \$ | Shift |
| (12) | 0 1 7 4 3 9 | S + (K * | + c \$ | e2 |
| (13) | 0 1 7 4 3 9 13 | S + (K * a | + c \$ | Reduce K → K*a |
| (14) | 0 1 7 4 3 | S + (K | + c \$ | Reduce C → K |
| (15) | 0 1 7 4 2 | S + (C | + c \$ | Reduce S → C |
| (16) | 0 1 7 4 1 0 | S + (S | + c \$ | Shift |
| (17) | 0 1 7 4 1 0 7 | S + (S + | c \$ | Shift |
| (18) | 0 1 7 4 1 0 7 6 | S + (S + c | \$ | Reduce K → c |
| (19) | 0 1 7 4 1 0 7 3 | S + (S + K | \$ | Reduce C → K |
| (20) | 0 1 7 4 1 0 7 1 1 | S + (S + C | \$ | Reduce S → S+C |
| (21) | 0 1 7 4 1 0 | S + (S | \$ | e3 |
| (22) | 0 1 7 4 1 0 1 4 | S + (S) | \$ | Reduce K → (S) |
| (23) | 0 1 7 3 | S + K | \$ | Reduce C → K |
| (24) | 0 1 7 1 1 | S + C | \$ | Reduce S → S+C |
| (25) | 0 1 | S | \$ | Accept |

Figura 8.4: Tabella di esecuzione del parser SLR, con gestione degli errori

8.2.3 Costruzione ed esecuzione del parser LALR

La costruzione del parser LALR, avviene in maniera analoga, innanzitutto calcolando la collezione $C = \{I_0, I_1, \dots, I_n\}$ degli insiemi di item $LR(1)$ di G' .

- $I_0 = \text{Closure}(S' \rightarrow \cdot S, \$) =$

$$\begin{aligned} & S' \rightarrow \cdot S, \$ \\ & S \rightarrow \cdot S + C, \$ + \\ & S \rightarrow \cdot C, \$ + \\ & C \rightarrow \cdot C . K, \$ + . \\ & C \rightarrow \cdot K, \$ + . \\ & K \rightarrow \cdot (S), \$ + . * \\ & K \rightarrow \cdot K * a, \$ + . * \\ & K \rightarrow \cdot b, \$ + . * \\ & K \rightarrow \cdot c, \$ + . * \end{aligned}$$

- $I_1 = Goto(I_0, S) =$

$$\begin{aligned} & S' \rightarrow S \cdot , \$ \\ & S \rightarrow S \cdot + C, \$ + \end{aligned}$$

- $I_2 = Goto(I_0, C) =$

$$\begin{aligned} & S \rightarrow C \cdot , \$ + \\ & C \rightarrow C \cdot . K, \$ + . \end{aligned}$$

- $I_3 = Goto(I_0, K) =$

$$\begin{aligned} & C \rightarrow K \cdot , \$ + . \\ & K \rightarrow K \cdot * a, \$ + . * \end{aligned}$$

- $I_4 = Goto(I_0, ()) =$

$$\begin{aligned} & K \rightarrow (\cdot S), \$ + . * \\ & S \rightarrow \cdot S + C,) + \\ & S \rightarrow \cdot C,) + \\ & C \rightarrow \cdot C . K,) + . \\ & C \rightarrow \cdot K,) + . \\ & K \rightarrow \cdot (S),) + . * \\ & K \rightarrow \cdot K * a,) + . * \\ & K \rightarrow \cdot b,) + . * \\ & K \rightarrow \cdot c,) + . * \end{aligned}$$

- $I_5 = Goto(I_0, b) =$

$$K \rightarrow b \cdot , \$ + . *$$

- $I_6 = Goto(I_0, c) =$

$$K \rightarrow c \cdot , \$ + . *$$

- $I_7 = Goto(I_1, +) =$
 $S \rightarrow S + \cdot C, \$ +$
 $C \rightarrow \cdot C . K, \$ + .$
 $C \rightarrow \cdot K, \$ + .$
 $K \rightarrow \cdot (S), \$ + . *$
 $K \rightarrow \cdot K * a, \$ + . *$
 $K \rightarrow \cdot b, \$ + . *$
 $K \rightarrow \cdot c, \$ + . *$

- $I_8 = Goto(I_2, .) =$
 $C \rightarrow C . \cdot K, \$ + .$
 $K \rightarrow \cdot (S), \$ + . *$
 $K \rightarrow \cdot K * a, \$ + . *$
 $K \rightarrow \cdot b, \$ + . *$
 $K \rightarrow \cdot c, \$ + . *$

- $I_9 = Goto(I_3, *) =$
 $K \rightarrow K * \cdot a, \$ + . *$

- $I_{10} = Goto(I_4, S) =$
 $K \rightarrow (S \cdot), \$ + . *$
 $S \rightarrow S \cdot + C,) +$

- $I_{11} = Goto(I_4, C) =$
 $S \rightarrow C \cdot ,) +$
 $C \rightarrow C \cdot . K,) + .$

- $I_{12} = Goto(I_4, K) =$
 $C \rightarrow K \cdot ,) + .$
 $K \rightarrow K \cdot * a,) + . *$

- $I_{13} = Goto(I_4, ()) =$
 $K \rightarrow (\cdot S),) + . *$
 $S \rightarrow \cdot S + C,) +$
 $S \rightarrow \cdot C,) +$
 $C \rightarrow \cdot C . K,) + .$
 $C \rightarrow \cdot K,) + .$
 $K \rightarrow \cdot (S),) + . *$
 $K \rightarrow \cdot K * a,) + . *$
 $K \rightarrow \cdot b,) + . *$
 $K \rightarrow \cdot c,) + . *$

- $I_{14} = Goto(I_4, b) =$
 $K \rightarrow b \cdot ,) + .*$
- $I_{15} = Goto(I_4, c) =$
 $K \rightarrow c \cdot ,) + .*$
- $I_{16} = Goto(I_7, C) =$
 $S \rightarrow S + C \cdot , \$ +$
 $C \rightarrow C \cdot . K, \$ + .$
- $I_{17} = Goto(I_8, K) =$
 $C \rightarrow C . K \cdot , \$ + .$
 $K \rightarrow K \cdot * a, \$ + .*$
- $I_{18} = Goto(I_9, a) =$
 $K \rightarrow K * a \cdot , \$ + .*$
- $I_{19} = Goto(I_{10},)) =$
 $K \rightarrow (S) \cdot , \$ + .*$
- $I_{20} = Goto(I_{10}, +) =$
 $S \rightarrow S + \cdot C,) +$
 $C \rightarrow \cdot C . K,) + .$
 $C \rightarrow \cdot K,) + .$
 $K \rightarrow \cdot (S),) + .*$
 $K \rightarrow \cdot K * a,) + .*$
 $K \rightarrow \cdot b,) + .*$
 $K \rightarrow \cdot c,) + .*$
- $I_{21} = Goto(I_{11}, ..) =$
 $C \rightarrow C . \cdot K,) + .$
 $K \rightarrow \cdot (S),) + .*$
 $K \rightarrow \cdot K * a,) + .*$
 $K \rightarrow \cdot b,) + .*$
 $K \rightarrow \cdot c,) + .*$
- $I_{22} = Goto(I_{12}, *) =$
 $K \rightarrow K * \cdot a,) + .*$
- $I_{23} = Goto(I_{13}, S) =$
 $K \rightarrow (S \cdot),) + .*$
 $S \rightarrow S \cdot + C,) +$

- $I_{24} = Goto(I_{20}, C) =$
 $S \rightarrow S + C \cdot ,) +$
 $C \rightarrow C \cdot . K,) + .$
- $I_{25} = Goto(I_{21}, K) =$
 $C \rightarrow C . K \cdot ,) + .$
 $K \rightarrow K \cdot * a,) + . *$
- $I_{26} = Goto(I_{22}, a) =$
 $K \rightarrow K * a \cdot ,) + . *$
- $I_{27} = Goto(I_{23},)) =$
 $K \rightarrow (S) \cdot ,) + . *$

Uniamo ora gli item con core comune.

- $I_{211} = I_2 \cup I_{11} =$
 $S \rightarrow C \cdot ,) + \$$
 $C \rightarrow C \cdot . K,) + . \$$
- $I_{312} = I_3 \cup I_{12} =$
 $C \rightarrow K \cdot ,) + . \$$
 $K \rightarrow K \cdot * a,) + . * \$$
- $I_{413} = I_4 \cup I_{13} =$
 $K \rightarrow (\cdot S),) + . * \$$
 $S \rightarrow \cdot S + C,) +$
 $S \rightarrow \cdot C,) +$
 $C \rightarrow \cdot C . K,) + .$
 $C \rightarrow \cdot K,) + .$
 $K \rightarrow \cdot (S),) + . *$
 $K \rightarrow \cdot K * a,) + . *$
 $K \rightarrow \cdot b,) + . *$
 $K \rightarrow \cdot c,) + . *$
- $I_5 = I_5 \cup I_{14} =$
 $K \rightarrow b \cdot ,) + . * \$$
- $I_{615} = I_6 \cup I_{15} =$
 $K \rightarrow c \cdot ,) + . * \$$
- $I_{720} = I_7 \cup I_{20} =$
 $S \rightarrow S + \cdot C,) + \$$
 $C \rightarrow \cdot C . K,) + . \$$

$$\begin{aligned} C &\rightarrow \cdot K,) + . \$ \\ K &\rightarrow \cdot (S),) + . * \$ \\ K &\rightarrow \cdot K * a,) + . * \$ \\ K &\rightarrow \cdot b,) + . * \$ \\ K &\rightarrow \cdot c,) + . * \$ \end{aligned}$$

- $I_{821} = I_8 \cup I_{21} =$

$$\begin{aligned} C &\rightarrow C. \cdot K,) + . \$ \\ K &\rightarrow \cdot (S),) + . * \$ \\ K &\rightarrow \cdot K * a,) + . * \$ \\ K &\rightarrow \cdot b,) + . * \$ \\ K &\rightarrow \cdot c,) + . * \$ \end{aligned}$$
- $I_{922} = I_9 \cup I_{22} =$

$$K \rightarrow K * \cdot a,) + . * \$$$
- $I_{1023} = I_{10} \cup I_{23} =$

$$\begin{aligned} K &\rightarrow (S \cdot),) + . * \$ \\ S &\rightarrow S \cdot + C,) + \end{aligned}$$
- $I_{1624} = I_{16} \cup I_{24} =$

$$\begin{aligned} S &\rightarrow S + C \cdot ,) + \$ \\ C &\rightarrow C \cdot . K,) + . \$ \end{aligned}$$
- $I_{1725} = I_{17} \cup I_{25} =$

$$\begin{aligned} C &\rightarrow C. K \cdot ,) + . \$ \\ K &\rightarrow K \cdot * a,) + . * \$ \end{aligned}$$
- $I_{1826} = I_{18} \cup I_{26} =$

$$K \rightarrow K * a \cdot ,) + . * \$$$
- $I_{1927} = I_{19} \cup I_{27} =$

$$K \rightarrow (S) \cdot ,) + . * \$$$

Alla luce della collezione appena ottenuta, a seguito delle unioni degli item con core comune, osserviamo come gli item per la costruzione della tabella SLR e LALR coincidano.

Questo risultato ci permette di fermarci qui, in quanto la tabella risultante sarà la stessa per entrambe le tipologie di parser. Discorso analogo per l'esecuzione dell'input: anche gli errori visti in precedenza vengono gestiti allo stesso modo.

Capitolo 9

Nono esercizio

9.1 Consegnna

Si consideri un linguaggio costruito a partire dalla stessa sintassi concreta di Python (<http://python.org/>).

In tale linguaggio, contrariamente a Python, i parametri formali di procedura/funzioni vengono dichiarati con modalità (default per valore opzionale) e tipi esplicativi (per le procedure si usi il tipo unitario di Python).

Si hanno alcune procedure predefinite (`print` | `read`) (`Int` | `Float` | `Char` | `String`) oltre alle operazioni aritmetiche, booleane e relazionali standard. Si possono dichiarare funzioni, oltre che variabili, all'interno di qualsiasi blocco.

I tipi ammessi siano interi, booleani, float, caratteri, stringhe, array e puntatori (con i costruttori e/o le operazioni di selezione relativi).

Non si hanno tipi di dato definiti dall'utente.

Il linguaggio deve implementare le modalità di passaggio dei parametri `value` e `value-result`.

Al linguaggio si aggiunga un rudimentale meccanismo di gestione delle eccezioni, con sintassi concreta `try Statement catch Statement`, che non prevede situazioni sofisticate come in Java per cui un'eccezione all'interno di una chiamata di procedura fatta nel corpo di un `try` venga gestita dall'handler del chiamante.

Si aggiunga alle istruzioni del three-address un'istruzione `onexception goto Label`.

Per detto linguaggio:

1. Si progetti una opportuna sintassi astratta.
2. Si progetti un type-system.

3. Si scriva un parser.
4. Si implementi il type-checker e altri opportuni controlli di semantica statica (ad esempio il rilevamento di utilizzo di r-expr illegali nel passaggio dei parametri).
5. Si implementi il generatore di three-address code considerando che:
 - gli assegnamenti devono valutare l-value prima di r-value
 - l'ordine delle espressioni e della valutazione degli argomenti si può scegliere a piacere (motivandolo)
 - le espressioni booleane nelle guardie devono essere valutate con short-cut. In altri contesti si può scegliere a piacere (motivandolo).

9.2 Svolgimento

9.2.1 Grammatica

Viene presentata ora la struttura lessicale del nostro Pascal modificato. Queste sono le parole riservate:

```
Array    Boolean   Char
Double   Integer   Multiple
String   Unit      and
catch    def       del
elif     else      for
if       in        is
lambda  not       or
print   range     read
return  try       val
valres  while
```

Questi sono i simboli usati:

```
(      )    ==
!=    <=  >=
<    >    is not
not in +  -
*     /   %
//    **  ,
&    [   ]
:     ;   =
{     }
```

La grammatica è la seguente:

```
<Program> ::= <ListBlock>

<RExp> ::= <RExp1>
          | <RExp> or <RExp1>

<RExp1> ::= <RExp2>
          | <RExp1> and <RExp2>

<RExp2> ::= <RExp3>
          | not <RExp3>

<RExp3> ::= <RExp4>
          | <RExp4> == <RExp4>
          | <RExp4> != <RExp4>
          | <RExp4> <= <RExp4>
          | <RExp4> >= <RExp4>
          | <RExp4> < <RExp4>
          | <RExp4> > <RExp4>
          | <RExp4> is not <RExp4>
          | <RExp4> is <RExp4>
          | <RExp4> not in <RExp4>
          | <RExp4> in <RExp4>

<RExp4> ::= <RExp5>
          | <RExp4> + <RExp5>
          | <RExp4> - <RExp5>

<RExp5> ::= <RExp6>
          | <RExp5> * <RExp6>
          | <RExp5> / <RExp6>
          | <RExp5> % <RExp6>
          | <RExp5> // <RExp6>

<RExp6> ::= <RExp7>
          | + <RExp7>
          | - <RExp7>

<RExp7> ::= <RExp8>
          | <RExp7> ** <RExp8>
```

```

⟨RExp8⟩ ::= ⟨RExp9⟩
| range ( ⟨Integer⟩ , ⟨Integer⟩ )

⟨RExp9⟩ ::= ⟨RExp10⟩
| ⟨Integer⟩
| ⟨Double⟩
| ⟨Char⟩
| ⟨String⟩
| ⟨Boolean⟩

⟨RExp10⟩ ::= ( ⟨RExp⟩ )
| ⟨LExp⟩
| & ⟨Ident⟩

⟨RExpList⟩ ::= ⟨RExp⟩
| ⟨RExp⟩ , ⟨RExpList⟩

⟨RoutCall⟩ ::= ⟨Ident⟩ ( )
| ⟨Ident⟩ ( ⟨RExpList⟩ )

⟨LExp⟩ ::= ⟨Ident⟩
| ⟨Array⟩
| * ⟨Ident⟩

⟨ArValue⟩ ::= ⟨Integer⟩
| ε

⟨Array⟩ ::= [ ⟨RExpList⟩ ]
| ⟨Ident⟩ [ ⟨RExp⟩ ]
| ⟨Ident⟩ [ ⟨ArValue⟩ : ⟨ArValue⟩ ]
| ⟨Ident⟩ [ ⟨ArValue⟩ : ⟨ArValue⟩ : ⟨ArValue⟩ ]

⟨Statement⟩ ::= ⟨Assignement⟩ ;
| del ⟨RExp10⟩ ;
| print ⟨RExp⟩ ;
| read ⟨RExp⟩ ;
| ⟨IterationalStm⟩
| ⟨ConditionalStm⟩
| ⟨ExcpStm⟩
| ⟨LambdaStm⟩ ;
| ⟨RoutCall⟩ ;

⟨StatementList⟩ ::= ⟨Statement⟩
| ⟨Statement⟩ ⟨StatementList⟩

⟨Assignement⟩ ::= ⟨LExp⟩ = ⟨RoutCall⟩
| ⟨LExp⟩ = ⟨RExp4⟩
| ⟨LExp⟩ = ⟨Assignement⟩

⟨IterationalStm⟩ ::= for ⟨Ident⟩ in ( ⟨RExp⟩ ) : { ⟨ListBlock⟩ }
| while ( ⟨RExp⟩ ) : { ⟨ListBlock⟩ }

⟨ConditionalStm⟩ ::= if ( ⟨RExp⟩ ) : { ⟨ListBlock⟩ }
| if ( ⟨RExp⟩ ) : { ⟨ListBlock⟩ } else : { ⟨ListBlock⟩ }
| if ( ⟨RExp⟩ ) : { ⟨ListBlock⟩ } elif ( ⟨RExp⟩ ) :
{ ⟨ListBlock⟩ } else : { ⟨ListBlock⟩ }

⟨ExcpStm⟩ ::= try ⟨Block⟩ catch ⟨ListBlock⟩

```

```

⟨LambdaStm⟩ ::= lambda ⟨ListIdent⟩ : ⟨RExp⟩

⟨ListIdent⟩ ::= ε
              | ⟨Ident⟩ ⟨ListIdent⟩

⟨DefFuncStm⟩ ::= def ⟨Ident⟩ ( ⟨ListParams⟩ ) : { ⟨ListBlock⟩ return ⟨RExp⟩ }

⟨DefProcStm⟩ ::= def ⟨Ident⟩ ( ⟨ListParams⟩ ) : { ⟨ListBlock⟩ }

⟨ListParams⟩ ::= ε
                | ⟨Params⟩
                | ⟨Params⟩ , ⟨ListParams⟩

⟨Params⟩ ::= ⟨Modality⟩ ⟨Types⟩ ⟨Ident⟩

⟨Modality⟩ ::= val
               | valres
               | ε

⟨Types⟩ ::= ⟨BasicType⟩
            | ⟨PointerType⟩
            | ⟨ArrayType⟩
            | Multiple
            | ⟨Unit⟩

⟨BasicType⟩ ::= Integer
               | Double
               | String
               | Char
               | Boolean

⟨PointerType⟩ ::= * ⟨BasicType⟩

⟨ArrayType⟩ ::= Array

⟨Unit⟩ ::= Unit

⟨Block⟩ ::= ⟨StatementList⟩
            | ⟨DefFuncStm⟩
            | ⟨DefProcStm⟩

⟨ListBlock⟩ ::= ε
              | ⟨Block⟩ ⟨ListBlock⟩

```

9.2.2 Type System

Presentiamo ora il sistema di tipi per il linguaggio in analisi. Per prima cosa mostriamo alcuni giudizi per il linguaggio:

| | |
|-----------------------------------|--|
| $\Gamma \vdash \diamond$ | Γ è un ambiente ben formato |
| $\Gamma \vdash Unit$ | Unit è un tipo ben formato in Γ |
| $\Gamma \vdash BasicType$ | BasicType è un tipo ben formato in Γ |
| $\Gamma \vdash Stmt$ | Stmt è uno statement ben formato in Γ |
| $\Gamma \vdash RExpr : BasicType$ | RExpr è un'espressione ben formata di tipo BasicType in Γ |

Di seguito sono invece presentate le regole del nostro sistema di tipi.

(Ambiente vuoto)

$$\overline{\phi \vdash \diamond}$$

(Ambiente Identificatore)

$$\frac{\Gamma \vdash \diamond \quad Ident \notin dom(\Gamma)}{\Gamma, Ident \vdash \diamond}$$

(Tipo Unit)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash unit : Unit}$$

(Tipo Integer)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash n : Integer}$$

(Tipo Double)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash f : Double}$$

(Tipo Char)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash c : Char}$$

(Tipo String)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash s : String}$$

(True Boolean)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash true : Boolean}$$

(False Boolean)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash false : Boolean}$$

(Bool Or)

$$\frac{\Gamma \vdash RExp : Boolean \quad \Gamma \vdash RExp1 : Boolean}{\Gamma \vdash RExp \text{ or } RExp1 : Boolean}$$

$$(Bool\ And) \quad \frac{\Gamma \vdash RExp1 : Boolean \quad \Gamma \vdash RExp2 : Boolean}{\Gamma \vdash RExp1 \text{ and } RExp2 : Boolean}$$

$$(Bool\ Not) \quad \frac{\Gamma \vdash RExp3 : Boolean}{\Gamma \vdash \text{not } RExp3 : Boolean}$$

$$(Bool\ Eq) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 == RExp4 : Boolean}$$

$$(Bool\ NEq) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 != RExp4 : Boolean}$$

$$(Bool\ LessEq) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 <= RExp4 : Boolean}$$

$$(Bool\ GreaterEq) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 >= RExp4 : Boolean}$$

$$(Bool\ Less) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 < RExp4 : Boolean}$$

$$(Bool\ Greater) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 > RExp4 : Boolean}$$

$$(Bool\ IsNot) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 \text{ is not } RExp4 : Boolean}$$

$$(Bool\ Is) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : BasicType}{\Gamma \vdash RExp4 \text{ is } RExp4 : Boolean}$$

$$(Bool\ NotIn) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : ArrayType}{\Gamma \vdash RExp4 \text{ not in } RExp4 : Boolean}$$

$$(Bool\ In) \quad \frac{\Gamma \vdash RExp4 : BasicType \quad \Gamma \vdash RExp4 : ArrayType}{\Gamma \vdash RExp4 \text{ in } RExp4 : Boolean}$$

$$(Somma\ espressioni\ 1) \quad \frac{\Gamma \vdash RExp4 : Integer \quad \Gamma \vdash RExp5 : Integer}{\Gamma \vdash RExp4 + RExp5 : Integer}$$

$$(Somma\ espressioni\ 2) \quad \frac{\Gamma \vdash RExp4 : Integer \quad \Gamma \vdash RExp5 : Double}{\Gamma \vdash RExp4 + RExp5 : Double}$$

$$(Somma\ espressioni\ 3) \quad \frac{\Gamma \vdash RExp4 : Double \quad \Gamma \vdash RExp5 : Integer}{\Gamma \vdash RExp4 + RExp5 : Double}$$

$$(Somma\ espressioni\ 4) \quad \frac{\Gamma \vdash RExp4 : Double \quad \Gamma \vdash RExp5 : Double}{\Gamma \vdash RExp4 + RExp5 : Double}$$

$$(Sottrazione\ espressioni\ 1) \quad \frac{\Gamma \vdash RExp4 : Integer \quad \Gamma \vdash RExp5 : Integer}{\Gamma \vdash RExp4 - RExp5 : Integer}$$

$$(Sottrazione\ espressioni\ 2) \quad \frac{\Gamma \vdash RExp4 : Integer \quad \Gamma \vdash RExp5 : Double}{\Gamma \vdash RExp4 - RExp5 : Double}$$

$$(Sottrazione\ espressioni\ 3) \quad \frac{\Gamma \vdash RExp4 : Double \quad \Gamma \vdash RExp5 : Integer}{\Gamma \vdash RExp4 - RExp5 : Double}$$

$$(Sottrazione\ espressioni\ 4) \quad \frac{\Gamma \vdash RExp4 : Double \quad \Gamma \vdash RExp5 : Double}{\Gamma \vdash RExp4 - RExp5 : Double}$$

$$(Moltiplicazione\ espressioni\ 1) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 * RExp6 : Integer}$$

$$(Moltiplicazione\ espressioni\ 2) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Double}{\Gamma \vdash RExp5 * RExp6 : Double}$$

$$(Moltiplicazione\ espressioni\ 3) \quad \frac{\Gamma \vdash RExp5 : Double \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 * RExp6 : Double}$$

$$(Moltiplicazione\ espressioni\ 4) \quad \frac{\Gamma \vdash RExp5 : Double \quad \Gamma \vdash RExp6 : Double}{\Gamma \vdash RExp5 * RExp6 : Double}$$

$$(Divisione\ espressioni\ 1) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 / RExp6 : Double}$$

$$(Divisione\ espressioni\ 2) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Double}{\Gamma \vdash RExp5 / RExp6 : Double}$$

$$(Divisione espressioni 3) \quad \frac{\Gamma \vdash RExp5 : Double \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 / RExp6 : Double}$$

$$(Divisione espressioni 4) \quad \frac{\Gamma \vdash RExp5 : Double \quad \Gamma \vdash RExp6 : Double}{\Gamma \vdash RExp5 / RExp6 : Double}$$

$$(Divisione espressioni 4) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 \% RExp6 : Integer}$$

$$(DivisioneIntera espressioni 1) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 // RExp6 : Integer}$$

$$(DivisioneIntera espressioni 2) \quad \frac{\Gamma \vdash RExp5 : Integer \quad \Gamma \vdash RExp6 : Double}{\Gamma \vdash RExp5 // RExp6 : Integer}$$

$$(DivisioneIntera espressioni 3) \quad \frac{\Gamma \vdash RExp5 : Double \quad \Gamma \vdash RExp6 : Integer}{\Gamma \vdash RExp5 // RExp6 : Integer}$$

$$(DivisioneIntera espressioni 4) \quad \frac{\Gamma \vdash RExp5 : Double \quad \Gamma \vdash RExp6 : Double}{\Gamma \vdash RExp5 // RExp6 : Integer}$$

$$(Espressioni positive 1) \quad \frac{\Gamma \vdash RExp7 : Int}{\Gamma \vdash + RExp7 : Int}$$

$$(Espressioni positive 2) \quad \frac{\Gamma \vdash RExp7 : Double}{\Gamma \vdash + RExp7 : Double}$$

$$(Espressioni negative 1) \quad \frac{\Gamma \vdash RExp7 : Int}{\Gamma \vdash - RExp7 : Int}$$

$$(Espressioni negative 2) \quad \frac{\Gamma \vdash RExp7 : Double}{\Gamma \vdash - RExp7 : Double}$$

$$(Espressioni Potenza) \quad \frac{\Gamma \vdash RExp7 : Integer \quad \Gamma \vdash RExp8 : Integer}{\Gamma \vdash RExp7 ** RExp8 : Integer}$$

$$(Espressioni Range) \quad \frac{\Gamma \vdash ArrayType \quad \Gamma \vdash n_1 : Integer \quad \Gamma \vdash n_2 : Integer \quad n_1.val, n_2.val > 0}{\Gamma \vdash \text{range} (n_1, n_2) : ArrayType}$$

$$(Deref\ Indirizzo) \quad \frac{\Gamma \vdash \text{BasicType}}{\Gamma \vdash * \text{ BasicType}}$$

$$(Ref\ Indirizzo) \quad \frac{\Gamma \vdash \text{Ident}}{\Gamma \vdash & \text{ Ident}}$$

$$(Dereferenziazione) \quad \frac{\Gamma \vdash \text{Ident} : \text{BasicType}}{\Gamma \vdash \text{Ident} *}$$

$$(Assegnamento\ Basic\ 1) \quad \frac{\Gamma \vdash LExp : \text{Unit} \quad \Gamma \vdash RExp4 : \text{BasicType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Basic\ 2) \quad \frac{\Gamma \vdash LExp : \text{BasicType} \quad \Gamma \vdash RExp4 : \text{BasicType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Basic\ 3) \quad \frac{\Gamma \vdash LExp : \text{ArrayType} \quad \Gamma \vdash RExp4 : \text{BasicType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Array\ 1) \quad \frac{\Gamma \vdash LExp : \text{Unit} \quad \Gamma \vdash RExp4 : \text{ArrayType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Array\ 2) \quad \frac{\Gamma \vdash LExp : \text{BasicType} \quad \Gamma \vdash RExp4 : \text{ArrayType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Array\ 3) \quad \frac{\Gamma \vdash LExp : \text{ArrayType} \quad \Gamma \vdash RExp4 : \text{ArrayType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 1) \quad \frac{\Gamma \vdash LExp : \text{Unit} \quad \Gamma \vdash RExp4 : \text{PointerType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 2) \quad \frac{\Gamma \vdash LExp : \text{BasicType} \quad \Gamma \vdash RExp4 : \text{PointerType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 3) \quad \frac{\Gamma \vdash LExp : \text{ArrayType} \quad \Gamma \vdash RExp4 : \text{PointerType}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 3 - 1) \quad \frac{\Gamma \vdash \text{Ident} : \text{Integer} \quad \Gamma, \text{Ident} \vdash LExp : \text{PointerType} \quad \Gamma \vdash RExp4 : \text{Integer}}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 3-2) \frac{\Gamma \vdash Ident : Double \quad \Gamma, Ident \vdash LExp : PointerType \quad \Gamma \vdash RExp4 : Double}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 3-3) \frac{\Gamma \vdash Ident : Char \quad \Gamma, Ident \vdash LExp : PointerType \quad \Gamma \vdash RExp4 : Char}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 3-4) \frac{\Gamma \vdash Ident : String \quad \Gamma, Ident \vdash LExp : PointerType \quad \Gamma \vdash RExp4 : String}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ Pointer\ 3-5) \frac{\Gamma \vdash Ident : Boolean \quad \Gamma, Ident \vdash LExp : PointerType \quad \Gamma \vdash RExp4 : Boolean}{\Gamma \vdash LExp = RExp4}$$

$$(Assegnamento\ RoutCall\ 1) \frac{\Gamma \vdash LExp \quad \Gamma \vdash RoutCall : Unit}{\Gamma \vdash LExp = RoutCall}$$

$$(Assegnamento\ RoutCall\ 2) \frac{\Gamma \vdash LExp \quad \Gamma \vdash RoutCall : BasicType}{\Gamma \vdash LExp = RoutCall}$$

$$(Assegnamento\ RoutCall\ 3) \frac{\Gamma \vdash LExp \quad \Gamma \vdash RoutCall : ArrayType}{\Gamma \vdash LExp = RoutCall}$$

$$(Assegnamento\ RoutCall\ 4) \frac{\Gamma \vdash LExp \quad \Gamma \vdash RoutCall : PointerType}{\Gamma \vdash LExp = RoutCall}$$

$$(If\ then) \frac{\Gamma \vdash RExp : Boolean \quad \Gamma \vdash Block}{\Gamma \vdash \text{if} (RExp) : \{ [Block] \}}$$

$$(If\ else) \frac{\Gamma \vdash RExp : Boolean \quad \Gamma \vdash Block_1 \quad \Gamma \vdash Block_2}{\Gamma \vdash \text{if} (RExp) : \{ [Block] \} \text{ else} : \{ [Block_2] \}}$$

$$(If\ elif\ else) \frac{\Gamma \vdash RExp_1 : Boolean \quad \Gamma \vdash RExp_2 : Boolean \quad \Gamma \vdash Block_1 \quad \Gamma \vdash Block_2 \quad \Gamma \vdash Block_3}{\Gamma \vdash \text{if} (RExp) : \{ [Block] \} \text{ elif} (RExp_2) : \{ [Block_2] \} \text{ else} : \{ [Block_3] \}}$$

$$(For) \frac{\Gamma \vdash Ident \quad \Gamma \vdash RExp : ArrayType \quad \Gamma \vdash Block}{\Gamma \vdash \text{for } Ident \text{ in } texttt(RExp) \text{ texttt: } texttt\{ texttt[Block] texttt\}}$$

$$(While) \frac{\Gamma \vdash RExpr : Boolean \quad \Gamma \vdash Body}{\Gamma \vdash \text{while} (RExpr) : \{ [Block] \}}$$

$$(Dichiarazione\ Array) \frac{\Gamma \vdash RExpList : MultipleType}{\Gamma \vdash [RExpList]}$$

$$(Array) \frac{\Gamma \vdash Ident \quad \Gamma \vdash RExp : Integer}{\Gamma \vdash Ident [RExp]}$$

$$(Array\ Sequence\ 1) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Integer \quad \Gamma \vdash ArValue_2 : Integer}{\Gamma \vdash Ident [ArValue_1 : ArValue_2]}$$

$$(Array\ Sequence\ 2) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Unit \quad \Gamma \vdash ArValue_2 : Integer}{\Gamma \vdash Ident [ArValue_1 : ArValue_2]}$$

$$(Array\ Sequence\ 3) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Integer \quad \Gamma \vdash ArValue_2 : Unit}{\Gamma \vdash Ident [ArValue_1 : ArValue_2]}$$

$$(Array\ Sequence\ 4) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Unit \quad \Gamma \vdash ArValue_2 : Unit}{\Gamma \vdash Ident [ArValue_1 : ArValue_2]}$$

$$(Array\ Slicing\ 1) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Integer \quad \Gamma \vdash ArValue_2 : Integer \quad \Gamma \vdash ArValue_3 : Integer}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

$$(Array\ Slicing\ 2) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Unit \quad \Gamma \vdash ArValue_2 : Integer \quad \Gamma \vdash ArValue_3 : Integer}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

$$(Array\ Slicing\ 3) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Integer \quad \Gamma \vdash ArValue_2 : Unit \quad \Gamma \vdash ArValue_3 : Integer}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

$$(Array\ Slicing\ 4) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Integer \quad \Gamma \vdash ArValue_2 : Integer \quad \Gamma \vdash ArValue_3 : Unit}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

$$(Array\ Slicing\ 5) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Unit \quad \Gamma \vdash ArValue_2 : Unit \quad \Gamma \vdash ArValue_3 : Integer}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

$$(Array\ Slicing\ 6) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Unit \quad \Gamma \vdash ArValue_2 : Integer \quad \Gamma \vdash ArValue_3 : Unit}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

$$(Array\ Slicing\ 7) \frac{\Gamma \vdash Ident : ArrayType \quad \Gamma \vdash ArValue_1 : Integer \quad \Gamma \vdash ArValue_2 : Unit \quad \Gamma \vdash ArValue_3 : Unit}{\Gamma \vdash Ident [ArValue_1 : ArValue_2 : ArValue_3]}$$

- (*Array Slicing 8*)

$$\frac{\Gamma \vdash \text{Ident} : \text{ArrayType} \quad \Gamma \vdash \text{ArValue}_1 : \text{Unit} \quad \Gamma \vdash \text{ArValue}_2 : \text{Unit} \quad \Gamma \vdash \text{ArValue}_3 : \text{Unit}}{\Gamma \vdash \text{Ident} [\text{ArValue}_1 : \text{ArValue}_2 : \text{ArValue}_3]}$$
- (*Try Catch*)

$$\frac{\Gamma \vdash \text{Block}_1 \quad \Gamma \vdash \text{Block}_2}{\Gamma \vdash \text{try } \text{Block}_1 ; \text{ catch } [\text{Block}_2]}$$
- (*Lambda*)

$$\frac{\Gamma \vdash \text{Ident} \quad \Gamma \vdash \text{RExp}}{\Gamma \vdash \text{lambda } [\text{Ident}] : \text{RExp}}$$
- (*Dichiarazione Funzione*)

$$\frac{\Gamma \vdash \text{Ident} \quad \Gamma \vdash \text{Params} : \text{MultipleType} \quad \Gamma \vdash \text{Block} \quad \Gamma \vdash \text{RExp}}{\Gamma \vdash \text{def } \text{Ident} ([\text{Params}]) : \{ [\text{Block}] \text{ return } \text{RExp} \}}$$
- (*Dichiarazione Procedura*)

$$\frac{\Gamma \vdash \text{Ident} \quad \Gamma \vdash \text{Params} : \text{MultipleType} \quad \Gamma \vdash \text{Block}}{\Gamma \vdash \text{def } \text{Ident} ([\text{Params}]) : \{ [\text{Block}] \}}$$
- (*Delete 1*)

$$\frac{\Gamma \vdash \text{RExp10} : \text{BasicType}}{\Gamma \vdash \text{del } \text{RExp10} ;}$$
- (*Delete 2*)

$$\frac{\Gamma \vdash \text{RExp10} : \text{PointerType}}{\Gamma \vdash \text{del } \text{RExp10} ;}$$
- (*Delete 3*)

$$\frac{\Gamma \vdash \text{RExp10} : \text{ArrayType}}{\Gamma \vdash \text{del } \text{RExp10} ;}$$
- (*Val Print 1*)

$$\frac{\Gamma \vdash \text{RExp} : \text{Integer}}{\Gamma \vdash \text{print(RExp)}}$$
- (*Val Print 2*)

$$\frac{\Gamma \vdash \text{RExp} : \text{Double}}{\Gamma \vdash \text{print(RExp)}}$$
- (*Val Print 3*)

$$\frac{\Gamma \vdash \text{RExp} : \text{String}}{\Gamma \vdash \text{print(RExp)}}$$
- (*Val Print 4*)

$$\frac{\Gamma \vdash \text{RExp} : \text{Char}}{\Gamma \vdash \text{print(RExp)}}$$
- (*Val Print 5*)

$$\frac{\Gamma \vdash \text{RExp} : \text{Boolean}}{\Gamma \vdash \text{print(RExp)}}$$

$$\frac{(Val\ Read\ 1) \\ \Gamma \vdash RExp : Integer}{\Gamma \vdash \text{read}(RExp)}$$

$$\frac{(Val\ Read\ 2) \\ \Gamma \vdash RExp : Double}{\Gamma \vdash \text{read}(RExp)}$$

$$\frac{(Val\ Read\ 3) \\ \Gamma \vdash RExp : String}{\Gamma \vdash \text{read}(RExp)}$$

$$\frac{(Val\ Read\ 4) \\ \Gamma \vdash RExp : Char}{\Gamma \vdash \text{read}(RExp)}$$

$$\frac{(Val\ Read\ 5) \\ \Gamma \vdash RExp : Boolean}{\Gamma \vdash \text{read}(RExp)}$$

9.2.3 Descrizione della soluzione

Per svolgere questo esercizio si è scelto di utilizzare “BNF Converter” e di modificare il successivamente i file prodotti per il parser.

I sorgenti sono presenti nella cartella `Esercizio9`.

9.2.4 Grammatica

E' stato scelto in fase di definizione della grammatica di non gestire le indentazioni, ma di sostituirle con l'apertura e la chiusura di parentesi graffe (come avviene in Haskell). E' stato inoltre vincolato l'uso del simbolo ; al termine di ogni statement ad eccezione degli Statement Condizionali ed Iterativi. E' stato altresì deciso di vietare alcune situazioni illegali nella scrittura del codice Python direttamente a livello sintattico, sfruttando i livelli di precedenza definibili nel file `python.cf`.

9.2.5 Type Checker

Il Type Checking è stato implementato come specificato precedentemente. Ad ogni produzione sono stati assegnati i relativi tipi. Per ogni operazione è stato effettuato un controllo sugli operandi ritenuti legali, gestendo la creazione di un messaggio di errore qualora i tipi degli operandi in analisi non siano ammessi. Durante l'attività implementativa quest'ultima parte ha fatto emergere delle problematiche che non hanno consentito di completare lo sviluppo del type-system sul front-end del compilatore sviluppato.