Corso di linguaggi e compilatori

Relazione – Parte 3 – Gruppo 2

Giovanni D’Agostino, Beatrice Portelli

17 Aprile 2019

**Esercizio 1**

**1) Grammatica LL(1)**

La grammatica LL(1) da noi definita equivalente alla grammatica L data è la seguente:

L → NM

M → ++L | ε

N → [O | E:N

O → LL | E..E]

LL → ] | E,LL

E → id | num

FIRST(L) = FIRST(N) = { [, id, num }  
FIRST(M) = { ++, ε}  
FIRST(O) = FIRST(LL) = { ], id , num }  
FIRST(E) = { id, num }

FOLLOW(L) = FOLLOW(M) = {$}  
FOLLOW(N) = FOLLOW(O) = FOLLOW(LL) = {++, $}  
FOLLOW(E) = { :, .., , , ]} (la virgola in rosso è un token della grammatica)

**2) Tabella di parsing ed error recovery**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | id | num | : | ++ | .. | , | [ | ] | $ |
| L | L→NM | L→NM | e1 | e2 | e2 | e2 | L→NM | e3 | s1 |
| M | e4 | e4 | e5 | M→++L | e6 | e7 | e8 | e9 | M→ε |
| N | N→E:N | N→E:N | e10 | s2 | e6 | e7 | N→[O | e3 | s3 |
| O | O→EP | O→EP | e5 | s2 | e1 | e1 | e11 | O→] | s3 |
| P | e4 | e4 | e5 | s2 | P→..E] | P→,LL | e12 | e13 | s3 |
| LL | LL→E,LL | LL→E,LL | e5 | s2 | e6 | e14 | e12 | LL→] | s3 |
| E | E→id | E→num | e5 | e15 | s4 | s5 | e12 | e16 | s3 |
|  |  |  |  |  |  |  |  |  |  |

Errori:

* per gli errori e1 ed e16 abbiamo scelto arbitrariamente di inserire un id, sarebbe stato possibile inserire un num
* nell’esecuzione del parsing sull’input dato abbiamo usato la convenzione (spiegata nel libro di testo, capitolo 4.5.5) di effettuare un pop sullo stack nel caso che il token al top of stack non corrisponda al simbolo in input, per poi riprendere il parsing

e1: echo "missing id or num before operator" insert id

e2: echo "input starting with unexpected token" skip

e3: echo "missing [ here" insert [

e4: echo "unexpected id or num found" skip

e5: echo "misuse of : here" skip

e6: echo "misuse of .. here" skip

e7: echo "unexpected , found" skip

e8: echo "missing ++ operator before [" insert ++

e9: echo "duplicated ] found" skip

e10: echo "duplicated : found" skip

e11: echo "duplicated [ found" skip

e12: echo "unexpected [ found" skip

e13: echo "missing , at the end of a list" insert ,

e14: echo "duplicated , found" skip

e15: echo "unexpected ++ found" skip

e16: echo "missing id or num before ]" insert id

s1: echo "unexpected eof" pop push M

s2: echo "unexpected ++ found" pop

s3: echo "unexpected eof" pop until M

s4: echo "misuse of .. inside a list" pop

s5: echo "misuse of , inside a list" pop

**3) Esecuzione dell’input**

|  |  |  |  |
| --- | --- | --- | --- |
| STACK (ToS a sx) | INPUT | ACTION | MATCHED |
| L$ | [[id..[num]++id,num]:[id id]$ | L → NM |  |
| NM$ | [[id..[num]++id,num]:[id id]$ | N → [O |  |
| [OM$ | [[id..[num]++id,num]:[id id]$ | match [ | [ |
| OM$ | [id..[num]++id,num]:[id id]$ | e11 (skip) | [ |
| OM$ | id..[num]++id,num]:[id id]$ | O → EP | [ |
| EPM$ | id..[num]++id,num]:[id id]$ | E → id | [ |
| idPM$ | id..[num]++id,num]:[id id]$ | match id | [id |
| PM$ | ..[num]++id,num]:[id id]$ | P → ..E] | [id |
| ..E]M$ | ..[num]++id,num]:[id id]$ | match .. | [id.. |
| E]M$ | [num]++id,num]:[id id]$ | e12 (skip) | [id.. |
| E]M$ | num]++id,num]:[id id]$ | E → num | [id.. |
| num]M$ | num]++id,num]:[id id]$ | match num | [id..num |
| ]M$ | ]++id,num]:[id id]$ | match ] | [id..num] |
| M$ | ++id,num]:[id id]$ | M → ++L | [id..num] |
| ++L$ | ++id,num]:[id id]$ | match ++ | [id..num]++ |
| L$ | id,num]:[id id]$ | L → NM | [id..num]++ |
| NM$ | id,num]:[id id]$ | N → E:N | [id..num]++ |
| E:NM$ | id,num]:[id id]$ | E → id | [id..num]++ |
| id:NM$ | id,num]:[id id]$ | match id | [id..num]++id |
| :NM$ | ,num]:[id id]$ | pop | [id..num]++id |
| NM$ | ,num]:[id id]$ | e7 (skip) | [id..num]++id |
| NM$ | num]:[id id]$ | N → E:N | [id..num]++id |
| E:NM$ | num]:[id id]$ | E → num | [id..num]++id |
| num:NM$ | num]:[id id]$ | match num | [id..num]++id num |
| :NM$ | ]:[id id]$ | pop | [id..num]++id num |
| NM$ | ]:[id id]$ | e3 (insert [) | [id..num]++id num |
| NM$ | []:[id id]$ | N → [O | [id..num]++id num |
| [OM$ | []:[id id]$ | match [ | [id..num]++id num[ |
| OM$ | ]:[id id]$ | O → ] | [id..num]++id num[ |
| ]M$ | ]:[id id]$ | match ] | [id..num]++id num[] |
| M$ | :[id id]$ | e5 (skip) | [id..num]++id num[] |
| M$ | [id id]$ | e8 (insert ++) | [id..num]++id num[] |
| M$ | ++[id id]$ | M → ++L | [id..num]++id num[] |
| ++L$ | ++[id id]$ | match ++ | [id..num]++id num[]++ |
| L$ | [id id]$ | L → NM | [id..num]++id num[]++ |
| NM$ | [id id]$ | N → [O | [id..num]++id num[]++ |
| [OM$ | [id id]$ | match [ | [id..num]++id num[]++[ |
| OM$ | id id]$ | O → EP | [id..num]++id num[]++[ |
| EPM$ | id id]$ | E → id | [id..num]++id num[]++[ |
| idPM$ | id id]$ | match id | [id..num]++id num[]++[id |
| PM$ | id]$ | e4 (skip) | [id..num]++id num[]++[id |
| PM$ | ]$ | e13 (insert ,) | [id..num]++id num[]++[id |
| PM$ | ,]$ | P→,LL | [id..num]++id num[]++[id |
| ,LLM$ | ,]$ | match , | [id..num]++id num[]++[id, |
| LLM$ | ]$ | LL→] | [id..num]++id num[]++[id, |
| ]M$ | ]$ | match ] | [id..num]++id num[]++[id,] |
| M$ | $ | M→ε | [id..num]++id num[]++[id,] |
| $ | $ | OK | [id..num]++id num[]++[id,] |

**Esercizio 2**

**1) Parser SLR e LALR**

**N.B.** Nella grammatica abbiamo introdotto il nuovo simbolo iniziale C’ e la produzione C’ → C

FIRST(C’) = FIRST(C) = {pred, num, id, ( }  
FIRST(E) = FIRST(Es) = {num, id, ( }  
  
FOLLOW(C’) = {$}  
FOLLOW(C) = {$, &}  
FOLLOW(E) = {rel, +, \*, ), , , $, & }  
FOLLOW(Es) = {)}

**2) Item LR(0)**

I 0

C' → • C

C → • E rel E

C → • C & C

C → • pred ( Es )

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 1

C' → C •

C → C • & C

I 2

C → E • rel E

E → E • + E

E → E • \* E

I 3

C → pred • ( Es )

I 4

E → num •

I 5

E → id •

I 6

E → ( • E )

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 7

C → C & • C

C → • E rel E

C → • C & C

C → • pred ( Es )

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 8

C → E rel • E

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 9

C → E + • E

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 10

C → E \* • E

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 11

C → pred ( • Es )

Es → • E

Es → • E , Es

E → • num

E → • id

E → • E + E

E → • E \* E

E → • ( E )

I 12

E → ( E • )

E → E • + E

E → E • \* E

I 13

C → C & C•

C → C • & C

I 14

C → E rel E •

E → E • + E

E → E • \* E

I 15

E → E + E •

E → E • + E

E → E • \* E

I 16

E → E \* E •

E → E • + E

E → E • \* E

I 17

C → pred ( Es • )

I 18

Es → E •

Es → E • , Es

E → E • + E

E → E • \* E

I 19

E → ( E ) •

I 20

C → pred ( Es ) •

I 21

Es → E , • Es

Es → • E

Es → • E , Es

E → • num

E → • id

E → • E + E

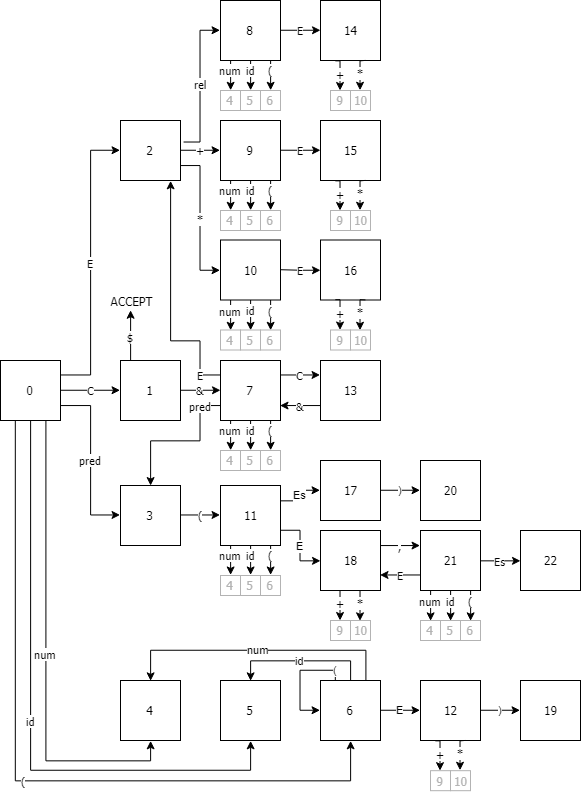
E → • E \* E

E → • ( E )

I 22

Es → E , Es •

**3) Automa SLR**



**4) Conflitti shift/reduce**

**N.B.** Nei conflitti shift/reduce della matrice di parsing sono state riportate in grassetto le action scelte, mentre in caratteri più piccoli le action scartate.

* ACTION[13, &]: abbiamo scelto di effettuare un reduce per implementare l’associatività a sinistra dell’operatore & nella regola C 🡪 C & C
* ACTION[15, \*]: abbiamo scelto di effettuare uno shift per implementare la precedenza dell’operatore \* sull’operatore +
* ACTION[15, +]: abbiamo scelto di effettuare un reduce per implementare l’associatività a sinistra dell’operatore + nella regola E 🡪 E + E
* ACTION[16, \*]: abbiamo scelto di effettuare un reduce per implementare l’associatività a sinistra dell’operatore \* nella regola E 🡪 E \* E
* ACTION[16, +]: abbiamo scelto di effettuare un reduce per implementare la precedenza dell’operatore \* sull’operatore +