

Analisi sintattico top-down o discendente

Parte seconda

Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica LL(1), cioè tale che per ogni coppia di produzioni a partire da uno stesso simbolo non terminale A :

$A \rightarrow \alpha$ e

$A \rightarrow \beta$, si ha

$$\text{Gui}(A \rightarrow \alpha) \cap \text{Gui}(A \rightarrow \beta) = \Phi.$$

Costruiamo a partire da G un riconoscitore che si *ispira* direttamente ad un automa a pila che accetta per stack vuoto.

Il riconoscitore scorre l'input (che sarà terminato da un simbolo di fine stringa (\$)) con una testina di lettura e usa una pila, che supporremo svilupparsi da destra a sinistra per rendere più evidente il significato delle sue mosse.

L'informazione relativa alla scelta della produzione si può condensare in una *tabella* che per ogni variabile sulla pila e ogni simbolo terminale indica la produzione (se esiste) da applicare.

Un parser LL(1) (versione iterativa) analizza una stringa leggendola una sola volta da sinistra a destra. Utilizza una tabella derivata dalla grammatica (tabella LL(1)) e una Pila.

Input. 1) Stringa da parsificare

2) Tabella **M** che memorizza, per ogni coppia
< variabile A, simbolo in input a >
la produzione $A \rightarrow \alpha$ se $a \in \text{Gui}(A \rightarrow \alpha)$,
' ' se $a \notin \text{Gui}(A \rightarrow \alpha)$.

Output. Stringa accettata o segnalazione di errore

Grammatiche LL(1): esempio 2 (più facile)

Una grammatica per $\{0^n 1^n \mid n > 0\}$

Si $\text{First}(S) = \{0\}$, $\text{First}(A) = \{0,1\}$, $\text{First}(S1) = \text{First}(S) = \{0\}$

| Produzione | Insieme guida |
|--------------------|---------------|
| $S \rightarrow 0A$ | $\{0\}$ |
| $A \rightarrow S1$ | $\{0\}$ |
| $A \rightarrow 1$ | $\{1\}$ |

| | 0 | 1 | \$ |
|---|--------------------|-------------------|----|
| S | $S \rightarrow 0A$ | | |
| A | $A \rightarrow S1$ | $A \rightarrow 1$ | |

La grammatica è LL(1)

Grammatiche LL(1): esempio 2

| Input | stack | derivazione |
|-------------|-------|-------------|
| 0011\$ ↑ | S | S |
| 0011\$ ↑ | 0A | 0A |
| 0011\$ ↑ | A | |
| 0011\$ ↑ | S1 | 0S1 |
| 0011\$ ↑ | 0A1 | 00A1 |
| 0011\$ ↑ | A1 | |
| 0011\$ ↑ | 11 | 0011 |
| 0011\$ ↑ | 1 | |
| 0011\$ ↑ | | |
| | | |

Derivazione della stringa: $S \rightarrow 0A \rightarrow 0S1 \rightarrow 00A1 \rightarrow 0011$

Espressioni aritmetiche

Produzione

Insieme guida

$$1. E \rightarrow T E'$$

$$F(TE') = F(T) = F(F) = \{ (, id \}$$

$$2. E' \rightarrow + T E'$$

$$\{ + \}$$

$$3. E' \rightarrow \varepsilon$$

$$\Phi \cup FW(E') = \{), \$ \}$$

$$\left. \begin{array}{l} \{ + \} \\ \{), \$ \} \end{array} \right\} \{ + \} \cap \{), \$ \} = \Phi$$

$$4. T \rightarrow F T'$$

$$F(F) = \{ (, id \}$$

$$5. T' \rightarrow * F T'$$

$$\{ * \}$$

$$6. T' \rightarrow \varepsilon$$

$$\Phi \cup FW(T') = \{ +,), \$ \}$$

$$\left. \begin{array}{l} \{ * \} \\ \{ +,), \$ \} \end{array} \right\} \{ * \} \cap \{ +,), \$ \} = \Phi$$

$$7. F \rightarrow (E)$$

$$\{ (\}$$

$$8. F \rightarrow id$$

$$\{ id \}$$

$$\left. \begin{array}{l} \{ (\} \\ \{ id \} \end{array} \right\} \{ (\} \cap \{ id \} = \Phi$$

$$FW(E') = FW(E) = \{), \$ \}$$

$$FW(T') = FW(T) = (F(E') - \{\varepsilon\}) \cup FW(E) = \{ +,), \$ \}$$

Per esempio

- $\text{FW}(E') = \{), \$\}$ perchè

$$E \rightarrow TE' \quad e$$

$$E \rightarrow^* (E) \rightarrow (TE')$$

- $\text{FW}(T') = \{+,), \$\}$ perché:

$$E \rightarrow TE' \rightarrow T$$

$$E \rightarrow^* (E) \rightarrow (TE') \rightarrow (T)$$

$$E \rightarrow^* TE' \rightarrow T + TE'$$

Parsificazione top-down: la tabella del parser LL(1)


Si possono visualizzare le produzioni da usare in funzione delle variabili e dei simboli in input con una **tabella** che contiene, per ogni coppia $\langle A, a \rangle$ la produzione $A \rightarrow \alpha$ se $a \in \text{Gui}(A \rightarrow \alpha)$, ' ' se $a \notin \text{Gui}(A \rightarrow \alpha)$.

Nel caso delle espressioni aritmetiche:

| | (|) | + | * | id | \$ |
|----|-----------------------|---------------------------|---------------------------|-------------------------|---------------------------|---------------------------|
| E | $E \rightarrow T E'$ | | | | $E \rightarrow T E'$ | |
| E' | | $E' \rightarrow \epsilon$ | $E' \rightarrow + T E'$ | | | $E' \rightarrow \epsilon$ |
| T | $T \rightarrow F T'$ | | | | $T \rightarrow F T'$ | |
| T' | | $T' \rightarrow \epsilon$ | $T' \rightarrow \epsilon$ | $T' \rightarrow * F T'$ | | $T' \rightarrow \epsilon$ |
| F | $F \rightarrow (E)$ | | | | $F \rightarrow \text{id}$ | |

La grammatica è LL(1) in quanto in ogni elemento della tabella compare al massimo una produzione: gli insiemi guida delle riscritture di una stessa variabile sono disgiunti.

Parsificazione top-down: esempio di analisi per espressioni aritmetiche

| input | stack  | derivazione |
|--------------|---|----------------|
| (id+id)*id- | E | E |
| ↑(id+id)*id- | TE' | TE' |
| ↑(id+id)*id- | FT'E' | FT'E' |
| ↑(id+id)*id- | (E)T'E' | (E)T'E' |
| ↑(id+id)*id- | E)T'E' | (E)T'E' |
| ↑(id+id)*id- | TE')T'E' | (TE')T'E' |
| ↑(id+id)*id- | FT'E')T'E' | (FT'E')T'E' |
| ↑(id+id)*id- | idT'E')T'E | (id T'E')T'E' |
| ↑(id+id)*id- | T'E')T'E | (id T'E')T'E' |
| ↑(id+id)*id- | E')T'E | (id E')T'E' |
| ↑(id+id)*id- | +TE')T'E | (id +T E')T'E' |

NOTA: leggasì \$ al posto di --|

Parsificazione top-down: esempio di analisi

| input | stack | derivazione |
|-------------|----------|--------------------|
| (id+id)*id- | TE')T'E | (id +T E')T'E' |
| (id+id)*id- | FT')T'E' | TE' |
| | | |
| (id+id)*id- | T'E' | (id + id)T'E' |
| (id+id)*id- | *FT'E' | (id + id)*FT'E' |
| (id+id)*id- | FT'E' | (id + id)* FT'E' |
| (id+id)*id- | id T'E' | (id + id)* id T'E' |
| (id+id)*id- | T'E' | (id + id)* id T'E' |
| (id+id)*id- | E' | (id + id)* id E' |
| (id+id)*id- | | (id + id)* id |

NOTA: leggasi \$ al posto di --|

**Data la
seguente
Grammatica:**

Produzione

Insieme guida

- | | |
|--------------------------------|---------------|
| 1. $S \rightarrow PQ$ | $\{a, b, c\}$ |
| 2. $Q \rightarrow \&PQ$ | $\{\&\}$ |
| 3. $Q \rightarrow \varepsilon$ | $\{\$\}$ |
| 4. $P \rightarrow aPb$ | $\{a\}$ |
| 5. $P \rightarrow bPa$ | $\{b\}$ |
| 6. $P \rightarrow c$ | $\{c\}$ |

Verificare gli insiemi guida calcolando I FIRST e i FOLLOW

Parsificazione top-down: altri esercizi

1. Data la grammatica con il seguente insieme di produzioni:
 $\{S \rightarrow RA, S \rightarrow A[S], R \rightarrow E = B, B \rightarrow b, E \rightarrow bA, A \rightarrow \varepsilon\}$
 - a) Calcolare gli inizi (first) e i seguiti (follow) dei simboli non terminali;
 - b) Dire se la grammatica è LL(1), motivando la risposta.
2. Per ognuna delle seguenti grammatiche, specificate dall'insieme delle produzioni, costruire gli insiemi guida delle produzioni e, se la grammatica è LL(1), scrivere la tabella di parsificazione.

$G_1:$ $N \rightarrow D K$
 $K \rightarrow N$
 $K \rightarrow \varepsilon$
 $D \rightarrow 0$
 $D \rightarrow 1$

$G_2:$ $E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid \text{id}$

Notazioni in pseudo-codice

Nella slide successiva viene utilizzato uno pseudo-codice per descrivere l'algoritmo di parsificazione. Si assume che il programma abbia accesso alla tabella LL(1) della grammatica da parsificare.

- cc contiene il primo simbolo dell'input, quello su cui verranno prese le decisioni;
- PROSS è una funzione che restituisce simbolo sotto la testina di lettura e avanza la testina stessa;

Analizzatore a Discesa Dicorsiva

Consideriamo una grammatica LL(1)

Ad ogni variabile A con produzioni

$A \rightarrow \alpha_1$

$A \rightarrow \alpha_2$

\dots
 $A \rightarrow \alpha_k$

si associa una procedura:

function A()

begin

if (cc \in Gui ($A \rightarrow \alpha_1$)) body (α_1)

else if (cc \in Gui($A \rightarrow \alpha_2$)) body (α_2)

\dots

else if (cc \in Gui($A \rightarrow \alpha_k$)) body (α_k)

else ERRORE (...)

end

Parsificazione top-down: analizzatore a discesa ricorsiva

Se $\alpha = \varepsilon$, $\text{body}(\varepsilon) = \underline{\text{do nothing}}$

Se $\alpha = X_1, \dots, X_m$, $\text{body}(X_1, \dots, X_m)$ è così definito:

$\text{body}(X_1, \dots, X_m) = \text{act}(X_1) \text{ act}(X_2) \dots \text{act}(X_m)$

$$\text{act}(X) = \begin{cases} X() & \text{se } X \in V \\ \begin{array}{l} \underline{\text{if}} (cc = X) \text{ cc} \leftarrow \text{PROSS} \\ \underline{\text{else}} \text{ERRORE}(\dots) \end{array} & \text{se } X \in \Sigma \end{cases}$$

```
main() //Program discesa_ricorsiva
begin cc ← PROSS
      S()
      if (cc = '$') "stringa accettata"
      else ERRORE(...)
end
```

Parsificazione top-down: analizzatore a discesa ricorsiva

Grammatica

1. $S \rightarrow PQ$
2. $Q \rightarrow \&PQ$
3. $Q \rightarrow \varepsilon$
4. $P \rightarrow aPb$
5. $P \rightarrow bPa$
6. $P \rightarrow c$

Insieme guida

- {a, b, c}
- {&}
- {\\$}
- {a}
- {b}
- {c}

// Program discesa_ricorsiva

main()

begin cc := PROSS

S()

if cc = '\$' "stringa accettata"

else ERRORE (...)

end

function S()

begin if cc = 'a' or cc = 'b' or cc = 'c'

P()

Q()

else ERRORE (...)

end

Parsificazione top-down: analizzatore a discesa ricorsiva

```
function P()  
begin if cc = 'a' cc ← PROSS  
      P()  
      if cc = 'b' cc ← PROSS  
      else ERRORE(...)  
else if cc = 'b' cc ← PROSS  
      P()  
      if cc = 'a' cc ← PROSS  
      else ERRORE(...)  
else if cc = 'c' cc ← PROSS  
else ERRORE(...)  
end
```

```
function Q()  
begin if cc = '&' cc ← PROSS  
      P()  
      Q()  
      else if cc = '$' do nothing  
      else ERRORE (...)  
end
```

Parsificazione top-down: analizzatore a discesa ricorsiva

a a c b b & b c a \$

[][][][][S]

[][][P][Q]

[][P][b][Q] 1

[P][b][b][Q] 2

[][b][b][Q]

[][][b][Q]

[][][][Q] 3

[][][P][Q]

[][P][a][Q] 4

[][][a][Q]

[][][][Q]

[][][][][]

if cc = 'a' or cc = 'b' or cc = 'c'

P()

if cc = 'a' cc ← PROSS

P() ←1

if cc = 'a' cc ← PROSS

P() ←2

if cc = 'c' cc ← PROSS

if cc = 'b' cc ← PROSS
if cc = 'b' cc ← PROSS

Q() ←3

if cc = '&' cc ← PROSS

P()

if cc = 'b' cc ← PROSS

P() ←4

if cc = 'c' then cc ← PROSS

if cc = 'a' cc ← PROSS

Q()

if cc = '\$' do nothing

S → PQ

Q → &PQ


Q → ε

P → aPb

P → bPa

P → c

Parsificazione top-down: analizzatore a discesa ricorsiva, altro esempio

| Grammatica | Insieme guida | |
|-----------------------------|---------------------|---|
| $S \rightarrow (S)$ | $\{ (\}$ |  LL(1) |
| $S \rightarrow [S]$ | $\{ [\}$ | |
| $S \rightarrow < S >$ | $\{ < \}$ | |
| $S \rightarrow \varepsilon$ | $\{),], >, \$ \}$ | |

```
main()  // Program discesa_ricorsiva
  begin cc ← PROSS
        S()
        if (cc = '$') "stringa accettata"
        else ERRORE(...)
  end
```

Parsificazione top-down: analizzatore a discesa ricorsiva

```
function S
  begin if (cc = '(') cc ← PROSS
      S()
      if (cc = ')') cc ← PROSS
      else ERRORE(...)
  else if (cc = '[') cc ← PROSS
      S()
      if (cc = ']') cc ← PROSS
      else ERRORE(...)
  else if (cc = '<') cc ← PROSS
      S()
      if (cc = '>') cc ← PROSS
      else ERRORE(...)
  else if (cc = ') or cc = ']' or cc = '>' or cc = '$')
      do nothing
  else ERRORE (...)
end
```

Parsificazione top-down: analizzatore a discesa ricorsiva, altro esempio

Grammatica

1 $S \rightarrow a B b$

2 $S \rightarrow B S$

3 $B \rightarrow b$

4 $B \rightarrow c$

Insieme guida

{a}

{b, c}

{b}

{c}

 LL(1)

```
main()    // Program discesa_ricorsiva
begin cc ← PROSS
          S()
          if (cc = '$') "stringa accettata"
          else ERRORE(...)
end
```

Parsificazione top-down: analizzatore a discesa ricorsiva, altro esempio

```
function S()  
  begin  if (cc = 'a')  
          cc ← PROSS  
          B()  
          if (cc = 'b') cc ← PROSS  
          else ERRORE(...)  
  else if (cc = 'b' or cc = 'c')  
          B()  
          S()  
  else ERRORE(...)  
end  
  
  function B()  
    begin  if (cc = 'b') cc ← PROSS  
          else if (cc = 'c') cc ← PROSS  
          else ERRORE(...)  
    end
```

Ricorsioni sinistre

Una grammatica ricorsiva sinistra, cioè tale che per qualche non terminale A si ha una derivazione $A \rightarrow^+ A\alpha$, non è LL(1).

$$S \rightarrow S a \mid b$$

$$\{b\} = \text{Gui}(S \rightarrow Sa) = \text{Gui}(S \rightarrow b)$$

.....

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \text{id}$$

$$\{ (, \text{id} \} = \text{Gui}(T \rightarrow T * F) = \text{Gui}(T \rightarrow F)$$

$$\{ (, \text{id} \} = \text{Gui}(E \rightarrow E + T) = \text{Gui}(E \rightarrow T)$$

Una grammatica LL(1) è:

1. Non ambigua
2. Senza ricorsioni sinistre
3. Per ogni coppia di produzioni del tipo:
 $A \rightarrow \alpha$
 $A \rightarrow \beta$
 - da α e β non derivano stringhe che iniziano con lo stesso terminale
 - al più una tra α e β è annullabile
 - se α è annullabile ($\alpha \Rightarrow^* \varepsilon$), da β non deriva nessuna stringa che inizia con un terminale nell'insieme $\text{Seg}(A)$.

Parsificazione top-down: trasformazioni di grammatiche

Data una grammatica non LL(1), è qualche volta possibile ottenerne una equivalente LL(1).

In particolare si puo':

1. **eliminare le ricorsioni sinistre**, sia immediate sia non immediate
2. cercare di rendere la scelta della produzione da usare ad ogni passo deterministica posticipando, quando possibile, la scelta tra diverse alternative di riscrittura che hanno un prefisso comune: **fattorizzazione sinistra**

Eliminazione delle ricorsioni sinistre immediate

$$\begin{array}{ll} A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_k & k \geq 1, \quad \alpha_i \neq \varepsilon \\ A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_h & h \geq 1 \end{array}$$



$$\begin{array}{l} A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_h A' \\ A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_k A' \mid \varepsilon \end{array}$$

Eliminazione delle ricorsioni sinistre

Esempio

$$E \rightarrow E + T \mid T$$

$$\begin{aligned} E &\rightarrow T E' \\ E' &\rightarrow + T E' \mid \varepsilon \end{aligned}$$

$$T \rightarrow T * F \mid F$$

$$\begin{aligned} T &\rightarrow F T' \\ T' &\rightarrow * F T' \mid \varepsilon \end{aligned}$$

$$F \rightarrow (E) \mid \text{id}$$

$$F \rightarrow (E) \mid \text{id}$$

Fattorizzazione sinistra

Per ogni non terminale A si trova il massimo prefisso α comune a due o più alternative.

Si sostituiscono tutte le produzioni:

$$A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid \dots \mid \alpha \beta_m$$

con $A \rightarrow \alpha A'$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

lasciando le altre produzioni da A inalterate.

1. Eliminare la ricorsione sinistra dalle grammatiche:

$$S \rightarrow R \mid 0$$

$$R \rightarrow RC \mid R0 \mid C$$

$$C \rightarrow 1 \mid 2 \mid 3$$

$$S \rightarrow S s- \mid S\{S\} \mid D \mid \varepsilon$$

$$D \rightarrow TL-R$$

$$R \rightarrow \varepsilon \mid D$$

$$L \rightarrow L ; m \mid m$$

$$T \rightarrow i \mid r$$

N.B. $\{s, -, \{, \}, ,, m, i, r\}$
è l'insieme dei simboli
terminali

2. Data la seguente grammatica:

$$A \rightarrow A s B$$

$$A \rightarrow A m B$$

$$A \rightarrow B$$

$$A \rightarrow a$$

$$B \rightarrow (A)$$

$$B \rightarrow b$$

- Eliminare le ricorsioni sinistre
- Costruire sia nella grammatica data, sia nella grammatica ottenuta senza ricorsioni sinistre, l'albero di derivazione per la stringa:
 $b m ((a s b) s b)$