

# Analisi sintattica. Grammatiche $LL(k)$ e $LR(k)$

*Dispensa del corso di Linguaggi e Traduttori*

A.A. 2005-2006

Giovanni Semeraro

## **Analisi sintattica (Parsing).**

Data una grammatica libera da contesto (CFG)  $G$  ed una stringa  $w$  di caratteri terminali, analizzare sintatticamente  $w$  significa determinare se  $w \in L(G)$  e, in tal caso, costruire l'albero sintattico di  $w$  (o gli alberi sintattici, qualora  $G$  sia ambigua).

Nel caso di linguaggi regolari, non si parla di analisi sintattica, ma soltanto di *riconoscimento* poiché la struttura delle frasi del linguaggio è nota a priori, trattandosi di alberi sintattici che crescono soltanto verso destra (o sinistra) se le regole sono del tipo  $A \rightarrow aB$  ( $A \rightarrow Ba$ ).

Il riconoscimento viene effettuato da un automa a stati finiti.

Generalmente, si è interessati all'analisi sintattica di linguaggi non contestuali, o addirittura di loro sottoclassi.

L'analisi di linguaggi più potenti, di tipo 1 o 0, è meno interessante, sia per la maggiore complessità cui si andrebbe incontro sia per la minore importanza applicativa.

Le caratteristiche degli algoritmi di analisi sintattica di interesse per il progettista di traduttori sintattici o compilatori sono: la velocità, la quantità di memoria occupata, la capacità di riconoscere ed elaborare errori sintattici, la generalità della classe di linguaggi riconosciuti e la possibilità di costruzione automatica dell'analizzatore.

L'analisi sintattica può essere svolta da un automa a pila che, ogni volta che riconosce una produzione, emette il relativo numero d'ordine (trasduttore a pila).

### Esempio 1

Si consideri la seguente grammatica:

- (1)  $S \rightarrow 0SAB$
- (2)  $S \rightarrow 1$
- (3)  $A \rightarrow 1A$
- (4)  $A \rightarrow 1$
- (5)  $B \rightarrow 2B$
- (6)  $B \rightarrow 2$

Il trasduttore a pila che effettua l'analisi sintattica di  $L(G)$  si ottiene facilmente dall'automa a pila che effettua il riconoscimento di  $L(G)$ .

$$T = (Q, X, \Gamma, \Omega, \delta, q_0, Z_0, F) = \\ = (\{q_0\}, \{0,1,2\}, \{S, A, B\}, \{1,2,3,4,5,6\}, \delta, q_0, S, \emptyset)$$

ove:

$$\delta: Q \times (X \cup \{\lambda\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^* \times \Omega^*}$$

$$\delta(q_0, 0, S) = \{(q_0, SAB, 1)\}$$

$$\delta(q_0, 1, S) = \{(q_0, \lambda, 2)\}$$

$$\delta(q_0, 1, A) = \{(q_0, A, 3), (q_0, \lambda, 4)\}$$

$$\delta(q_0, 2, B) = \{(q_0, B, 5), (q_0, \lambda, 6)\}$$

Durante il riconoscimento, in condizioni di pila vuota, di una stringa  $x \in L(G)$ , il trasduttore emette una stringa  $\omega \in \Omega^*$  che rappresenta la sequenza delle regole applicate nella derivazione sinistra di  $x$ .

Per esempio, se  $x = 0111222$ , si ottiene  $\omega = 123455$ .

Si è già detto che un analizzatore sintattico di importanza pratica deve essere veloce.

Nel caso dell'automa precedente si è in presenza di un automa non deterministico e ciò ha una influenza negativa sul tempo di analisi. Infatti, poiché non è possibile scegliere univocamente una transizione in ogni configurazione e, d'altra parte, poiché non è possibile elaborare contemporaneamente tutti i possibili tentativi di analisi conseguenti al non determinismo, in alcuni passi del processo di riconoscimento è necessario scegliere tra diverse alternative, col rischio che la scelta effettuata ad un certo istante risulti successivamente scorretta, e che quindi si debba ritornare indietro ad effettuare un'altra delle possibili scelte precedentemente inevase.

Per esempio, per la stringa  $x = 0111222$ , l'automa nella configurazione  $(q_0, 11222, AB)$  potrebbe scegliere la transizione  $\delta(q_0, 1, A) = (q_0, \lambda, 4)$ , che genera la configurazione  $(q_0, 1222, B)$  dalla quale non è definita alcuna mossa.

E' necessario ritornare alla configurazione precedente e scegliere la transizione alternativa  $\delta(q_0,1,A) = (q_0,A,3)$ .

Questo modo di procedere avanti-indietro (*backtracking*), caratteristico del comportamento non deterministico, è inaccettabile da un punto di vista pratico, poiché dà luogo a tempi di analisi sintattica che crescono in modo più che proporzionale con la lunghezza della stringa.

Nel campo della compilazione, in realtà, questi inconvenienti non sono molto frequenti, poiché le grammatiche che descrivono i convenzionali linguaggi di programmazione sono tali da consentire un'analisi sintattica deterministica.

Nel caso di un automa che ricostruisce l'albero sintattico della stringa dall'alto verso il basso e da sinistra a destra si parla di *analisi discendente* (equivale all'attraversamento dell'albero in ordine anticipato), in contrapposizione all'*analisi ascendente*, che ricostruisce l'albero sintattico dal basso verso l'alto (cioè in ordine differito).

Nell'analisi discendente l'algoritmo è predittivo, nel senso che quando si espande un non terminale  $A$  mediante la regola  $A \rightarrow A_1 A_2 \dots A_k$  si predice che dovranno essere incontrate successivamente nella stringa da analizzare anche le sottostringhe derivate da  $A_2 \dots A_k$ .

Al contrario, nell'analisi ascendente si effettuano riduzioni delle parti destre delle produzioni alle rispettive parti sinistre soltanto dopo che la parte destra è stata tutta esaminata.

Spesso il non determinismo degli analizzatori può essere eliminato se si consente la prospezione di due o più caratteri della stringa sorgente, prima di decidere la mossa dell'automata.

Nell'esempio precedente il non determinismo che si ha quando si legge 1 in ingresso e  $A$  è posto in cima alla pila, può essere sciolto se si leggono due caratteri in ingresso. Infatti, se il secondo carattere è un 1, deve essere applicata la transizione  $\delta(q_0,1,A) = (q_0,A,3)$ ; altrimenti, se esso è un 2, si utilizza  $\delta(q_0,1,A) = (q_0,\lambda,4)$ .

### **Grammatiche LL( $k$ ).**

Una grammatica LL( $k$ )<sup>1</sup> genera un linguaggio non contestuale la cui analisi può essere fatta da un algoritmo operante in modo discendente che:

- a) scandisce la stringa corrente da sinistra verso destra;
- b) costruisce la derivazione sinistra corrispondente all'albero sintattico;
- c) opera deterministicamente esaminando il carattere sotto la testina di lettura e non più di  $k-1$  caratteri successivi e consecutivi della stringa sorgente.

**Definizione 1** ( $First_k$ )

Sia  $\alpha = X_1X_2 \dots X_i$ ,  $i \geq 0$ ,  $\alpha \in (V \cup X)^*$ .

Definiamo  $First_k(\alpha)$ ,  $k \geq 0$ , come segue:

$$First_k(\alpha) = \left\{ w \in X^* \left| \begin{array}{l} |w| < k \text{ e } \alpha \xRightarrow{*} w \quad \text{oppure} \\ |w| = k \text{ e } \alpha \xRightarrow{*} wz, \text{ per qualche } z \end{array} \right. \right\}$$

Se  $\alpha \in X^*$ , allora:

$$First_k(\alpha) = \{ X_1X_2 \dots X_k \mid |\alpha| > k \} \cup \{ \alpha \mid |\alpha| \leq k \}^2$$

**Definizione 2** (Grammatica LL( $k$ ))

Una CFG  $G$  è LL( $k$ ),  $k \geq 1$ , se l'esistenza di due derivazioni:

- 1)  $S \xRightarrow{lm} wA\alpha \xRightarrow{lm} w\beta\alpha \xRightarrow{lm} wx$
- 2)  $S \xRightarrow{lm} wA\alpha \xRightarrow{lm} w\gamma\alpha \xRightarrow{lm} wy$

con  $First_k(x) = First_k(y)$ , implica  $\beta = \gamma$ .

Se  $G$  è LL( $k$ ), la regola che espande  $A$  deve essere identica in 1) e 2).

Informalmente,  $G$  è LL( $k$ ) se, data una stringa  $wA\alpha \in (V \cup X)^*$  ed i primi  $k$  simboli terminali (se esistono) derivati da  $A\alpha$ , esiste al più una produzione che può essere applicata ad  $A$  per fornire una derivazione di una stringa terminale che inizia con  $w$  ed è seguita da questi  $k$  terminali.

**Osservazione**

Se  $G$  è LL( $k$ ),  $G$  è a maggior ragione LL( $k'$ ), per  $k' > k$ .

Dunque si pone il problema di determinare il minimo  $k$  per cui  $G$  è LL( $k$ ).

Dalla definizione precedente si deduce immediatamente il seguente teorema.

**Teorema 1**

Una grammatica LL( $k$ ) non contiene ricorsioni sinistre.

<sup>1</sup> la sigla LL( $k$ ) indica che la stringa viene esaminata dalla sinistra – *Left to right* – e che viene ricostruita la derivazione sinistra – *Leftmost derivation*.

<sup>2</sup> si noti che i due insiemi sono disgiunti.

### **Esempio 2**

Si consideri la grammatica:

$$A \rightarrow Ab$$

$$A \rightarrow a$$

esistono allora le derivazioni:

$$1) \quad A \xRightarrow[*]{lm} Ab^j \xRightarrow{lm} ab^j, \quad j \geq 0$$

$$2) \quad A \xRightarrow[*]{lm} Ab^j \xRightarrow{lm} Ab^{j+1} \xRightarrow{lm} ab^{j+1}$$

Supponiamo, per assurdo, che  $G$  sia  $LL(k)$ ; preso allora  $j \geq k$ , le sottostringhe  $b^j$  e  $b^{j+1}$  coincidono per i primi  $k$  caratteri.

Ma, per la definizione di grammatica  $LL(k)$ , la regola che espande  $A$  dovrebbe essere identica in 1) e 2). Si ha dunque una contraddizione.

## Esercizi.

### Esercizio 1

Verificare che la seguente grammatica  $G$  è LL(1):

$$S \rightarrow aAS \mid b$$

$$A \rightarrow bSA \mid a$$

Intuitivamente  $G$  è LL(1) perché, detto 'C' il non terminale più a sinistra in ogni stringa e 'c' il successivo simbolo di ingresso, c'è al più una produzione per C capace di derivare una stringa di terminali per c.

In base alla definizione di LL( $k$ ), si ha:

$$\begin{aligned} S &\xRightarrow[lm]{*} wS\alpha \xRightarrow[lm]{*} waAS\alpha \xRightarrow[lm]{*} wax' \\ &\quad \quad \quad \underbrace{\quad}_{\beta} \quad \quad \quad \underbrace{\quad}_x \\ S &\xRightarrow[lm]{*} wS\alpha \xRightarrow[lm]{*} wb\alpha \xRightarrow[lm]{*} wby' \\ &\quad \quad \quad \underbrace{\quad}_{\gamma} \quad \quad \quad \underbrace{\quad}_y \end{aligned}$$

E' dunque impossibile che esistano due derivazioni distinte che generano una stringa che inizia per  $wa$  ( $wb$ ).

Specificamente, se  $x$  e  $y$  iniziano per  $a$  allora è stata utilizzata la produzione  $S \rightarrow aAS$  e  $\beta = \gamma = aAS$ .

Se  $x$  e  $y$  iniziano per  $b$ , la produzione usata deve essere  $S \rightarrow b$  e  $\beta = \gamma = b$ .

Si osservi che  $x = y = \lambda$  è impossibile, dato che  $\lambda$  non è derivabile da  $S$  in  $G$ .

Analogo ragionamento si può fare quando consideriamo le due derivazioni:

$$\begin{aligned} S &\xRightarrow[lm]{*} wA\alpha \xRightarrow[lm]{*} w\beta\alpha \xRightarrow[lm]{*} wx \quad \text{e} \\ S &\xRightarrow[lm]{*} wA\alpha \xRightarrow[lm]{*} w\gamma\alpha \xRightarrow[lm]{*} wy \end{aligned}$$

### Esercizio 2

Verificare che la seguente grammatica è LL(2):

$$S \rightarrow \lambda \mid abA$$

$$A \rightarrow Saa \mid b$$

Innanzitutto verifichiamo che  $G$  non sia LL(1).

Per fare ciò, dobbiamo determinare due derivazioni:

$$\begin{aligned} 1) \quad & S \xRightarrow[lm]{*} wA\alpha \xRightarrow[lm]{*} w\beta\alpha \xRightarrow[lm]{*} wx \\ 2) \quad & S \xRightarrow[lm]{*} wA\alpha \xRightarrow[lm]{*} w\gamma\alpha \xRightarrow[lm]{*} wy \end{aligned}$$

con  $First_1(x) = First_1(y)$  e  $\beta \neq \gamma$ .

Consideriamo le seguenti derivazioni:

$$\begin{array}{l}
 1) \quad S \xRightarrow{(2)} abA \xRightarrow{(3)} abS \xRightarrow{\alpha} aa \xRightarrow{(2)} ab \xRightarrow{\beta} abA \xRightarrow{\alpha} aa \xRightarrow{(4)} ab \xRightarrow{w} abbaa \\
 2) \quad S \xRightarrow{(2)} abA \xRightarrow{(3)} abS \xRightarrow{\alpha} aa \xRightarrow{(1)} ab \xRightarrow{\gamma=\lambda} aa \xRightarrow{(4)} ab \xRightarrow{w} aa
 \end{array}$$

Dunque si ha:

$$First_1(x) = First_1(abbaa) = \{a\} = First_1(aa) = First_1(y)$$

mentre  $\beta = abA \neq \lambda = \gamma$ ; e quindi  $G$  non è LL(1).

Mostriamo ora che  $G$  è LL(2).

Per far ciò, mostreremo che se  $wB\alpha$  è una qualunque stringa ottenuta attraverso una *leftmost derivation* e  $wx$  è una parola di  $L(G)$ , allora esiste al più una produzione  $B \rightarrow \beta$  in  $G$  tale che:  $First_2(\beta\alpha) \supseteq First_2(x)$ .

Supponiamo che  $B = S$  e che esistano le due derivazioni:

$$\begin{array}{l}
 1) \quad S \xRightarrow{lm}^* wS\alpha \xRightarrow{lm}^* w\beta\alpha \xRightarrow{*} wx \\
 2) \quad S \xRightarrow{lm}^* wS\alpha \xRightarrow{lm}^* w\gamma\alpha \xRightarrow{*} wy
 \end{array}$$

con  $First_2(x) = First_2(y)$ .

### Consideriamo $\alpha$ .

E' immediato osservare che  $\alpha$  deve essere una stringa di soli terminali.

Più precisamente, si ha una delle seguenti due condizioni:

o  $w = \alpha = \lambda$

oppure l'ultima produzione usata nella derivazione  $S \xRightarrow{lm}^* wS\alpha$  è stata la (3)  $A \rightarrow Saa$  (altrimenti  $S$  non potrebbe comparire nella stringa  $wB\alpha$ ) e dunque  $\alpha$  comincia per  $aa$ .

Supponiamo che si utilizzi la produzione:

$$(1) S \rightarrow \lambda$$

per andare da  $wS\alpha$  a  $w\beta\alpha$ .

Dunque:  $\beta = \lambda$  ed  $x$  o è la stringa vuota ( $x = \lambda$ ) oppure  $x$  comincia per  $aa$ .

Parimenti, se (1)  $S \rightarrow \lambda$  è utilizzata per andare da  $wS\alpha$  a  $w\gamma\alpha$ , allora  $\gamma = \lambda$  ed  $y = \lambda$  oppure  $y$  comincia per  $aa$ .

Supponiamo che si utilizzi la produzione:

$$(2) S \rightarrow abA$$

per andare da  $wS\alpha$  a  $w\beta\alpha$ .

Dunque:  $\beta = abA$  ed  $x$  inizia per  $ab$ .

Similmente, se (2)  $S \rightarrow abA$  è utilizzata per andare da  $wS\alpha$  a  $w\gamma\alpha$ , allora  $\gamma = abA$  ed  $y$  inizia per  $ab$ .

Per riassumere, non vi sono che tre possibilità:

- a)  $x = y = \lambda$
- b)  $x$  e  $y$  iniziano per  $aa$
- c)  $x$  e  $y$  iniziano per  $ab$

Qualunque altra possibilità sui primi due simboli di  $x$  e  $y$  rende impossibile una delle due o entrambe le derivazioni.

Casi a) e b):

- (1)  $S \rightarrow \lambda$  è utilizzata in entrambe le derivazioni e  $\beta = \gamma = \lambda$ .

Caso c):

- (2)  $S \rightarrow abA$  dev'essere utilizzata in entrambe le derivazioni e  $\beta = \gamma = abA$ .

Supponiamo ora che nella generica stringa  $wB\alpha$  sia  $B = A$  e che esistano le due derivazioni:

$$1) \quad S \xRightarrow[im]{*} wA\alpha \xRightarrow[im]{*} w\beta\alpha \xRightarrow[im]{*} wx$$

$$2) \quad S \xRightarrow[im]{*} wA\alpha \xRightarrow[im]{*} w\gamma\alpha \xRightarrow[im]{*} wy$$

con  $First_2(x) = First_2(y)$ .

Si osserva immediatamente che l'ultima produzione utilizzata in  $S \xRightarrow[im]{*} wA\alpha$  è necessariamente la (2)

$S \rightarrow abA$ .

Dunque, per  $\alpha$ , si ha certamente  $\alpha \in X^*$  ed inoltre:

( $w = ab$  e)  $\alpha = \lambda$  se la (2)  $S \rightarrow abA$  è l'unica produzione applicata, ossia:

$$S \xRightarrow[im]{} abA.$$

oppure:

$\alpha$  inizia per  $aa$  se sono state applicate prima le produzioni (2)  $S \rightarrow abA$  e poi ripetutamente in sequenza le produzioni (3)  $A \rightarrow Saa$  e (2)  $S \rightarrow abA$ :

$$S \xRightarrow[im]{(2)} abA \xRightarrow[im]{(3)} abSaa \xRightarrow[im]{(2)} (ab)^2 Aaa \xRightarrow[im]{*} \dots \xRightarrow[im]{(2)} (ab)^{i+1} A(aa)^i$$

Supponiamo che si utilizzi la produzione:

$$(3) \quad A \rightarrow Saa$$

per andare da  $wA\alpha$  a  $w\beta\alpha$ .

Dunque:  $\beta = Saa$  e ricadiamo nel caso analizzato in precedenza, in quanto  $x$  sarà generato da  $S$ .

Pertanto o  $x$  inizia per  $aa$  (se è utilizzata  $S \rightarrow \lambda$ )

Oppure  $x$  inizia per  $ab$ , per quanto visto in precedenza.

Similmente, se (3)  $A \rightarrow Saa$  è utilizzata per andare da  $wA\alpha$  a  $w\gamma\alpha$ , allora  $\gamma = Saa$  e si ha che  $y$  inizia per  $aa$  oppure  $y$  inizia per  $ab$ , per quanto visto in precedenza.

Supponiamo che si utilizzi la produzione:

$$(4) A \rightarrow b$$

per andare da  $wA\alpha$  a  $w\beta\alpha$ .

Dunque:  $\beta = b$  ed  $x$  inizia per  $b$  ( $x$  può essere o solo  $b$  oppure iniziare per  $ba$ , per le considerazioni fatte su  $\alpha$ ).

Parimenti, se (4)  $A \rightarrow b$  è utilizzata per andare da  $wA\alpha$  a  $w\gamma\alpha$ , allora  $\gamma = b$  e  $y$  inizia per  $b$ .

Per riassumere, le tre possibilità sono:

1.  $x$  e  $y$  iniziano per  $aa$
2.  $x$  e  $y$  iniziano per  $ab$
3.  $x$  e  $y$  iniziano per  $b$

Caso a):

la sequenza di produzioni (3)  $A \rightarrow Saa$  e (1)  $S \rightarrow \lambda$  è utilizzata necessariamente in entrambe le derivazioni e  $\beta = \gamma = Saa$ .

Caso b):

le produzioni (3)  $A \rightarrow Saa$  e (2)  $S \rightarrow abA$  sono usate necessariamente una o più volte in sequenza in entrambe le derivazioni e  $\beta = \gamma = Saa$ .

Caso c):

(4)  $A \rightarrow b$  deve essere usata in entrambe le derivazioni e  $\beta = \gamma = b$ .

### **Esercizio 3**

Verificare che la seguente grammatica è LL(2):

$$(1) \quad S \rightarrow aSA$$

$$(2) \quad S \rightarrow aAS$$

$$(3) \quad A \rightarrow cA$$

$$(4) \quad A \rightarrow ba$$

Occorre verificare innanzitutto che  $G$  non sia LL(1).

Aniché utilizzare direttamente la definizione di grammatica LL( $k$ ), possiamo usare la seguente caratterizzazione:

## **Teorema 2**

Una CFG è LL( $k$ ) se e solo se vale la seguente condizione:

se  $A \rightarrow \beta$  e  $A \rightarrow \gamma$  sono produzioni distinte in P allora

$$First_k(\beta\alpha) \cap First_k(\gamma\alpha) = \emptyset \text{ per ogni } wA\alpha \text{ tale che } S \xRightarrow{*}_{lm} wA\alpha.$$

Considero dunque le produzioni distinte:

$$(1) \quad S \rightarrow aSA$$

$$(2) \quad S \rightarrow aAS$$

e le derivazioni:

$$1) \quad S \xRightarrow{*}_{lm} wS\alpha \xRightarrow{(1)}_{lm} w \underbrace{aSA}_{\beta} \alpha$$

$$2) \quad S \xRightarrow{*}_{lm} wS\alpha \xRightarrow{(2)}_{lm} w \underbrace{aAS}_{\gamma} \alpha$$

Si ha:

$$First_1(\beta\alpha) = \{a\} = First_1(\gamma\alpha).$$

Dunque  $First_1(\beta\alpha)$  e  $First_1(\gamma\alpha)$  non sono disgiunti e  $G$  non è LL(1).

Mostriamo ora che  $G$  è LL(2):

Utilizziamo nuovamente la caratterizzazione vista in precedenza.

Considero nuovamente le due produzioni distinte di  $G$ :

$$(1) \quad S \rightarrow aSA$$

$$(2) \quad S \rightarrow aAS$$

e le derivazioni:

$$1) \quad S \xRightarrow{*}_{lm} wS\alpha \xRightarrow{(1)}_{lm} w \underbrace{aSA}_{\beta} \alpha$$

$$2) \quad S \xRightarrow{*}_{lm} wS\alpha \xRightarrow{(2)}_{lm} w \underbrace{aAS}_{\gamma} \alpha$$

Calcolo:

$$First_2(\beta\alpha) = First_2(aSA\alpha) = \{aa\}$$

in quanto le derivazioni da  $\beta\alpha$  in  $G$  sono solo dei seguenti due tipi:

$$\beta\alpha = aSA\alpha \xRightarrow{(1)}_{\underline{\underline{}}} aaSAA\alpha$$

$$\beta\alpha = aSA\alpha \xRightarrow{(2)}_{\underline{\underline{}}} aaASA\alpha$$

Calcolo:

$$First_2(\gamma\alpha) = First_2(aAS\alpha) = \{ac, ab\}$$

in quanto le derivazioni da  $\gamma\alpha$  in  $G$  sono solo dei seguenti due tipi:

$$\gamma\alpha = aAS\alpha \xrightarrow{(3)} acAS\alpha$$

$$\gamma\alpha = aAS\alpha \xrightarrow{(4)} abAS\alpha$$

Dunque:

$$First_2(\beta\alpha) = \{aa\} \cap \{ac, ab\} = First_2(\gamma\alpha) = \emptyset$$

Considero ora le due produzioni distinte che riscrivono  $A$ :

$$(3) \quad A \rightarrow cA$$

$$(4) \quad A \rightarrow ba$$

e le derivazioni:

$$1) \quad S \xrightarrow{lm} wA\alpha \xrightarrow{(3)} w \underbrace{cA}_{\beta} \alpha$$

$$2) \quad S \xrightarrow{lm} wA\alpha \xrightarrow{(4)} w \underbrace{ba}_{\gamma} \alpha$$

Calcolo  $First_2(\beta\alpha)$  e  $First_2(\gamma\alpha)$ :

$$First_2(\beta\alpha) = First_2(cA\alpha) = \{cc, cb\}$$

in quanto tutte le stringhe derivabili in  $G$  da  $\beta\alpha$  lo sono attraverso i due seguenti tipi di derivazioni:

$$\beta\alpha = cA\alpha \xrightarrow{(3)} ccA\alpha$$

$$\beta\alpha = cA\alpha \xrightarrow{(4)} cba\alpha$$

e

$$First_2(\gamma\alpha) = First_2(ba\alpha) = \{ba\}$$

Dunque:

$$First_2(\beta\alpha) = \{cc, cb\} \cap \{ba\} = First_2(\gamma\alpha) = \emptyset$$

e  $G$  è LL(2), in quanto non vi sono altre coppie di produzioni di  $G$  da considerare per verificare la condizione della caratterizzazione delle grammatiche LL( $k$ ).

### **Proposizioni**

Data una grammatica non contestuale  $G$ , valgono le seguenti proposizioni:

- è **decidibile** il problema di stabilire se  $G$  è  $LL(k)$ , fissato  $k \geq 1$  (ossia esiste un algoritmo che decide se  $G$  è  $LL(k)$ );
- è **indecidibile** il problema di stabilire l'esistenza di un  $k$  finito per cui  $G$  è  $LL(k)$ ;
- è **indecidibile** il problema di stabilire se, per una data  $G$  che non è  $LL(1)$ , esiste  $G'$  equivalente a  $G$  ( $L(G) = L(G')$ ) e che sia  $LL(1)$ .

L'ultimo risultato ha notevole interesse pratico, in quanto gli analizzatori sintattici per grammatiche  $LL(1)$  sono particolarmente efficienti e dunque sarebbe stato importante stabilire l'esistenza di una procedura meccanica (algoritmo) che, se esiste, genera una grammatica  $G'$  di tipo  $LL(1)$  equivalente ad una data grammatica  $G$ .

Questo risultato non esclude la possibilità di applicare metodi euristici per effettuare questa trasformazione, come illustrato dal seguente esercizio.

#### **Esercizio 4**

Sia data la seguente grammatica non contestuale che genera espressioni aritmetiche:

$$G_E = (X, V, S, P)$$

dove:

$$X = \{i, (, ), +, -, *, /\}$$

$$V = \{E, T, F\} \quad E = \text{espressione}; T = \text{termine}; F = \text{fattore}$$

$$S = E$$

$P$ :

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

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

$$F \rightarrow i \mid (E)$$

Determinare, se esiste, una grammatica equivalente a  $G_E$  e che sia LL(1).

$G_E$  non è LL(1). (Dimostrarlo per esercizio).

Intuitivamente,  $G_E$  non è LL(1) perché esistono più parti destre nelle riscritture di uno stesso nonterminale (ossia nelle produzioni che hanno uno stesso nonterminale come parte sinistra) che iniziano con lo stesso carattere.

Per eliminare ciò, si opera una **fattorizzazione** (primo metodo euristico): essa consiste nell'utilizzare la proprietà distributiva della concatenazione di caratteri rispetto al simbolo “ | ” del metalinguaggio usato per descrivere le produzioni di una grammatica e nell'introdurre nuovi nonterminali.

**Esempio:**  $E \rightarrow E+T \mid E-T$  viene fattorizzato come segue:  $E \rightarrow E \cdot (+T \mid -T)$  ed introducendo un nuovo nonterminale, si ha:

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

Per fattorizzazione si ottiene una grammatica  $G'_E$  equivalente a  $G_E$ :

$$\begin{aligned} G'_E \\ E &\rightarrow T \mid EE' \\ E' &\rightarrow +T \mid -T \\ T &\rightarrow F \mid TT' \\ T' &\rightarrow *F \mid /F \\ F &\rightarrow i \mid (E) \end{aligned}$$

Trasformiamo ora  $G'_E$  in **forma normale priva di ricorsioni sinistre** (*NLR normal form*), eliminando le ricorsioni sinistre per  $E$  e per  $T$ , secondo l'algoritmo visto in precedenza:

$$A \rightarrow A\alpha \mid \beta$$

diventa:

$$\begin{aligned} A &\rightarrow \beta B \\ B &\rightarrow \alpha B \mid \lambda \end{aligned}$$

Otteniamo così  $G''_E$  equivalente a  $G'_E$ :

$$\begin{aligned} E &\rightarrow TE'' \\ E'' &\rightarrow E'E'' \mid \lambda \\ E' &\rightarrow +T \mid -T \\ T &\rightarrow FT'' \\ T'' &\rightarrow T'T'' \mid \lambda \\ T' &\rightarrow *F \mid /F \\ F &\rightarrow i \mid (E) \end{aligned}$$

$G''_E$  è LL(1). (Dimostrarlo per esercizio).

Le regole empiriche sopra impiegate per trasformare la grammatica  $G_E$ , nella speranza di portarla in una forma LL(1), cioè la fattorizzazione e la eliminazione delle produzioni ricorsive a sinistra, sono gli strumenti fondamentali che si impiegano per operare la trasformazione desiderata.

### Grammatiche LR(k).

Dopo aver visto le grammatiche LL( $k$ ), che consentono una analisi sintattica deterministica discendente (*top-down*), consideriamo una analoga classe di grammatiche, dette LR( $k$ )<sup>3</sup>, che consentono ancora una analisi deterministica, ma ascendente (*bottom-up*).

Supporremo nel seguito che, come avviene di solito, un metodo di analisi sintattica bottom-up sia in grado di ottenere l'albero sintattico ricostruendo la derivazione destra, in forma di stringa di cifre corrispondente alla sequenza di produzioni applicate.

Se:

$$S = \alpha_0 \xrightarrow{rm} \alpha_1 \xrightarrow{rm} \dots \xrightarrow{rm} \alpha_i \xrightarrow{rm} \alpha_{i+1} \xrightarrow{rm} \dots \xrightarrow{rm} \alpha_n = w$$

è la derivazione destra della stringa  $w$ , l'analisi sintattica ascendente consiste nel determinare la produzione che permette di passare da  $\alpha_{i+1}$  ad  $\alpha_i$  per  $i = n-1, n-2, \dots, 0$ .

Se  $\alpha_i = \alpha A v$  e  $\alpha_{i+1} = \alpha \beta v$ , si dice che  $\beta$  viene ridotto ad  $A$  e che  $\beta$  è la *chiave* o *parte riducibile* di  $\alpha_{i+1}$ .

Per determinare ad ogni passo della derivazione la chiave, si utilizza un metodo che scandisce i caratteri della stringa da analizzare da sinistra a destra e li sposta in cima ad una pila, finché sulla pila non compare la chiave, cui viene allora sostituita la parte sinistra.

Questo processo si ripete finché non è più possibile procedere perché la stringa non appartiene al linguaggio considerato (si è cioè rilevato un errore) o tutta la stringa è stata scandita e sulla pila è presente solo l'assioma  $S$ . In tal caso la stringa è accettata o riconosciuta.

Il processo suddetto prende il nome di **algoritmo** (di analisi sintattica) **a spostamento e riduzione**.

Formalmente, si dà la seguente:

**Definizione 3** (Algoritmo a spostamento e riduzione)

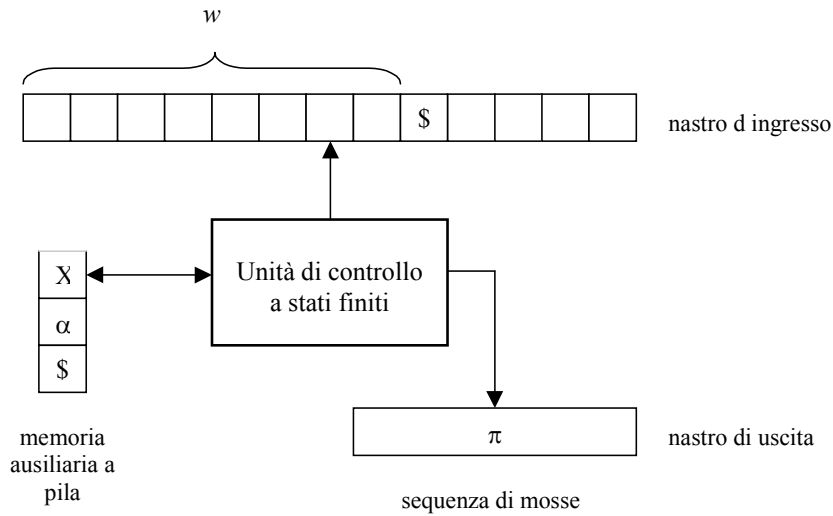
Un algoritmo di analisi sintattica a spostamento e riduzione per una grammatica non contestuale  $G = (X, V, S, P)$ , le cui produzioni sono etichettate da 1 a  $p$ , è definito da una funzione parziale:

$$f : (\{\$\} \cdot (X \cup V)^*) \times (X^* \cdot \{\$\}) \rightarrow \{sposta, riduci\ 1, \dots, riduci\ p\}$$

$\$$  funge da marcatore di fine pila e da marcatore di fine stringa sul nastro di ingresso.

---

<sup>3</sup> la sigla LR( $k$ ) indica che, scandendo la stringa da analizzare da sinistra verso destra — *Left to right* —, ogni passo nella costruzione della derivazione destra — *Rightmost derivation* — è determinato esaminando i primi  $k$  caratteri della parte di stringa non ancora scandita.



Intuitivamente, si ha  $f = riduci\ i$ , se sulla pila è presente come chiave la parte destra della produzione  $i$ -esima di  $G$ .

Sia  $x$  la stringa da analizzare, delimitata a destra da  $\$$ ; una configurazione dell'algorithm è definita da una tripla:

$$(\$ \alpha, w \$, \pi)$$

dove:

- $\$$  è il simbolo inizialmente presente sulla pila;
- $\$ \alpha = \$ X_1 X_2 \dots X_m$  è il contenuto della pila (con in cima  $X_m$ ), dove  $X_i \in X \cup V$ ,  $i = 1, 2, \dots, m$ ,  $m > 0$ ;
- $w = a_1 a_2 \dots a_n$  è la parte di stringa ancora da analizzare (con  $a_1$  sotto la testina di lettura), dove  $a_i \in X$ ,  $i = 1, 2, \dots, n$ ,  $n \geq 0$ ;
- $\pi = p_1 p_2 \dots p_q$  è la stringa associata alla sequenza delle etichette delle produzioni impiegate nella derivazione  $\alpha w \xRightarrow{*}_{rm} x$ , dove  $p_i \in \{1, 2, \dots, p\}$  per  $1 \leq i \leq q$ ,  $q \geq 0$ .

La configurazione:

$(\$, x\$, \lambda)$  è detta iniziale

$(\$, \$, \pi)$  è detta finale

Il passaggio da una configurazione ad una successiva è definito dalle seguenti operazioni:

- se  $f(\$, w\$) = \text{sposta}$  allora  $(\$, w\$, \pi) \Rightarrow (\$a_1, w'\$, \pi)$  dove  $w = a_1w'$ .
- se  $f(\$, w\$) = \text{riduci } i$  allora  $(\$, w\$, \pi) \Rightarrow (\$A', w\$, \pi i)$  dove  $\alpha = \alpha'\beta$  e  $i: A \rightarrow \beta \in P$ ;
- se  $f(\$, w\$)$  non è definita, l'algoritmo si ferma.

Una stringa  $w$  è accettata se  $(\$, w\$, \lambda) \xRightarrow{*} (\$, \$, \pi)$ .

In tal caso:  $S \xRightarrow[\pi^k]{*} w$  cioè  $\pi$  è l'inversa della stringa associata alla derivazione destra di  $w$ .

Un algoritmo a spostamento e riduzione è corretto per una grammatica  $G$  se

$$L(G) = \{w \mid w \text{ è accettata dall'algoritmo}\}.$$

### **Esempio 3**

Si consideri la grammatica:

- (1)  $E \rightarrow T$
- (2)  $E \rightarrow E + T$
- (3)  $T \rightarrow d$
- (4)  $T \rightarrow T * d$

L'algoritmo a spostamento e riduzione è il seguente:

supponiamo:

$$\begin{aligned}\alpha &\in \{+, *, d, E, T\}^* \\ y &\in \{+, *, d\}^* \cdot \{\$\} \\ a &\in \{+, *, \$\}\end{aligned}$$

Configurazione	$f$
$(\$, dy)$	<i>sposta</i>
$(\$d, ay)$	<i>riduci 3</i>
$(\$\alpha T, +y)$	<i>riduci 1</i>
$(\$\alpha+d, ay)$	<i>riduci 3</i>
$(\$\alpha T*d, ay)$	<i>riduci 4</i>
$(\$\alpha^*, dy)$	<i>sposta</i>
$(\$\alpha T, *y)$	<i>sposta</i>
$(\$\alpha+, dy)$	<i>sposta</i>
$(\$\alpha E, +y)$	<i>sposta</i>
$(\$T, \$)$	<i>riduci 1</i>
$(\$E+T, +y)$	<i>riduci 2</i>
$(\$E+T, \$)$	<i>riduci 2</i>

Vediamo l'analisi della stringa  $d+d$ :

$$\begin{aligned}
 (\$, d + d\$, \lambda) &\Rightarrow (\$d, +d\$, \lambda) \Rightarrow (\$T, +d\$, 3) \Rightarrow (\$E, +d\$, 31) \Rightarrow (\$E, d\$, 31) \Rightarrow \\
 &\Rightarrow (\$E + d, \$, 31) \Rightarrow (\$E + T, \$, 313) \Rightarrow (\$E, \$, 3132)
 \end{aligned}$$

Dunque, in corrispondenza della derivazione destra:

$$\begin{array}{ccccccc}
 E & \Rightarrow & E + T & \Rightarrow & E + d & \Rightarrow & T + d & \Rightarrow & d + d \\
 & & \text{(2)} & & \text{(3)} & & \text{(1)} & & \text{(3)} \\
 & & \textit{rm} & & \textit{rm} & & \textit{rm} & & \textit{rm}
 \end{array}$$

si ha la stringa associata:

$$2313 = (3132)^R$$

In generale, per la determinazione della mossa dell'algorithm a spostamento e riduzione, può essere necessario consultare l'intero contenuto della pila e quanto rimane della stringa da analizzare.

Per rendere ragionevolmente efficiente l'algorithm, si cerca di ridurre al minimo possibile le parti di pila e stringa da consultare.

**Definizione 4** ( Grammatica LR( $k$ ) )

Una CFG  $G$  è LR( $k$ ),  $k \geq 1$ , se l'esistenza di due derivazioni:

$$\begin{aligned}
 1) \quad & S \xRightarrow[rm]{*} \alpha A w \xRightarrow[rm]{*} \alpha \beta w \\
 2) \quad & S \xRightarrow[rm]{*} \gamma B x \xRightarrow[rm]{*} \alpha \beta y
 \end{aligned}$$

con  $First_k(y) = First_k(w)$  implica  $\alpha = \gamma$ ,  $A = B$  e  $x = y$ .

**Definizione 5** (*Augmented grammar*)

Sia  $G = (X, V, S, P)$  una CFG.

La grammatica  $G' = (X, V \cup \{S'\}, S', P \cup \{S' \rightarrow S\})$  è, per definizione, la *augmented grammar* derivata da  $G$ .

La augmented grammar  $G'$  è semplicemente  $G$  con una nuova produzione di partenza  $S' \rightarrow S$ , ove  $S'$  è un nuovo assioma.

Aggiungiamo tale produzione di partenza in modo che quando è richiesta una riduzione che utilizza questa produzione, possiamo interpretare questa "riduzione" come un segnale di accettazione.

**Esercizio 5**

Dimostrare che la seguente grammatica è LR(1):

$$\begin{aligned}
 S &\rightarrow aS \mid bA \\
 A &\rightarrow cA \mid c
 \end{aligned}$$

Poiché  $S$  appare a destra nella produzione  $S \rightarrow aS$ , considero l'augmented grammar derivata da  $G$ :

$$\begin{aligned}
 G' &= (X, V \cup \{S'\}, S', P') \\
 P' : \\
 S' &\rightarrow S \\
 S &\rightarrow aS \mid bA \\
 A &\rightarrow cA \mid c
 \end{aligned}$$

Per definizione:

$$G \text{ è LR}(1) \stackrel{def}{\Leftrightarrow} \left. \begin{aligned}
 (i) \quad & S' \xRightarrow[rm]{*} \alpha A' w \xRightarrow[rm]{*} \alpha \beta w \\
 (ii) \quad & S' \xRightarrow[rm]{*} \gamma B x \xRightarrow[rm]{*} \alpha \beta y \\
 (iii) \quad & First_1(y) = First_1(w)
 \end{aligned} \right\} \Rightarrow \begin{aligned}
 &\alpha = \gamma \\
 &A' = B \\
 &x = y
 \end{aligned}$$

1° caso)

Considero  $A' = S'$  allora  $\alpha = w = \lambda$ .

Inoltre da  $S'$  derivo solo  $S$  ( $S' \rightarrow S$ ) in quanto  $\underbrace{S'}_{\alpha} \Rightarrow \underbrace{S}_{\beta}$ .

Poiché  $First_1(y) = First_1(w) = \{\lambda\}$  si ha:  $y = \lambda$ .

Considero:  $\underbrace{\gamma Bx}_{\alpha \beta y} \Rightarrow \underbrace{S}_{\alpha \beta y}$

$S$  può essere prodotto da  $S' \rightarrow S$  oppure  $S \rightarrow aS$ .

Ma nel secondo caso  $\beta = S$  dovrebbe essere preceduto da "a". Contraddizione.

Dunque l'unica produzione possibile è  $S' \rightarrow S$  allora

$$B = S' = A' \text{ e } \underbrace{\gamma Bx}_{\alpha \beta y} \Rightarrow \underbrace{S}_{\alpha \beta y}$$

da cui:  $\gamma = \lambda = \alpha$  e  $x = \lambda = y$ .

2° caso)  $A' = S$

Si ha:

$$\underbrace{S'}_{\alpha} \xRightarrow{*} \underbrace{a^k S}_{\alpha A' w} \quad \alpha = a^k, k \geq 0, w = \lambda$$

Poiché  $First_1(w) = \{\lambda\} = First_1(y)$  si ha  $y = \lambda$  e ad  $S$  posso applicare  $S \rightarrow aS$  oppure  $S \rightarrow bA$ .

i) Se considero  $S \rightarrow \underbrace{aS}_{\beta}$

$$\underbrace{S'}_{\alpha} \xRightarrow{*} \underbrace{a^k S}_{\alpha A' w} \Rightarrow \underbrace{a^k aS}_{\alpha \beta w}$$

Considero:

$$\underbrace{S'}_{\alpha} \xRightarrow{*} \underbrace{\gamma Bx}_{\alpha \beta y} \Rightarrow \underbrace{a^k aS y}_{\alpha \beta y}$$

$\beta$  si può ottenere solo da  $S \rightarrow aS$  perché se usassi  $S' \rightarrow S$  allora dovrei avere  $\beta = S$ , quindi

$$B = S = A' \text{ e } \underbrace{S'}_{\alpha} \xRightarrow{*} \underbrace{\gamma Sx}_{\alpha \beta y} \Rightarrow \underbrace{a^k aS}_{\alpha \beta y} \text{ da cui: } \gamma = a^k = \alpha \text{ e } x = \lambda = y.$$

ii) Se considero  $S \rightarrow \underbrace{bA}_{\beta}$  :

$$S' \xRightarrow{rm} a^k S \xRightarrow{rm} a^k bA$$

$$S' \xRightarrow{rm} \gamma Bx \xRightarrow{rm} a^k bA$$

$A$  può essere prodotto da  $S \rightarrow bA$  oppure da  $A \rightarrow cA$ , ma nel secondo caso  $A$  non sarebbe preceduto da “ $b$ ” quindi  $B = S = A'$  allora:

$$S' \xRightarrow{rm} \gamma Sx \xRightarrow{rm} a^k bA$$

da cui:

$$\gamma = a^k = \alpha \quad e \quad x = \lambda = y.$$

3° caso)  $A' = A$

$$S' \xRightarrow{rm} a^k S \xRightarrow{rm} a^k bA \xRightarrow{rm} a^k bc^n A \quad k, n \geq 0$$

allora  $\alpha = a^k bc^n$ ,  $k, n \geq 0$ ,  $w = \lambda$ .

Inoltre  $First_1(w) = First_1(y) = \{\lambda\} \Rightarrow y = \lambda$ .

Da  $A$  posso applicare  $A \rightarrow cA$  oppure  $A \rightarrow c$

i) Considero  $A \rightarrow \underbrace{cA}_{\beta}$

$$S' \xRightarrow{rm} a^k bc^n A \xRightarrow{rm} a^k bc^n cA$$

Considero:

$$S' \xRightarrow{rm} \gamma Bx \xRightarrow{rm} a^k bc^n cA$$

$A$  può essere prodotto da  $A \rightarrow cA$  o da  $S \rightarrow bA$ , ma nel secondo caso  $\beta \neq cA$  perché  $A$  sarebbe preceduto da “ $b$ ”, quindi posso applicare solo  $A \rightarrow cA$ , cioè  $B = A = A'$ .

$$S' \xRightarrow{rm} \gamma Ax \xRightarrow{rm} a^k bc^n cA$$

da cui  $\gamma = a^k bc^n = \alpha$  e  $x = \lambda = y$ .

ii) Considero  $A \rightarrow \underbrace{c}_{\beta}$

$$S' \xRightarrow{*} \underbrace{a^k bc^n}_{\alpha} \underbrace{A}_{A'} \underbrace{w}_{w} \Rightarrow \underbrace{a^k bc^n}_{\alpha} \underbrace{c}_{\beta} \underbrace{w}_{w}$$

Considero:

$$S' \xRightarrow{*} \gamma \underbrace{Bx}_{\alpha} \Rightarrow \underbrace{a^k bc^n}_{\alpha} \underbrace{c}_{\beta} \underbrace{y}_{y}$$

“c” può essere prodotto da  $A \rightarrow c$  oppure da  $A \rightarrow cA$ . Ma nel secondo caso “c” sarebbe seguito da A. Dunque posso applicare solo  $A \rightarrow c$ .

Allora  $B = A = A'$  e  $S' \xRightarrow{*} \gamma Ax \Rightarrow \underbrace{a^k bc^n}_{\alpha} \underbrace{c}_{\beta} \underbrace{y}_{y}$  da cui:

$$\gamma = a^k bc^n = \alpha \text{ e } x = \lambda = y.$$

Per cui  $G$  è LR(1).

## Esercizi proposti.

### Esercizio 1

Sia data la seguente grammatica:

$$G = (X, V, S, P)$$

$$\text{ove } X = \{a, b, c, d\}$$

$$V = \{S, A, B\}$$

$$P = \{ S \rightarrow SaA \mid A,$$

$$A \rightarrow AbB \mid B,$$

$$B \rightarrow dSd \mid c \}$$

Descrivere il linguaggio  $L(G)$  generato da  $G$ .

Indicare se  $G$  è  $LL(k)$  per un certo valore di  $k$ .

Si trasformi  $G$  in una grammatica  $G'$  priva di ricorsioni sinistre.

Indicare se  $G'$  è  $LL(k)$  per un certo valore di  $k$ .

### Esercizio 2

Sia data la seguente grammatica:

$$G = (X, V, E, P)$$

$$\text{ove } X = \{Id, (, ), +, *\}$$

$$V = \{E, E', T, T', F\}$$

$$P = \{ E \rightarrow TE',$$

$$E' \rightarrow +TE' \mid \lambda,$$

$$T \rightarrow FT',$$

$$T' \rightarrow *FT' \mid \lambda,$$

$$F \rightarrow Id \mid (E) \}$$

Descrivere il linguaggio  $L(G)$ .

Indicare se  $G$  è  $LL(k)$  per un certo valore di  $k$ .

Giustificare formalmente la precedente risposta.

Descrivere un algoritmo di parsing con stack esplicito per la grammatica  $G$ .

### **Esercizio 3**

Sia data la seguente grammatica:

$$G = (X, V, S, P)$$

$$\text{ove } X = \{a, b, c, d, e\}$$

$$V = \{S, A, B\}$$

$$P = \{ S \rightarrow aA \mid cB ,$$

$$A \rightarrow Sb \mid b,$$

$$B \rightarrow Bd \mid e \}$$

Costruire l'albero di derivazione per la parola  $w = aacedddb$ .

Indicare se  $G$  è  $LL(k)$  per un certo valore di  $k$ .