

PARSER TOP-DOWN

Presentiamo un primo esempio di parser top-down, non deterministico, che usa implicitamente una pila per gestire le chiamate ricorsive

Parser a discesa ricorsiva

Data una grammatica libera $G = (NT, T, S, R)$, per ogni nonterminale A con produzioni:

$$A \rightarrow X_1^1 \dots X_{n_1}^1 \mid \dots \mid X_1^k \dots X_{n_k}^k$$

definisce la funzione

function $A()$ {

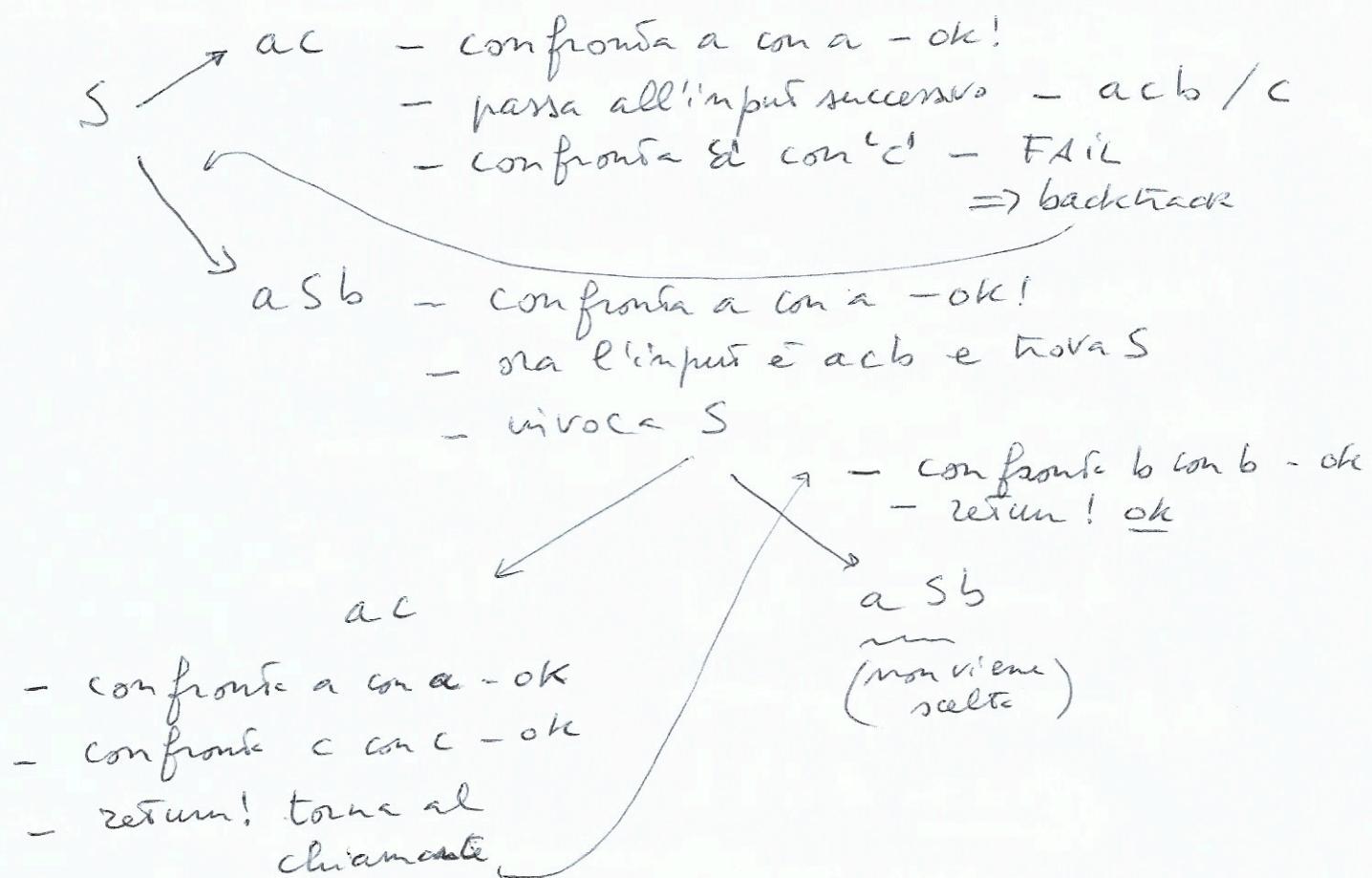
- scegli nondeterministicamente h tra 1 e k ,
 - ovvero una produzione $A \rightarrow X_1^h \dots X_{n_h}^h$;
 - for $i=1$ to n_h {
 - if $X_i^h \in NT$ then $X_i^h()$;
 - else if $X_i^h = \text{simbolo corrente}$ dell'input
 - then avanza di un simbolo sull'input
 - else Fail();
 - }
 - }
- return;
- }
- } }
backtracking!
si torna al punto
e si sceglie
un'altra produzione

Si comincia invocando la funzione per il simbolo iniziale S

Esempio $S \rightarrow a\text{c} \mid aSb$ $L = \{a^{n+1}c b^n \mid n \geq 0\}$ (2)

input aacb

Si invoca la funzione del simbolo iniziale S



Se lo vediamo come una pile ...

input
aacb
-

ac**b**

Stack delle chiamate

S

ac

c fail

aa**c**b

S

aS**b**

s**b**

a**c****b**

c**b**

b

ok

Parser a discesa ricorsiva è molto inefficiente;

(3)

- Nondeterminismo:

necessità di esplorare, nel caso peggiore, tutte le alternative!

⇒ esponenziale nella lunghezza della stringa w ,
dove la base b è data dal massimo numero
di produzioni per uno stesso non terminale

$$O(b^{|w|})$$

⇒ cerchiamo di guidare la scelta della produzione

Come?

Guardando il prossimo carattere (o i "prossimi" caratteri) dell'input da leggere

Esempio 1) $A \rightarrow aB \mid bC$

⇒ se il prossimo input è a , uso $A \rightarrow aB$

⇒ se il prossimo input è b , uso $A \rightarrow bC$

2) $S \rightarrow ac \mid aSb$

⇒ se i prossimi 2 input sono ac , uso $S \rightarrow ac$

⇒ se i prossimi 2 input sono ab , uso $S \rightarrow aSb$

⇒ introduciamo 2 funzioni ausiliarie:

- First

- Follow

Attention: Da qui in poi assumeremo di avere un simbolo speciale $\$$, che non faccia parte dei simboli di nessuna grammatica, che useremo come segnalatore di fine input! (4)

Perché? Un parser top-down è essenzialmente un DPDA che riconosce per pila vuota. Allora è necessario che L gode delle prefix property, e $L \cdot \$$ sicuramente gode di questa proprietà.

FIRST

Data una grammatica libera G e $\alpha \in (T \cup NT)^*$, diciamo che $\text{First}(\alpha)$ è l'insieme dei terminali che possono stare in prima posizione in una stringa che si deriva da α .

- per $a \in T$, $a \in \text{First}(\alpha)$ se $\alpha \Rightarrow^* a\beta$ per $\beta \in (T \cup NT)^*$
 - inoltre se $\alpha \Rightarrow^* \varepsilon$, allora $\varepsilon \in \text{First}(\alpha)$
-

Ese: $A \rightarrow \alpha_1 \mid \alpha_2$

se $\text{First}(\alpha_1) \cap \text{First}(\alpha_2) = \emptyset$, allora la scelta della produzione è deterministica!

$A \rightarrow aB \mid bC$

$\text{First}(aB) = \{a\}$ } \Rightarrow determinismo con
 $\text{First}(bC) = \{b\}$ } un solo carattere in lettura

Ma il First può non bastare!

$$\text{Ese: } S \rightarrow A b \mid c \\ A \rightarrow a A \mid \epsilon$$

$$\begin{aligned} \text{First}(Ab) &= \{a, b\} && \text{perché } A \xrightarrow{\epsilon} \epsilon \\ \text{First}(c) &= \{c\} \\ \bullet \text{First}(Ab) \cap \text{First}(c) &= \emptyset \end{aligned}$$

$$\begin{aligned} \text{First}(aA) &= \{a\} \\ \text{First}(\epsilon) &= \{\epsilon\} \\ \bullet \text{First}(aA) \cap \text{First}(\epsilon) &= \emptyset \end{aligned}$$

Però, se l'input è ab

$$\begin{array}{c} S \xrightarrow{a} \{ \} \\ \text{simboli} \\ \text{in lettura} \\ \text{sull'input} \\ A \xrightarrow{a} \stackrel{a}{=} A \\ \text{consumo } a \\ \downarrow \\ A \xrightarrow{b} ? \quad b \notin \text{First}(aA) \\ \qquad \qquad b \notin \text{First}(\epsilon) \end{array}$$

\Rightarrow quale produzione applicare?

Devo vedere cosa può "seguire" A: nel nostro esempio, A è sempre seguito da b

\Rightarrow devo scegliere $A \rightarrow \epsilon$!

Come puile...

input	Stack
a b \$	\leftarrow S
	Ab
	a Ab
b \$	Ab
	b
\$	ϵ

Follow

Data una grammatica libra G e $A \in NT$,
 diciamo che $\text{Follow}(A)$ è l'insieme dei terminali
 che possono comparire immediatamente a destra
 di A in una forma sentenziale.

- $a \in \text{Follow}(A)$ se $S \Rightarrow^* \alpha A \beta$
 per qualche $\alpha, \beta \in (T \cup NT)^*$
 - $\$ \in \text{Follow}(A)$ se $S \Rightarrow^* \alpha A$
 (Poiché $S \Rightarrow^* S$, allora $\$ \in \text{Follow}(S)$!)
-

Esempio

$$S \rightarrow Ab \mid c$$

$$A \rightarrow aA \mid \epsilon$$

$$\text{Follow}(S) = \{ \$ \}$$

$$\text{Follow}(A) = \{ b \}$$

$$\hookrightarrow S \Rightarrow^* S$$

$$\hookrightarrow S \Rightarrow^* Ab$$

Come calcolare First?

(7)

Sia $N(G) \subseteq NT$ l'insieme dei simboli annullabili
 $(A \in N(G) \text{ se } A \Rightarrow^* \epsilon)$

- Per ogni $x \in T$, $\text{First}(x) = \{x\}$
- Per ogni $X \in NT$, inizializza $\text{First}(X) = \emptyset$,
- Ripeti il seguente ciclo finché nessun $\text{First}(X)$ viene più modificato in una iterazione:
 - Per ogni produzione $X \rightarrow Y_1 \dots Y_k$
 - per ogni i da 1 a k
 - se $(Y_1, \dots, Y_{i-1} \in N(G))$ / true se $i=1$
 - allora $\text{First}(X) := \text{First}(X) \cup (\text{First}(Y_i) \setminus \{\epsilon\})$
- Per ogni $X \in N(G)$, $\text{First}(X) := \text{First}(X) \cup \{\epsilon\}$

First può essere estesa ad $\alpha \in (T \cup NT)^*$ come segue:

- $\text{First}(\epsilon) = \{\epsilon\}$
- $\text{First}(X\beta) = \text{First}(X) \quad \text{se } X \notin N(G)$
- $\text{First}(X\beta) = (\text{First}(X) \setminus \{\epsilon\}) \cup \text{First}(\beta) \quad \text{se } X \in N(G)$

In pratica, se $A \rightarrow \alpha_1 \dots \alpha_k$, allora

$$\text{First}(A) = \text{First}(\alpha_1) \cup \dots \cup \text{First}(\alpha_k)$$

Esempio

(8)

$$S \rightarrow Ab \mid c$$

$$\text{First}(S) = \text{First}(Ab) \cup \text{First}(c)$$

$$A \rightarrow a \mid \epsilon$$

$$= (\text{First}(A) \setminus \{\epsilon\}) \cup \text{First}(b) \cup \{\epsilon\}$$

$$= \{a\} \cup \{b\} \cup \{\epsilon\} = \{a, b, \epsilon\}$$

$$\text{First}(A) = \text{First}(a A) \cup \text{First}(\epsilon)$$

$$= \{a\} \cup \{\epsilon\} = \{a, \epsilon\}$$

$$S \rightarrow ABC \mid CB$$

$$A \rightarrow a \mid \epsilon$$

$$\text{First}(A) = \{a, \epsilon\}$$

$$B \rightarrow b \mid \epsilon$$

$$\text{First}(B) = \{b, \epsilon\}$$

$$C \rightarrow c$$

$$\text{First}(C) = \{c\}$$

$$\text{First}(S) = \text{First}(ABC) \cup \text{First}(CB)$$

$$= (\text{First}(A) \setminus \{\epsilon\}) \cup \text{First}(BC) \cup \text{First}(C)$$

$$= \{a\} \cup (\text{First}(B) \setminus \{\epsilon\}) \cup \text{First}(C) \cup \{c\}$$

$$= \{a\} \cup \{b\} \cup \{c\} = \{a, b, c\}$$

Procedura per calcolare $\text{Follow}(X)$ (con XENT) (9)

- Per ogn. $X \in NT$, inizializza $\text{Follow}(X) := \emptyset$
- $\text{Follow}(S) := \{\$\}$
- Ripeti il seguente ciclo finché nessun $\text{Follow}(X)$ viene più modificato in una iterazione:
 - 1) Per ogni produzione $X \rightarrow \alpha Y \beta$
 $\text{Follow}(Y) := \text{Follow}(Y) \cup (\text{First}(\beta) \setminus \{\epsilon\})$
 - 2) Per ogni produzione $X \rightarrow \alpha Y$ e per ogni produzione $X \rightarrow \alpha Y \beta$ con $\epsilon \in \text{First}(\beta)$
 $\text{Follow}(Y) := \text{Follow}(Y) \cup \text{Follow}(X)$

In pratica, bisogna cercare tutte le produzioni in cui $X \in NT$ appare ϵ , per ognuna di esse, applicare la 1 o la 2 sopra.

$$S \rightarrow ABC$$

$$A \rightarrow aaA | \epsilon$$

$$B \rightarrow b | \epsilon$$

$$C \rightarrow ccC | \epsilon$$

	First	Follow
S	a, b, c, ϵ	\$
A	a, ϵ	b, c, \$
B	b, ϵ	c, \$
C	c, ϵ	\$

Poiché $S \rightarrow ABC$, il $\text{Follow}(B) \supseteq \text{First}(C) \setminus \{\epsilon\}$
 ed anche $\text{Follow}(B) \supseteq \text{Follow}(S)$
 poiché $\epsilon \in \text{First}(C)$!

$$E \rightarrow TE'$$

$$E' \rightarrow \epsilon \mid + E \mid - E$$

$$T \rightarrow AT'$$

$$T' \rightarrow \epsilon \mid * T$$

$$A \rightarrow a \mid b \mid (E)$$

First

Follow

E	a, b, (\$,)
E'	$\epsilon, +, -$	\$,)
T	a, b, (\$,), +, -
T'	$\epsilon, *$	\$,), +, -
A	a, b, (\$,), +, -, *

\$ perché \$ è il simbolo finale

Follow(E) = $\{ \}$ perché $A \rightarrow (E)$

? $E' \rightarrow + E \mid - E$ richiedono che
 $\text{Follow}(E') \subseteq \text{Follow}(E)$

Follow(E') = poiché $E \rightarrow TE'$, deve esser
 $\text{Follow}(E) \subseteq \text{Follow}(E')$

$$\Rightarrow \begin{array}{l} \text{Follow}(E) \\ = \text{Follow}(E') \end{array}$$

Follow(T) = $E \rightarrow TE' \Rightarrow$ include $\text{First}(E') \setminus \{\epsilon\}$
 + poiché $\epsilon \in \text{First}(E')$
 include anche $\text{Follow}(E)$!

$$T' \rightarrow * T \Rightarrow \text{Follow}(T') \subseteq \text{Follow}(T)$$

$$\text{Follow}(T') = T \rightarrow AT' \Rightarrow \text{Follow}(T) \subseteq \text{Follow}(T') \Rightarrow \text{Follow}(T) = \text{Follow}(T')$$

$$\text{Follow}(A) = T \rightarrow AT' \Rightarrow$$

- include $\text{First}(T') \setminus \{\epsilon\}$
- + poiché $\epsilon \in \text{First}(T')$, \Rightarrow include $\text{Follow}(T)$

Tabella di Parsing LL(1)

strumento per risolvere
il nondeterminismo!

input
left-to-right
un simbolo da
look-ahead
derivation
leftmost

Matrice bidimensionale M

- righe: nonterminal
- colonne: Terminali (più \$)
- casella (A, a) : $M[A, a]$ contiene le produzioni che possono essere scelte dal parser mentre tenta di espandere A e l'input corrente è a .

Se ogni casella contiene al più una produzione,
allora il parser è deterministico!

Come si compone la Tabella?

Per ogni produzione $A \rightarrow \alpha$

- 1) per ogni $a \in T$ e $a \in \text{First}(\alpha)$, inserisci $A \rightarrow \alpha$
nella casella $M[A, a]$
- 2) se $\epsilon \in \text{First}(\alpha)$, inserisci $A \rightarrow \alpha$ in tutte le caselle
 $M[A, x]$ per $x \in \text{Follow}(A)$ (x può essere \$)

Ogni casella vuota, dopo aver elaborato tutte le produzioni, è un errore (cioè la funzione ricorsiva chiama "fail")

(12)

Def Una grammatica è LL(1) se ogni casella della tabella di parsing LL(1) contiene al più una produzione.

Parser "predittivo" deterministico: se $G \in \text{LL}(1)$, allora il parser ricostuisce l'albero di derivazione, per l'input w , in modo top-down, predicendo quale produzione usare (tra le molte possibili) guardando il prossimo carattere dell'input.

Teorema $G \in \text{LL}(1)$ se per ogni coppia di produzioni distinte con la stessa testa

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

si ha che

- 1) $\text{First}(\alpha) \cap \text{First}(\beta) = \emptyset$
- 2) a) se $\epsilon \in \text{First}(\alpha)$, allora $\text{First}(\beta) \cap \text{Follow}(A) = \emptyset$
b) se $\epsilon \in \text{First}(\beta)$, allora $\text{First}(\alpha) \cap \text{Follow}(A) = \emptyset$

Dim Se sono soddisfatte le condizioni 1) e 2) per ogni coppia di produzioni distinte con medesima Testa, allora la tabella di parsing LL(1) contiene al più una produzione in ogni casella.

Ma vale anche il viceversa!

Esempio

(13)

$$\begin{array}{l} S \rightarrow A \mid B \\ A \rightarrow ab \mid cd \\ B \rightarrow ad \mid cb \end{array}$$

G non è LL(1)

$$S \rightarrow A \mid B$$

$$\text{First}(A) = \{a, c\}$$

$$\text{First}(B) = \{a, c\}$$

$$\text{First}(A) \cap \text{First}(B) = \{a, c\}$$

($M[S, a] \in M[S, c]$ contemporaneamente
ma $S \rightarrow A$ che $S \rightarrow B$)

Poniamo però manipolarla per farla diventare LL(1)

Prima espando

$$S \rightarrow ab \mid cd \mid ad \mid cb$$

Poi fattorizzo

$$S \rightarrow aT \mid cT'$$

$$T \rightarrow b \mid d$$

$$T' \rightarrow b \mid d$$

Poi osservo che T e T' sono identiche

$$\begin{array}{l} S \rightarrow aT \mid cT \\ T \rightarrow b \mid d \end{array} \quad G'$$

G' è banalmente LL(1)

Oss: G non è LL(1), ma $L(G) = \{ab, cd, ad, cb\}$

è un ling LL(1) perché esiste una grammatica LL(1) (ovvero G') che lo genera !!

$$E \rightarrow TE'$$

$$E' \rightarrow \epsilon | + E | - E$$

$$T \rightarrow AT'$$

$$T' \rightarrow \epsilon | * T$$

$$A \rightarrow a | b | (E)$$

14
First Follow

E	a, b, (\$,)
E'	$\epsilon, +, -$	\$,)
T	a, b, (\$,), +, -
T'	$\epsilon, *$	\$,), +, -
A	a, b, (\$,), +, -, *

	a	b	()	+	-	*	\$
E	$E \rightarrow TE'$	$E \rightarrow TE'$	$E \rightarrow TE'$					$E' \rightarrow \epsilon$
E'					$E' \rightarrow \epsilon$	$E' \rightarrow + E$	$E' \rightarrow - E$	
T	$T \rightarrow AT'$	$T \rightarrow AT'$	$T \rightarrow AT'$					
T'				$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	$T' \rightarrow * T$	$T' \rightarrow \epsilon$
A	$A \rightarrow a$	$A \rightarrow b$	$A \rightarrow (E)$					

Funzionamento per parser

(uso una pila
perché più
semplice)

input	stack
<u>a</u> * (b+a) \$	<u>E</u>
	<u>TE'</u>
	<u>AT'E'</u>
	<u>aT'E'</u>
<u>*</u> (b+a) \$	<u>T'E'</u>
<u>-</u>	<u>*T'E'</u>
<u>(b+a) \$</u>	<u>TE'</u>
	<u>AT'E'</u>
	<u>(E)T'E'</u>
<u>b+a) \$</u>	<u>E)T'E'</u>
	<u>TE')T'E'</u>
	<u>AT'E')T'E'</u>
	<u>b T'E')T'E'</u>

input	stack
<u>+a) \$</u>	<u>T'E') T'E'</u>
	<u>E')T'E'</u>
	<u>+E) T'E'</u>
	<u>E) T'E'</u>
<u>a) \$</u>	<u>TE')T'E'</u>
	<u>AT'E')T'E'</u>
	<u>aT'E')T'E'</u>
	<u>T'E')T'E'</u>
	<u>E')T'E'</u>
	<u>)T'E'</u>

Esercizio: costruire
l'albero di derivazione!

AVuto la
pila è accettato!

(15)

Parser LL(1) non ricorsivo (usando esplicitamente una pila)

- Pila := $S\$$ (cima della pila a sinistra);
 - $X := S$ (top della pila);
 - input := $w\$$; $i_c :=$ primo carattere dell'input;
 - while ($X \neq \$$) { % finché la pila non è vuota
 - if (X è un terminale) {
 - 1) if ($X = i_c$) {
 - pop X dalla pila; avanza i_c sull'input;
 - }
 - else errore(); % caso no match
 - 2) $X :=$ top della pila }
 - else {
 - 1) if ($M[X, i_c] = X \rightarrow Y_1 \dots Y_n$) {
 - pop X dalla pila;
 - push $Y_1 \dots Y_n$ sulla pila (Y_1 in cima);
 - output la produzione $X \rightarrow Y_1 \dots Y_n$;
 - }
 - else errore(); % caso "bianco"
 - 2) $X :=$ top della pila;
 - }
- }
- if ($i_c \neq \$$) errore(); % caso "ho svuotato la pila ma non ho finito l'input!"

(16)

$$S \rightarrow aAB \mid bS$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$G \quad L(G) = \mathcal{L}[b^*aab]$$

$G \vdash L(L(1))$ per chw

$$\text{First}(aAB) \cap \text{First}(bS) = \emptyset$$

$$\{a\} \cap \{b\} = \emptyset$$

First Follow

S	a, b	\$
A	a	b
B	b	\$

	a	b	\$
S	$S \rightarrow aAB$	$S \rightarrow bS$	
A	$A \rightarrow a$		
B		$B \rightarrow b$	

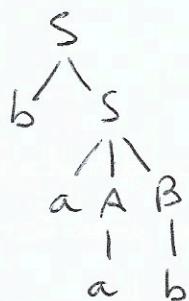
Tabelle di parsif

input	Stack
<u>baab\$</u>	<u>S\$</u>
	<u>bS\$</u>
<u>aab\$</u>	<u>S\$</u>
-	<u>aAB\$</u>
<u>ab\$</u>	<u>AB\$</u>
-	<u>aB\$</u>
<u>b\$</u>	<u>B\$</u>
-	<u>b\$</u>
<u>\$</u>	<u>\$</u>

ok

input	Stack
<u>aabb\$</u>	<u>S\$</u>
-	<u>aAB\$</u>
<u>abb\$</u>	<u>AB\$</u>
-	<u>aB\$</u>
<u>bb\$</u>	<u>B\$</u>
<u>b\$</u>	<u>b\$</u>
<u>\$</u>	<u>\$</u>

errore!
ho svuotato
la pila, ma non ho
finito di leggere l'input



<u>abb\$</u>	<u>S\$</u>
-	<u>aAB\$</u>
<u>bb\$</u>	<u>AB\$</u>

errore!
casella bianca!

(17)

$$S \rightarrow aAB \mid B \\ A \rightarrow a \\ B \rightarrow bB \mid b$$

$$L(G) = (aa|\epsilon)b^+$$

G non è LL(1) per $B \rightarrow bB \mid b$

\Downarrow
follow primo

$$S \rightarrow aAB \mid B \\ A \rightarrow a \\ B \rightarrow bB' \\ B' \rightarrow B \mid \epsilon$$

G' è LL(1) perché

- $\text{First}(aAB) \cap \text{First}(B) = \emptyset$
 $\{a\} \cap \{b\} = \emptyset$
- $\text{First}(B) \cap \text{First}(\epsilon) = \emptyset$
- $\text{First}(B) \cap \text{Follow}(B') = \emptyset$

First Follow

	First	Follow
S	a, b	\$
A	a	b
B	b	\$
B'	ϵ, b	\$

	a	b	\$
S	$S \rightarrow aAB$	$S \rightarrow B$	
A	$A \rightarrow a$		
B		$B \rightarrow bB'$	
B'		$B' \rightarrow B$	$B' \rightarrow \epsilon$

input	stack
<u>a</u> a bb \$	S \$
<u>a</u>	aAB \$
<u>bb</u> \$	A B \$
<u>a</u>	a B \$
<u>bb</u> \$	B \$
<u>b</u> \$	bB' \$
	B' \$
	B \$
	bB' \$
	B' \$
\$	\$

OK

input	stack
aa \$	S \$
-	aAB \$
a \$	AB \$
-	aB \$
\$	B \$

errore!
(casella bianca)

Ho finito l'input, ma
la pila non è vuota!

$M[B, \$] = \text{"bianco"}$

$$\begin{array}{l} S \rightarrow aAB \\ A \rightarrow C|D \\ B \rightarrow b \\ C \rightarrow c|\epsilon \\ D \rightarrow d \end{array} \quad G$$

(18)

$$L(G) = \{ab, acb, adb\}$$

	First	Follow
S	a	\$
A	c, d, ε	b
B	b	\$
C	c, ε	b
D	d	b

	a	b	c	d	\$
S	$S \rightarrow aAB$				
A		$A \rightarrow C$	$A \rightarrow C$	$A \rightarrow D$	
B		$B \rightarrow b$			
C		$C \rightarrow \epsilon$	$C \rightarrow c$		
D				$D \rightarrow d$	

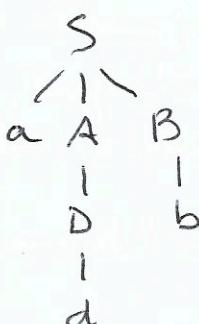
perché $\epsilon \in \text{First}(\epsilon)$
 e $b \in \text{Follow}(C)$

perché $\epsilon \in \text{First}(C)$
 e $b \in \text{Follow}(A)$

Output	Input	Stack
$S \rightarrow aAB$	<u>a</u> db\$	S\$
		<u>a</u> AB\$
	<u>d</u> b\$	AB\$
		DB\$
	<u>b</u> \$	<u>d</u> B\$
		B\$
	<u>b</u> \$	<u>b</u> \$
		\$
		ok

Input	Stack
a bb\$	S\$
	<u>a</u> AB\$
	AB\$
	CB\$
	B\$
	<u>b</u> \$
	\$
	error!

Ho svuotato la pila
 ma non ho finito di
 leggere l'input



Teorema Ogni linguaggio regolare è generabile⁽¹⁹⁾ da una grammatica G di classe LL(1).

Dim Se L è regolare, allora $\exists \text{ DFA } M = (Q, \Sigma, \delta, q_0, F)$ tale che $L = L[M]$.

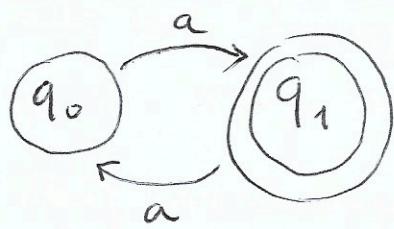
A partire da M , costruiamo la grammatica regolare $G = (NT, T, S, R)$ con $NT = \{[q] \mid q \in Q\}$, $T = \Sigma$, $S = [q_0]$ e R definita da

- se $\delta(q, a) = q'$, allora $[q] \xrightarrow{a} [q'] \in R$
 - se $q \in F$, allora $[q] \xrightarrow{\epsilon} \in R$
- (seconda tecnica
per trasformare un
DFA in gr. regolare)

$\Rightarrow G$ è LL(1)!

Inoltre, poiché M è deterministico, da ogni $q \in Q$ per ogni $a \in \Sigma \exists! q' \mid q \xrightarrow{a} q'$, cioè $[q]$ avrà una sola produzione $[q] \xrightarrow{a} [q']$ che inizia per "a".

Inoltre, se q è finale, allora $[q] \xrightarrow{\epsilon}$ è applicabile solo per i $\text{Follow}([q]) = \{ \# \}$! \Rightarrow nessun conflitto nel riempimento della tabella di parsing, poiché nessuna produzione genera $\#$.



$$[q_0] \xrightarrow{a} [q_1]$$

$$[q_1] \xrightarrow{a} [q_0] \mid \epsilon$$

è LL(1) poiché

- $\text{First}(a[q_0]) \cap \text{First}(\epsilon) = \emptyset$
- $\text{First}(a[q_0]) \cap \text{Follow}(q_1) = \emptyset$
 $\{a\} \cap \{\#\} = \emptyset$

Grammatiche LL(k)

(20)

First_k(α)

$w \in \text{First}_k(\alpha)$ se $\alpha \Rightarrow^* w\beta$ con $|w|=k$, $w \in T^*$, $\beta \in (T \cup NT)^*$
oppure $\alpha \Rightarrow^* w$ con $|w| \leq k$, $w \in T^*$

Follow_k(A)

$w \in \text{Follow}_k(A)$ se $S \Rightarrow^* \alpha Aw\beta$ con $|w|=k$ e $w \in T^*$
 $\alpha, \beta \in (T \cup NT)^*$
oppure $S \Rightarrow^* \alpha Aw$ con $|w| \leq k$ e $w \in T^*$

Tabella di Parsing LL(k)

- righe: nonterminali
- colonne: $\{w \in T^* \mid |w| \leq k\}$ - (solo quelli necessari)
- Per ogni produzione $A \rightarrow \alpha$, $M[A, w]$ contiene $A \rightarrow \alpha$, per ogni $w \in \text{First}_k(\alpha)$ ($w \neq \epsilon$) e per ogni $w \in \text{Follow}_k(A)$ se $\epsilon \in \text{First}_k(\alpha)$
- Se ogni entrata/casella contiene al più una produzione, e non esistono $w_1 \neq w_2$ tale che w_1 prefigura w_2 con le due entrate corrispondenti in una riga entrambe riempite, allora G è LL(k)

Vedi esempio in 21bis

N.B. Le colonne sono tante quanti i w che appartengono a $\text{First}_k(\alpha)$ per $A \rightarrow \alpha$ o a $\text{Follow}_k(A)$ per $A \rightarrow \alpha$. E questo va fatto per tutte le produzioni.

(se $\epsilon \in \text{First}_k(\alpha)$)

Esempio

$$S \rightarrow aSb \mid ab \mid c \quad G \quad L(G) = \{a^n b^n \mid n \geq 1\} \\ \cup \{a^n c b^n \mid n \geq 0\}$$

- $\text{First}_2(aSb) = \{aa, ac\}$

- $\text{First}_2(ab) = \{ab\}$

- $\text{First}_2(c) = \{c\}$

$G \in LL(2)$ perché

- $\text{First}_2(aSb) \cap \text{First}_2(ab) = \emptyset$

- $\text{First}_2(aSb) \cap \text{First}_2(c) = \emptyset$

- $\text{First}_2(ab) \cap \text{First}_2(c) = \emptyset$

	aa	ab	ac	c
S	$S \rightarrow aSb$	$S \rightarrow ab$	$S \rightarrow aSb$	$S \rightarrow c$

Teorema

- 1) Una grammatica wcorsiva sinistra non è $LL(k)$ per nessun k .
- 2) Una grammatica ambigua non è $LL(k)$ per nessun k
- 3) Se $G \in LL(k)$ per qualche k , allora G non è ambigua !!
- 4) Se $G \in LL(k)$, allora $L(G)$ è libero deterministico !
- 5) Esiste L libero deterministico tale che non esiste G di classe $LL(k)$ - per nessun k - tale che $L = L(G)$.
↑
lo vedremo!
(pagina 23)

Esempio

21 bis

$$G_1 \left[\begin{array}{l} S \rightarrow aSb \mid aAc \\ A \rightarrow cA \mid c \end{array} \right]$$

$$\text{First}_2(aSb) = \{aa\} \quad \text{First}_2(aAc) = \{ac\}$$

$$\text{First}_2(cA) = \{cc\} \quad \text{First}_2(c) = \{c\}$$

	aa	ac	cc	c	#
S	$S \rightarrow aSb$	$S \rightarrow aAc$			
A			$A \rightarrow cA$	$A \rightarrow c$	

Tabella di parsing LL(2) senza conflitti,
ma "c" è prefisso di "cc" e quindi il parsing
non è deterministico quando A è in cima alla pila

<u>aaccb\$</u>	<u>s</u>				
	<u>aSb</u>				
		<u>s b</u>			
			<u>aAc b</u>		
				<u>cc b</u>	
					<u>Ac b</u>
					<u>cc</u>
					<u>c</u>
					questa è quella che mi servirebbe
					qui ho due mosse possibili
					G ₁ non è LL(2)

Esercizio: Verificare che

$$S \rightarrow aSb \mid aAc \quad] G_2 \text{ è } LL(2)$$

$$A \rightarrow cA \mid d$$

$$\text{First}_2(aSb) = \{aa\} \quad \text{First}_2(aAc) = \{ac, ad\}$$

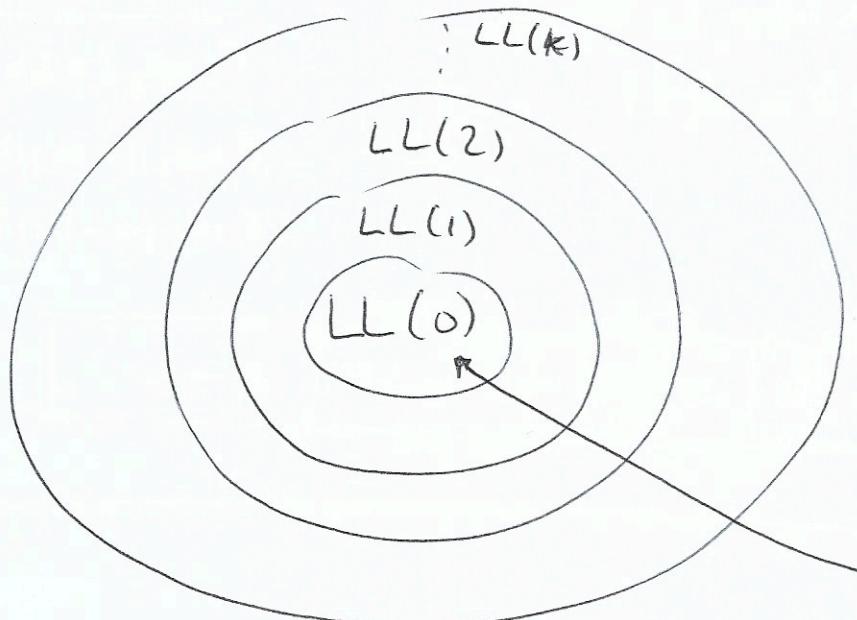
$$\text{First}_2(cA) = \{cc, cd\} \quad \text{First}_2(d) = \{d\}$$

	aa	ac	ad	cc	cd	d	\$
S	$S \rightarrow aSb$	$S \rightarrow aAc$	$S \rightarrow aAc$		$A \rightarrow cA$	$A \rightarrow cA$	$A \rightarrow d$
A							

Tabella LL(2) senza conflitti e \neq w1w2 tab che w1
è prefisso di w2 $\Rightarrow G_2 \text{ è } LL(2)$

Def Un linguaggio L è di classe $LL(k)$ se esiste G di classe $LL(k)$ tale che $L = L(G)$.

Prop. Per ogni $k \geq 0$, la classe dei linguaggi $LL(k+1)$ contiene strettamente la classe dei linguaggi $LL(k)$.



In pratica si usa solo $LL(1)$!

$G \in LL(0)$ se ogni A ∈ NT ha una sola produzione
 $\Rightarrow L(G) = \{w\}$
 (una sola parola, al massimo)

Se G non è $LL(1)$, spesso la si può manipolare (fattorizzare, eliminare ricorsione sx, disambiguare, ecc...) trasformandola in una equivalente $LL(1)$.

$S \rightarrow aSb \mid ab \mid c \quad] G \in LL(2) \text{ ma non } LL(1)$

$S \rightarrow aT \mid c \quad] G'$ ottenuta fattorizzando G , è $LL(1)$!
 $T \rightarrow Sb \mid b \quad]$

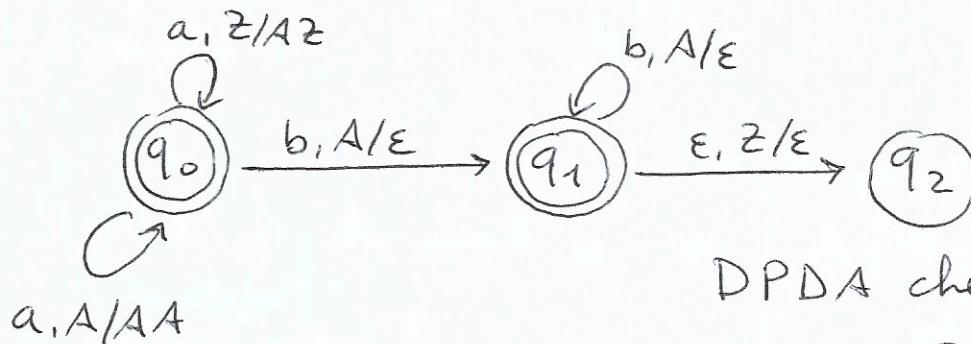
- $\text{First}(aT) \cap \text{First}(c) = \emptyset$
- $\text{First}(Sb) \cap \text{First}(b) = \emptyset$
- $\{a, c\} \cap \{b\} = \emptyset$

$L = \{a^n b^n \mid n \geq 1\} \cup \{a^n c b^n \mid n \geq 0\}$ è un lang. di classe $LL(1)$ perché G' è $LL(1)$!

$$L = \{ a^i b^j \mid i \geq j \} \text{ è libero det.}$$

(23)

ma non è LL(k) per nessun k!



DPDA che riconosce L
per stato finale

Possibile grammatica libera per L

$$\begin{aligned} S &\rightarrow aS \mid B \\ B &\rightarrow aBb \mid \epsilon \end{aligned} \quad G \quad L = L(G)$$

- Come faccio a scegliere tra $S \rightarrow aS$ e $S \rightarrow B$?
Dovrei leggere fino in fondo l'input per sapere quante b in meno di a ci sono nella stringa!
 \Rightarrow G non può essere LL(k) per nessun k
(per quanto sia grande k, posso trovare una stringa più lunga che richiede di leggere più di k simboli di look-ahead)
- Non è possibile trovare G' e k tali che $L(G') = L$ e $G' \in LL(k)$
(dimostrazione difficile)

Esercizio

(24)

$L = \{a^i b^j \mid i \geq j \geq 0\}$ è un lang. di classe LL(1).

$S \rightarrow S_1 B$
 $S_1 \rightarrow a S_1 b \mid \epsilon$
 $B \rightarrow b B \mid \epsilon$

G è tale che $L(G) = L$

First Follow

S	a, b, ϵ	\$
S_1	a, ϵ	$b, \$$
B	b, ϵ	\$

	a	b	\$
S	$S \rightarrow S_1 B$	$S \rightarrow S_1 B$	$S \rightarrow S_1 B$
S_1	$S_1 \rightarrow a S_1 b$	$S_1 \rightarrow \epsilon$	$S_1 \rightarrow \epsilon$
B		$B \rightarrow b B$	$B \rightarrow \epsilon$