PARSER TOP-DOWN

Presentiamo un primo esempio di parser top-down, nondeterministico, 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

$$A \to X_1^1 \dots X_{n1}^1 \mid \dots \mid X_1^k \dots X_{nk}^k$$

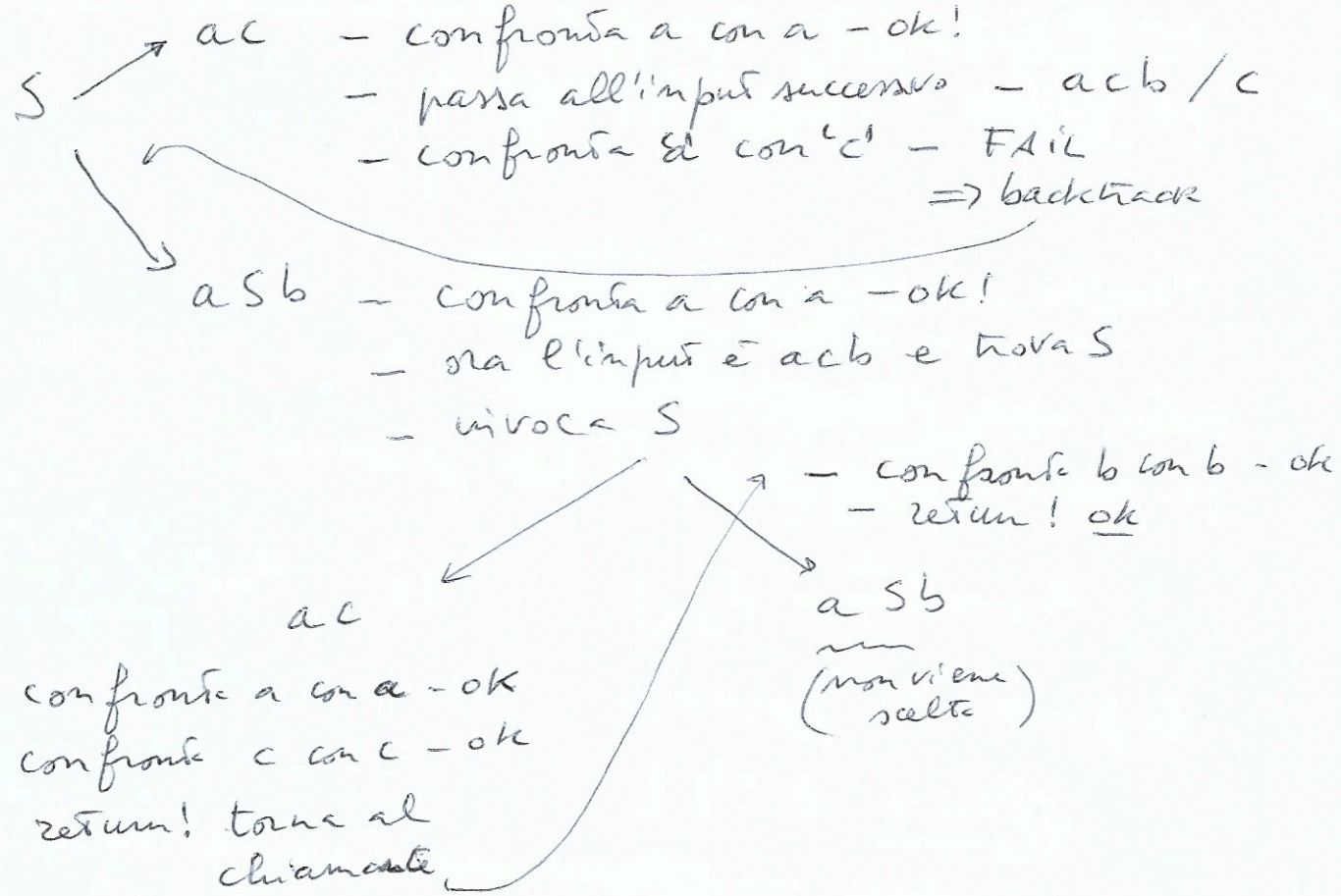
definisco la funzione

```
function A() {
  - scelgo nondeterministicamente h tra 1 e k,
    ovvero una produzione A → X_1^h ... X_nh^h ;
  - for i=1 to n_h {
        if X_i^h ∈ NT then X_i^h();
        else if X_i^h = simbolo corrente
                        dell'input
             then avanza di un simbolo
                        sull'input
             else Fail();
    }                  }
                       backtracking!
                       si torna al punto
                       e si sceglie
                       un'altra produzione
  return;
}
```

Esempio $S \to ac \mid aSb$ $L = \{ a^{n+1} c b^n \mid n \geq 0 \}$

input aacb

Si invoca la funzione del simbolo iniziale S



Se lo vediamo come una pila...

input	Stack delle chiamate
a a c b	S
a c b	a C
	c fail

---

a a c b	S
	a S b
a c b	S b
	a c b
c b	c b
	b
b	
	ok

Parser a discesa ricorsiva è molto inefficiente:

- Nondeterminismo:

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

$\Rightarrow$ esponenziale nella lunghezza della stringa w, dove la base b è data dal massimo numero di produzioni per uno stesso nonterminale

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

$\Rightarrow$ Cerchiamo di guidare la scelta della produzione

Come?

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

---

E1: 1) $A \to aB \mid bC$

$\Rightarrow$ se il prossimo input è a, uso $A \to aB$

$\Rightarrow$ se il prossimo input è b, uso $A \to bC$

2) $S \to ac \mid asb$

$\Rightarrow$ se i prossimi 2 input sono ac, uso $S \to ac$

$\Rightarrow$ se i prossimi 2 input sono aa, uso $S \to asb$

$\Rightarrow$ introduciamo 2 funzioni ausiliarie:

- First
- Follow

Attenzione: 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!

Perché? Un parser top-down è essenzialmente un DPDA che riconosce per pila vuota. Allora è necessario che L goda della prefix property, e L·# sicuramente gode di questa proprietà.

FIRST

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

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

Es: $A \to \alpha_1 \mid \alpha_2$

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

$A \to aB \mid bC$

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

Ma il First può non bastare!

Es: $S \to Ab \mid c$
$A \to aA \mid \varepsilon$

$First(Ab) = \{a, b\}$ (ε perché $A \to \varepsilon$)
$First(c) = \{c\}$

- $First(Ab) \cap First(c) = \emptyset$

$First(aA) = \{a\}$
$First(\varepsilon) = \{\varepsilon\}$

- $First(aA) \cap First(\varepsilon) = \emptyset$

Però, se l'input è ab

$S \xrightarrow{a} Ab$

simbolo in lettura sull'input

$A \xrightarrow{a} aA$ = consumo a

$A \xrightarrow{b} ?$

$b \notin First(aA)$
$b \notin First(\varepsilon)$

$\Rightarrow$ quale produzione applicare?

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

$\Rightarrow$ devo scegliere $A \to \varepsilon$!

Come pila...

input	Stack ←
a b $	S
	Ab
	aAb
b $	Ab
	b
$	ε

Follow

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

- $\forall a \in T$, $a \in Follow(A)$ se $S \Rightarrow^* \alpha A a \beta$ per qualche $\alpha, \beta \in (T \cup NT)^*$
- $\$ \in Follow(A)$ se $S \Rightarrow^* \alpha A$

(Poiché $S \Rightarrow^* S$, allora $\$ \in Follow(S)$!)

Esempio

$S \to Ab \mid c$

$A \to aA \mid \varepsilon$

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

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

$S \Rightarrow^* S$

$S \Rightarrow^* Ab$

Come calcolare i First?

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

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

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

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

In pratica, se $A \to \alpha_1 | \dots | \alpha_k$, allora

$First(A) = First(\alpha_1) \cup \dots \cup First(\alpha_k)$

Esempio

$S \to Ab \mid c$

$A \to a\,A \mid \varepsilon$

$First(S) = First(Ab) \cup First(c)$

$= (First(A) \setminus \{\varepsilon\}) \cup First(b) \cup \{c\}$

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

$First(A) = First(a\,A) \cup First(\varepsilon)$

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

$S \to ABC \mid cB$

$A \to a \mid \varepsilon$

$B \to b \mid \varepsilon$

$C \to c$

$First(A) = \{a, \varepsilon\}$

$First(B) = \{b, \varepsilon\}$

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

$First(S) = First(ABC) \cup First(CB)$

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

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

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

Procedura per calcolare Follow(X) (con $X \in NT$) (9

- Per ogni $X \in NT$, inizializza $Follow(X) := \emptyset$
- $Follow(S) := \{\$\}$
- Ripeti il seguente ciclo finché nessun $Follow(X)$ viene più modificato in una iterazione:

  1) Per ogni produzione $X \to \alpha Y \beta$

  $$Follow(Y) := Follow(Y) \cup (First(\beta) \setminus \{\varepsilon\})$$

  2) Per ogni produzione $X \to \alpha Y$ e per ogni produzione $X \to \alpha Y \beta$ con $\varepsilon \in First(\beta)$

  $$Follow(Y) := Follow(Y) \cup Follow(X)$$

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

$S \to ABC$

$A \to aaA \mid \varepsilon$

$B \to b \mid \varepsilon$

$C \to cC \mid \varepsilon$

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

Poiché $S \to ABC$, il $Follow(B) \supseteq First(C) \setminus \{\varepsilon\}$
ed anche $Follow(B) \supseteq Follow(S)$
perché $\varepsilon \in First(C)$!

$E \to TE'$

$E' \to \varepsilon \mid +E \mid -E$

$T \to AT'$

$T' \to \varepsilon \mid *T$

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

	First	Follow
E	a, b, (	\$,)
E'	ε, +, −	\$,)
T	a, b, (	\$,), +, −
T'	ε, *	\$,), +, −
A	a, b, (	\$,), +, −, *

Follow(E) = \$ perché E è il simbolo iniziale
) perché $A \to (E)$
? $E' \to +E \mid -E$ richiedono che Follow(E') ⊆ Follow(E)

Follow(E') = poiché $E \to TE'$, deve essere Follow(E) ⊆ Follow(E') ⇒ Follow(E) = Follow(E')

Follow(T) = $E \to TE'$ ⇒ • include First(E') \ {ε}
+ • poiché ε ∈ First(E') include anche Follow(E)!
$T' \to *T$ ⇒ Follow(T') ⊆ Follow(T)

Follow(T') = $T \to AT'$ ⇒ Follow(T) ⊆ Follow(T') ⇒ Follow(T) = Follow(T')

Follow(A) = $T \to AT'$ ⇒ • include First(T') \ {ε}
+ • poiché ε ∈ First(T'), ⇒ include Follow(T)

Tabella di Parsing LL(1)

- strumento per risolvere il nondeterminismo!
- input left-to-right
- un simbolo di look-ahead
- derivazione leftmost

Matrice bidimensionale M

- righe: nonterminali
- 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 riempie la tabella?

Per ogni produzione $A \to \alpha$

1) per ogni $a \in T$ e $a \in First(\alpha)$, inserisco $A \to \alpha$ nella casella $M[A, a]$
2) se $\varepsilon \in First(\alpha)$, inserisco $A \to \alpha$ in tutte le caselle $M[A, x]$ per $x \in Follow(A)$ (x può essere $)

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

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

Parser "predittivo" deterministico: se G è LL(1), allora il parser ricostruisce 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 è LL(1) sse per ogni coppia di produzioni distinte con la stessa testa

$$A \to \alpha \mid \beta$$

si ha che

1) $First(\alpha) \cap First(\beta) = \emptyset$

2) a) se $\varepsilon \in First(\alpha)$, allora $First(\beta) \cap Follow(A) = \emptyset$

   b) se $\varepsilon \in First(\beta)$, allora $First(\alpha) \cap Follow(A) = \emptyset$

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

$$\left.\begin{array}{l} S \to A \mid B \\ A \to ab \mid cd \\ B \to ad \mid cb \end{array}\right] \quad G \text{ non è LL(1)}$$

$S \to A \mid B$

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

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

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

($M[S, a]$ e $M[S, b]$ contengono sia $S \to A$ che $S \to B$)

Possiamo però manipolarla per farla diventare LL(1)

Prima espando

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

Poi fattorizzo

$$S \to aT \mid cT'$$
$$T \to b \mid d$$
$$T' \to b \mid d$$

Poi osservo che T e T' sono identici

$$\left.\begin{array}{l} S \to aT \mid cT \\ T \to b \mid d \end{array}\right] G'$$

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 !!

$E \to TE'$

$E' \to \varepsilon \mid +E \mid -E$

$T \to AT'$

$T' \to \varepsilon \mid *T$

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

	First	Follow
E	a, b, (	$,)
E'	ε, +, -	$,)
T	a, b, (	$,), +, -
T'	ε, *	$,), +, -
A	a, b, (	$,), +, -, *

	a	b	(	)	+	-	*	$
E	$E \to TE'$	$E \to TE'$	$E \to TE'$					
E'				$E' \to \varepsilon$	$E' \to +E$	$E' \to -E$		$E' \to \varepsilon$
T	$T \to AT'$	$T \to AT'$	$T \to AT'$					
T'				$T' \to \varepsilon$	$T' \to \varepsilon$	$T' \to \varepsilon$	$T' \to *T$	$T' \to \varepsilon$
A	$A \to a$	$A \to b$	$A \to (E)$					

Funzionamento per parser (uso una pila perché più semplice)

input →	stack ←
a*(b+a)$	E
	TE'
	AT'E'
a*(b+a)$	aT'E'
*(b+a)$	T'E'
	*TE'
(b+a)$	TE'
	AT'E'
	(E)T'E'
b+a)$	E)T'E'
	TE')T'E'
	AT'E')T'E'
	bT'E')T'E'
+a)$	T'E')T'E'
	E')T'E'
	+E)T'E'
a)$	E)T'E'
	TE')T'E'
	AT'E')T'E'
	aT'E')T'E'
)$	T'E')T'E'
	E')T'E'
	)T'E'
$	T'E'
	E'
	ε

Esercizio: costruire l'albero di derivazione!

svuoto la pila e accetto!

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 ≠ $) { % finché la pila non è vuota

  if (X è un terminale) {

  1) if (X = i_c) {
     pop X dalla pila; avanta i_c sull'input;
     }
     else errore(); % caso no match
     ↖ se G è LL(1) questo caso non si verifica
  2) X := top della pila }

  else {

  1) if (M[X, i_c] = $X \to Y_1 \dots Y_n$) {
     pop X dalla pila;
     push $Y_1 \dots Y_n$ sulla pila (Y_1 in cima);
     output la produzione $X \to Y_1 \dots Y_n$;
     }
     else errore(); % caso "bianco"
  2) X := top della pila;
  }
  }
- if (i_c ≠ $) errore(); % caso "ho svuotato la pila ma non ho finito l'input!"

$S \to aAB \mid bS$
$A \to a$
$B \to b$

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

G è LL(1) perché

$First(aAB) \cap First(bS) = \emptyset$

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

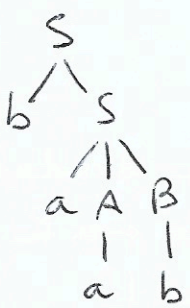
	First	Follow
S	a, b	$
A	a	b
B	b	$

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

tabella di parsing

input	stack
baab$	S$
	bS$
aab$	S$
	aAB$
ab$	AB$
	aB$
b$	B$
	b$
$	$

ok



input	stack
aabb$	S$
	aAB$
abb$	AB$
	aB$
bb$	B$
	b$
b$	$

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

---

abb$	S$
	aAB$
bb$	AB$

errore! casella bianca!

$S \to aAB \mid B$
$A \to a$
$B \to bB \mid b$
] G

$L(G) = (aa \mid \varepsilon)\, b^+$

G non è LL(1) per $B \to bB \mid b$

⇓ fattorizziamo

$S \to aAB \mid B$
$A \to a$
$B \to bB'$
$B' \to B \mid \varepsilon$
] G'

G' è LL(1) perché

- $First(aAB) \wedge First(B) = \emptyset$
  $\{a\} \wedge \{b\} = \emptyset$
- $First(B) \wedge First(\varepsilon) = \emptyset$
- $First(B) \wedge Follow(B') = \emptyset$

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

	a	b	$
S	$S \to aAB$	$S \to B$	
A	$A \to a$		
B		$B \to bB'$	
B'		$B' \to B$	$B' \to \varepsilon$

input	stack
aabb$	S$
	aAB$
abb$	AB$
	aB$
bb$	B$
	bB'$
b$	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, $] = "bianco"

$S \to aAB$
$A \to C \mid D$
$B \to b$
$C \to c \mid \varepsilon$
$D \to d$

G

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

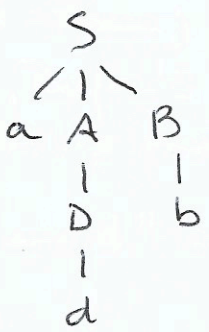
	First	Follow
S	a	$
A	c, d, ε	b
B	b	$
C	c, ε	b
D	d	b

	a	b	c	d	$
S	S → aAB				
A		A → C	A → C	A → D	
B		B → b			
C		C → ε	C → c		
D				D → d	

perché ε ∈ First(ε) e b ∈ Follow(C)

perché ε ∈ First(C) e b ∈ Follow(A)

Output	Input	Stack
	adb$	S$
S → aAB		aAB$
	db$	AB$
A → D		DB$
D → d		dB$
	b$	B$
B → b		b$
	$	$
		ok



Input	Stack
abb$	S$
	aAB$
bb$	AB$
	CB$
	B$
	b$
b$	$
	errore!

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

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

Dim Se L è regolare, allora $\exists$ 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

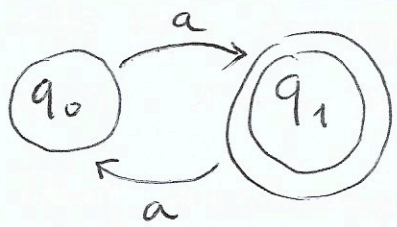
- se $\delta(q,a) = q'$, allora $[q] \to a\,[q'] \in R$
- se $q \in F$, allora $[q] \to \varepsilon \in R$

(seconda tecnica per trasformare un DFA in gr. regolare)

$\Rightarrow$ G è LL(1)!

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

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



$[q_0] \to a\,[q_1]$

$[q_1] \to a\,[q_0] \mid \varepsilon$

è LL(1) perché

- $First(a\,[q_0]) \cap First(\varepsilon) = \emptyset$
- $First(a\,[q_0]) \cap Follow([q_1]) = \emptyset$

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

Grammatiche LL(k)

$First_k(\alpha)$

$w \in First_k(\alpha)$ sse $\alpha \Rightarrow^* w\beta$ con $|w| = k$, $w \in T^*$, $\beta \in (T \cup NT)^*$

oppure $\alpha \Rightarrow^* w$ con $|w| \leq k$, $w \in T^*$

$Follow_k(A)$

$w \in Follow_k(A)$ sse $S \Rightarrow^* \alpha A w \beta$ con $|w| = k$ e $w \in T^*$
$\alpha, \beta \in (T \cup NT)^*$

oppure $S \Rightarrow^* \alpha A w$ 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 $A \to \alpha$, $M[A, w]$ contiene $A \to \alpha$, per ogni $w \in First_k(\alpha)$ $(w \neq \varepsilon)$ e per ogni $w \in Follow_k(A)$ se $\varepsilon \in First_k(\alpha)$
- Se ogni entrata/casella contiene al più una produzione, allora G è LL(k)!

N.B. Le colonne sono tante quanti i w che appartengono a $First_k(\alpha)$ per $A \to \alpha$ o a $Follow_k(A)$ per $A \to \alpha$. E questo va fatto per tutte le produzioni.

(se $\varepsilon \in First_k(\alpha)$)

Esempio

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

- $First_2(aSb) = \{aa, ac\}$
- $First_2(ab) = \{ab\}$
- $First_2(c) = \{c\}$

G è LL(2) perché

- $First_2(aSb) \cap First_2(ab) = \emptyset$
- $First_2(aSb) \cap First_2(c) = \emptyset$
- $First_2(ab) \cap First_2(c) = \emptyset$

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

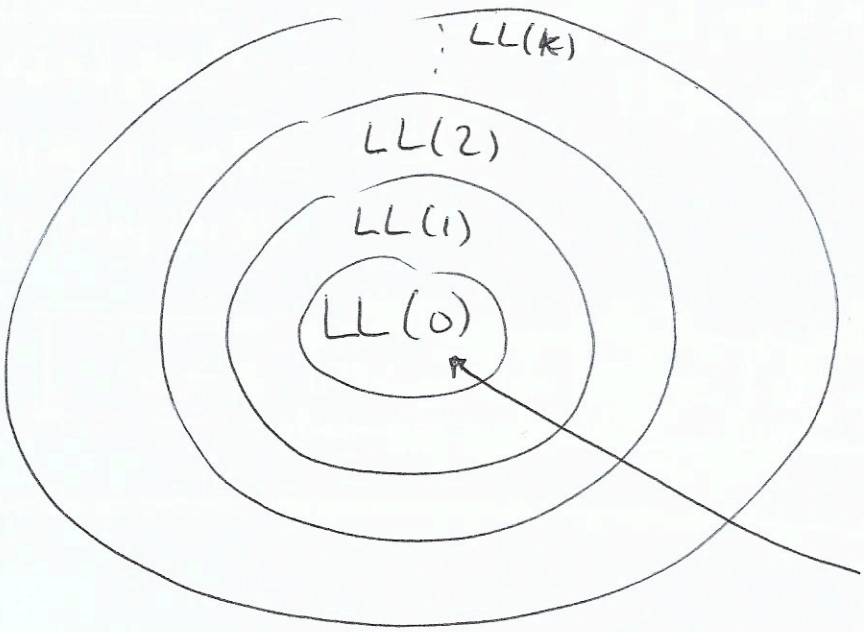
Teorema

1) Una grammatica ricorsiva sinistra non è LL(k) per nessun k.
2) Una grammatica ambigua non è LL(k) per nessun k
3) Se G è LL(k) per qualche k, allora G non è ambigua!!
4) Se G è 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)

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 $LL(k)$.



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

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

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

$S \to aSb \mid ab \mid c$] G è $LL(2)$ ma non $LL(1)$

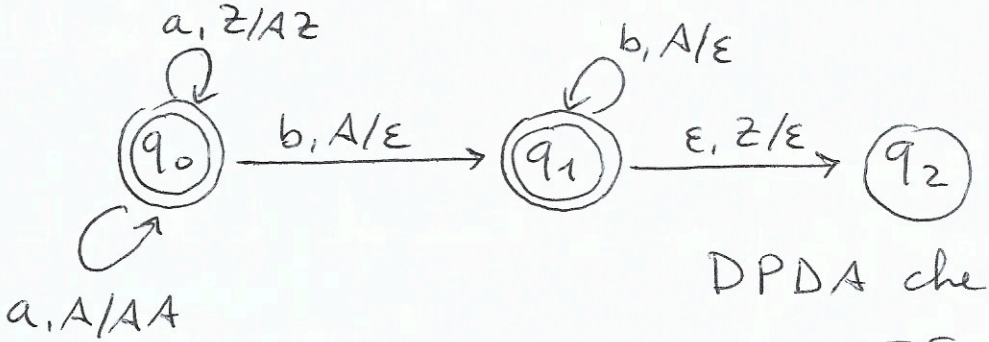
$S \to aT \mid c$
$T \to Sb \mid b$] G' ottenuta fattorizzando G, è $LL(1)$!

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

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

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

ma non è LL(k) per nessun k!



DPDA che riconosce L per stato finale

Possibile grammatica libera per L

$$\left.\begin{array}{l} S \to aS \mid B \\ B \to aBb \mid \varepsilon \end{array}\right] G \qquad L = L(G)$$

- Come faccio a scegliere tra $S \to aS$ e $S \to B$? 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)

Esempio

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

$$\left.\begin{array}{l} S \to S_1 B \\ S_1 \to a S_1 b \mid \varepsilon \\ B \to b B \mid \varepsilon \end{array}\right] \quad G \text{ è tale che } L(G) = L$$

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

	a	b	\$
S	$S \to S_1 B$	$S \to S_1 B$	$S \to S_1 B$
S_1	$S_1 \to a S_1 b$	$S_1 \to \varepsilon$	$S_1 \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$