Esercizi su grammatiche e linguaggi

Alberto Marchetti Spaccamela

Esercizi su linguaggi

- Derivare un'espressione regolare che generi l'insieme di tutte le sequenze di 0 ed 1 che contengono un numero di 1 divisibile per 3
  - se assumo anche 0 come multiplo di 3 allora l'espressione con soli 1 in un numero multiplo di 3 è data da **(111)***
  - Devo poter avere degli zero prima e dopo ciascun 1: quindi la risposta è (0*10*10*10*)*

  Nota : (0*10*10*1)* non e' corretto perché? Fornire un esempio

Esercizi su linguaggi

- Data la seguente grammatica tipo 1

  S→A   A→AAA   A→a

  - Fornire- se esistono - una derivazione per aaaa e una per aaaaa

necessariamente all'inizio devo usare S→A e A→AAA e ottengo quindi S→ A→AAA a questo punto se uso solo la terza produzione (A→a) ottengo aaa; quindi uso nuovamente la seconda produzione (A→AAA) e ottengo S→ A→AAA → AAAAA Quindi

- Da AAAAA posso ottenere aaaaa
- non posso ottenere in nessun modo aaaa (le produzioni permettono di ottenere stringhe di 3 o 5 caratteri e non riducono mai la lunghezza della stringa)

Esercizi su linguaggi

- Data la grammatica tipo 1

  $S \rightarrow A \quad A \rightarrow AAA \quad A \rightarrow a$

  - Descrivere il linguaggio generato dalla grammatica
- la grammatica permette di generare la stringa a ($S \rightarrow A \rightarrow a$) e la stringa aaa: $S \rightarrow AAA \rightarrow aAA \rightarrow aaA \rightarrow aaa$
- Per ogni A la produzione $S \rightarrow AAA$ aumenta il numero di A di due unità
- Quindi ad ogni successiva applicazione di $A \rightarrow AAA$ posso ottenere le stringhe con un numero dispari di A
- Quindi il linguaggio generato è quello con un numero dispari di a

Esercizi su linguaggi

- Si consideri la seguente grammatica tipo 1

  S→A   A→AAA A→a

  – Esiste una grammatica regolare (tipo 3) che genera lo stesso linguaggio?
- Le produzioni S→A   A→AAA non rispettano la definizione di gramamatica tipo 3
- Abbiamo visto che la grammatica genera il linguaggio con un numero dispari di a
- Le produzioni S→SAA e S→A generano le stringhe con un numero dispari di A
- Quindi la risposta è S→SAA   S→A   A→a

Esercizi su linguaggi regolari

- Scrivere la grammatica di tipo 3 che genera il linguaggio descritto dall'espressione regolare
  b^+ (Soluz. $S \rightarrow b \mid bS$) **b^*** (Soluz. $S \rightarrow \varepsilon \mid bS$)

- Si consideri la seguente espressioni regolare
  a(a+b)*b
  - Dare una grammatica (di qualunque tipo,) che genera lo stesso linguaggio
  - Dare una grammatica regolare che genera lo stesso linguaggio
- Ripetere l'esercizio per le espressioni
  - (a+b)(aa+bb)(a+b)
  - ((aa) +(ab))*

Esercizi su linguaggi regolari

Si consideri la seguente espressioni regolare a(a+b)*b

- Dare una grammatica regolare che genera lo stesso linguaggio
- Sol. Le stringhe del linguaggio iniziano con una a
- Quindi iniziamo con $S \rightarrow aA$
- Dopo a le stringhe continuano con (a+b)*b
- Quindi la produzione $A \rightarrow TA \mid b$ permette di ottenere stringhe del tipo $S \rightarrow \ldots \rightarrow aT^{*}b$; concludiamo aggiungendo $T \rightarrow a|b$
- NOTA: La produzione $A \rightarrow TA \mid b$ non è di tipo 3; per tipo 3 abbiamo $A \rightarrow aA \mid bA \mid b$

Domande

Dimostrare che un linguaggio finito è regolare

- Sol. Il linguaggio che ha una sola stringa (di lunghezza finita) è regolare: è facile trovare un automa che decide (un'espressione regolare che genera) la stringa
- Quindi è anche facile combinare gli automi (o le espressioni regolari) che decidono (generano) o un'espressione;
- automa uso nondeterminismo; espressioni regolari uso simbolo + (unione)

Dimostrare che il linguaggio delle stringhe di 0 e 1 di lunghezza esattamente 1000 è regolare : analogo

Domande

Dimostrare che un linguaggio finito è regolare

Esempio: L= {010, 00, 1}

Nel seguito l'espressioni regolare che definisce il linguaggio

- Consideriamo il Linguaggio L1={010}
- Ad esso corrisponde l'espr. regolare e1= 010 (uso concatenazione)
- Quindi ottengo e1= 010   e2=00   e3=1
- Pertanto L corrisponde all'espressione regolare

  (010 + 00 + 1)

Dimostrare che il linguaggio delle stringhe di 0 e 1 di lunghezza 1000 è regolare : analogo

Esercizi

- Sia *rev* un operatore che inverte le sequenze di simboli (ES. *rev(abc)= cba*). Sia L un linguaggio generato da un'espressione regolare; dimostrare che esiste un'espressione regolare che genera Lr così definito

  Lr ={x : rev(x), x appartiene a L}
- Idea: Per ipotesi se L è regolare esiste un'espressione regolare e che definisce L
- da e derivare l'espressione regolare che definisce Lr= rev(L)

Esercizi

Sia *rev* un operatore che inverte le sequenze dI simboli (ES. *rev(abc)= cba*). Sia L un linguaggio generato da un'espressione regolare; dimostrare che esiste un'espressione regolare che genera Lr così definito Lr ={x : rev(x), x appartiene a L}

- SOL: Consideriamo le operazioni con cui definiamo le espressioni regolari e verifichiamo cosa succede applicando l'operatore rev alle operazioni che definiscono le espressioni regolari
- Operazione di concatenazione: invertire l'ordine
- Unione, chiusura : non fare nulla

Esempio: data e= (a+b)* c (aa+ba) scriviamo e= e1 e2 e3 - con

e1=(a+b)* e2= c e3= (aa+ba)

Otteniamo rev(e)= rev(e3) rev(e2) rev(e1)

rev(e1) = e1 rev(e2)= e2 rev(e3)=(rev(aa) + rev (ba))=(aa+ab)

Quindi risposta è: rev(e)= (aa+ab) c (a+b)*

Esercizi su linguaggi regolari e non

- Dare una grammatica regolare per il linguaggio $L=\{a^n b^m$, con $n > 0$ e $m>0\}$
- Dato il linguaggio $L=\{a^n b^m$, con $n > m>0\}$ (linguaggio di n a seguite da m b, con n>m)
  - Fornire una grammatica (tipo 2) che genera il linguaggio
  - Il linguaggio non è regolare; fornire una prova o fornire una argomentazione in tal senso sugg. Utilizzare ragionamenti analoghi a quanto fatto per il linguaggio

    $L=\{a^n b^n$, con $n > 0\}$

Esercizi su linguaggi regolari

- Dare una grammatica regolare (tipo 3) per il linguaggio $L=\{a^n b^m$, con $n > 0$ e $m>0\}$
  - Soluzione facile (ma non corretta perché tipo 2)

    $S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid b$
  - Per ottenere grammatica 3 due passi: prima generiamo tutte le a poi passiamo alle b
  - Tutte le a: $S \rightarrow aS \quad S \rightarrow aB$ (queste due produzioni generano stringhe del tipo a^nB, $n>0$)
  - Tutte le b: $B \rightarrow bB \mid b$
  - Grammaica finale: $S \rightarrow aS \quad S \rightarrow aB \quad B \rightarrow bB \mid b$

Esercizi su grammatiche libere dal contesto

Dato il linguaggio $L=\{a^n b^m$, con $n > m>0\}$ (linguaggio di n a seguite da m b, con n>m)

- Fornire una grammatica che genera il linguaggio

- La grammatica $S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid b$ non è corretta (perché?)
- Per generare più a di b due passi: primo passo genero solo a (almeno 1); secondo passo genero $a^n b^m$ con n>m
- Primo passo: $S \rightarrow aS \mid aT$ (in questo modo ottengo stringhe del tipo $a^i T$)
- Secondo passo: $T \rightarrow aTb \mid ab$

Esercizi su grammatiche libere dal contesto

- Dato il linguaggio $L=\{a^n b^m$, con $n > m>0\}$ (linguaggio di n volte a seguite da m volte b, con n>m)
  - Il linguaggio non è regolare; fornire una prova o fornire una argomentazione in tal senso
  - sugg. Utilizzare ragionamenti analoghi a quanto fatto per il linguaggio

    $L=\{a^n b^n$, con $n > 0\}$

Esercizi su grammatiche libere dal contesto

Descrivere una grammatica libera da contesto che generi il seguente linguaggio

$L=\{x^R\#x \mid x \in \{0,1\}^* \; |x| > 1\}$ dove x^R rappresenta la stringa invertita di x e non è vuota

Esempio la stringa 001#001 non appartiene; le stringhe 001#100 e 101#101 appartengono a L

$S \rightarrow 0S0 \mid 1S1 \mid \#$

es: per generare

001#100: $S \rightarrow 0S0 \rightarrow 00S00 \rightarrow 001S100 \rightarrow 001\#100$

Esercizi

- Fornire le espressioni regolari che descrivono i seguenti linguaggi
- Sia $\Sigma = \{a,b\}$
  1. L = $\{w \in \Sigma^* \mid |w|$ è divisibile per 3$\}$
  2. L = $\{w \in \Sigma^* \mid$ la sottostringa ab occorre esattamente due volte in w ma non alla fine$\}$
- Sia $\Sigma = \{a,b,c\}$
  3. L=$\{w \in \{\Sigma^* \mid$w almeno un carattere fra a,b,c non è presente in w$\}$
  4. L= $\{w \in \{a,b,c\}^* \mid$w ogni a è immediatamente seguita da una b$\}$

Esercizi su linguaggi

- Descrivere una grammatica che generi il linguaggio costituito da tutte e sole le stringhe binarie contenenti un numero pari di simboli 1
- Descrivere una grammatica regolare che generi il linguaggio costituito da tutte e sole le stringhe binarie contenenti almeno due simboli 0.
- Dimostrare che il linguaggio costituito da tutte e sole le stringhe binarie con un numero pari di simboli 0 oppure esattamente due simboli 1 è regolare.
- Dimostrare che i due seguenti linguaggi definiti sull'alfabeto {a, b, c} sono regolari
  - Insieme delle stringhe in cui ogni a è immediatamente seguita da una b
  - Insieme delle stringhe in cui ogni a è immediatamente preceduta da una c

Esercizi su grammatiche libere dal contesto

Descrivere una grammatica libera da contesto che generi il seguente linguaggio

$L=\{x^R\#y|x,y\in\{0,1\}^*$, x è una sottostringa di y$\}$ dove x^R srappresenta la stringa invertita di x e non è vuota

Esempio la stringa 001#00111 non appartiene; le stringhe 001#01000 e 10#11110 appartengono a L

$S\rightarrow 0T0U \mid 1T1U$

$T\rightarrow 0T0 \mid 1T1 \mid \#U$

$U\rightarrow 0U \mid 1U \mid \varepsilon$ (ε stringa vuota)

Esercizi su grammatiche libere dal contesto

Si considerino i seguenti linguaggi

1. $L=\{0^n12^n\ n>0\}$ - costituito da tutte e sole le stringhe binarie di n 0 seguiti da un 1 seguito da n 2
2. $L=\{a^nb^{2n,}\ \ n>0\}$ – costituito da n a seguite da 2n b
3. $L=\{a^nb^m\ |m \geq 0, n\geq 0, m > n\}$
4. $L=\{a^nb^m\ |m \geq 0, n\geq 0, m \neq n$ - m diverso da n}

Descrivere una grammatica libera da contesto che generi ciascun linguaggio

Sugg. per 1,2,3 : modificare opportunatamente grammatica per $L=\{a^nb^n\}$

Sugg. per 4: utilizzare due grammatiche: una per $L=\{a^nb^m | m\geq 0, n\geq 0, m > n\}$ e una per $L=\{a^nb^m | m \geq 0, n\geq 0, m < n\}$

Domande

Quali frasi sono vere e quali false; motivare la risposta

(1) Se L non è di tipo 2 allora non è di tipo 3 VERO

(2) Se L1 è di tipo 2 e non di tipo 3 e L1 $\subseteq$ L2, allora L2 non è di tipo 3 FALSO (ad es. nel caso in cui L2 è il linguaggio banale che include tutte le stringhe) .

(3) Se L1 è regolare e L2 è di tipo 2 ma non di tipo 3 allora L1 $\cap$ L2 è di tipo 3 FALSO (stesso esempio del caso precedente)

(4) Dato il linguaggio L abbiamo che $L^* = L \cdot L^*$ se e solo se la stringa vuota ε appartiene a L. VERO

equivalenza fra le diverse definizioni di linguaggi regolari

Scrivere l'espressione regolare che descrive il linguaggio generato dalla grammatica di tipo 3

$S \rightarrow a\,S \mid b\,M$

$M \rightarrow a\,M \mid b\,N \mid b$

$N \rightarrow a\,N \mid a$

Uno: Scrivere il sistema corrispondente

$S = a\,S + b\,M$

$M = a\,M + b\,N + b$

$N = a\,N + a$

equivalenza fra le diverse definizioni di linguaggi regolari

Uno: Scrivere il sistema corrispondente

$S = a\,S + b\,M$

$M = a\,M + b\,N + b$

$N = a\,N + a$

Due: Eliminare la ricorsione

$S = a^*\,b\,M$

$M = a^*\,(b\,N + b)$

$N = a+$

equivalenza fra le diverse definizioni di linguaggi regolari

Due: Eliminare la ricorsione

1. $S = a^* \, b \, M$
2. $M = a^* \, (b \, N + b)$
3. $N = a^+$

Tre: Sostituire le variabili e semplificare

- Sostituendo $N=a^+$ nella 2 otteniamo

  $M = a^* \, (ba^+ \; + \; b)$

- Semplificando otteniamo $M = a^*ba^*$
- Sostituendo $M = a^*ba^*$ nella 1 otteniamo

  $S = a^* \, ba^* \, ba^*$

equivalenza fra le diverse definizioni di linguaggi regolari

- Disegnare l'automa che riconosce il linguaggio descritto dall'espressione regolare **$(a^{+}b + b^{+}a)$**

  Prima trovare automa che costruisce $a^{+}b$ (per $b^{+}a$ analogo); poi combinare i due automi

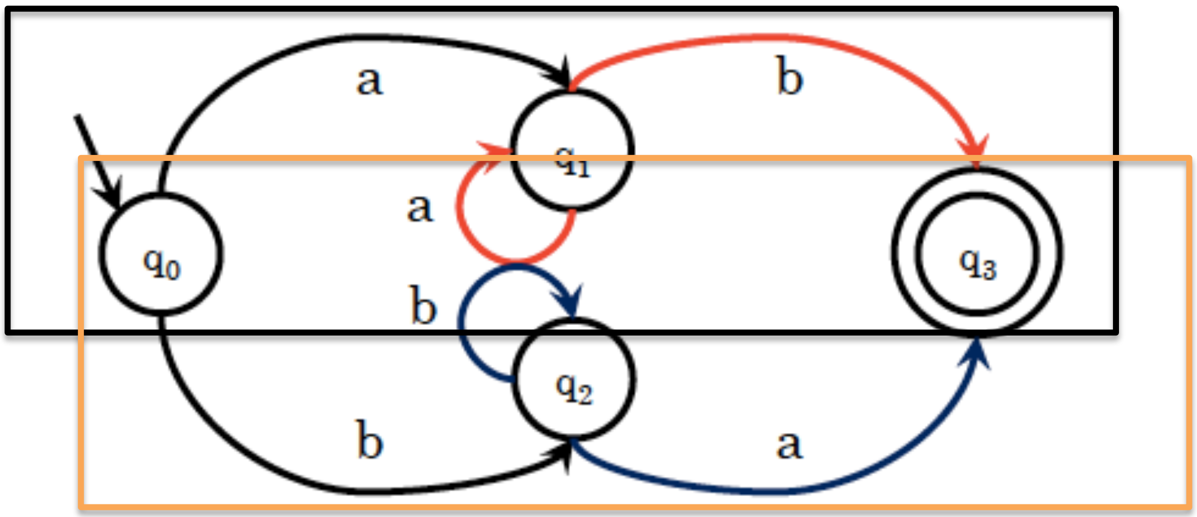
equivalenza fra le diverse definizioni di linguaggi regolari

Disegnare l'automa che riconosce il linguaggio descritto dall'espressione regolare **$(a^+b + b^+a)$**

Prima trovare automa che costruisce a^+b (per b^+a analogo); poi combinare i due automi

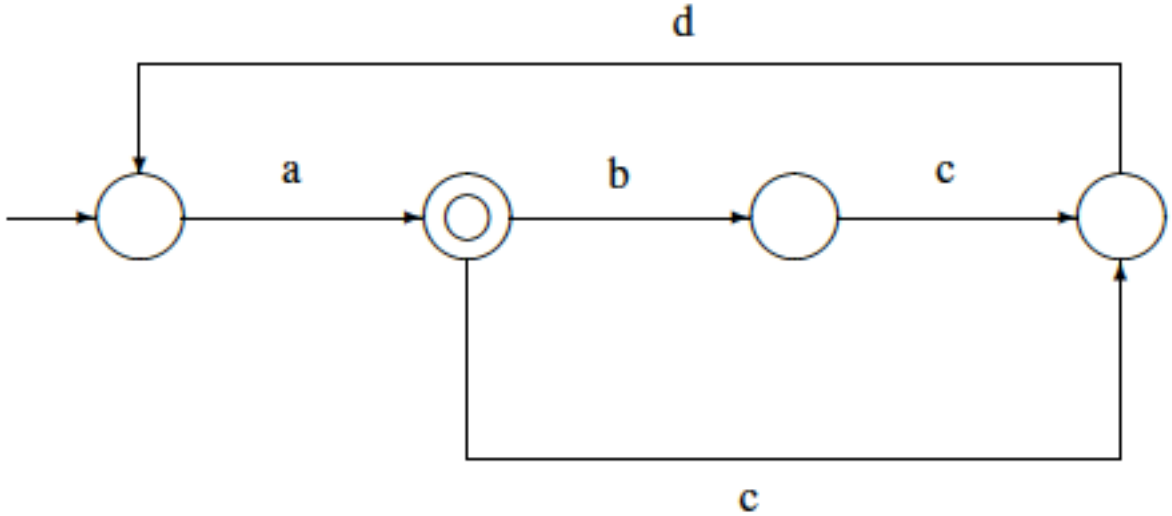
Automa per a^+b

Automa per b^+a



equivalenza fra le diverse definizioni di linguaggi regolari

Definire un'espressione regolare che generi il linguaggio accettato dal seguente automa a stati finiti



Sol. L'espressione regolare inizia con a (dallo stato inziale con a si va nell'unico stato finale)

Dallo stato finale possiamo ritornare nello stato finale attraverso la sequenza bcda oppure cda; questi due percorsi si possono scrivere come (bc+c)da

Questo ciclo può essere eseguito 0 o un numero qualunque di volte

Mettendo insieme tutti i pezzi otteniamo a((bc+c)da)*

equivalenza fra le diverse definizioni di linguaggi regolari

Scrivere l'Automa che riconosce il linguaggio generato dalla grammatica $S \rightarrow aN \qquad N \rightarrow a\,N \mid b\,N \mid b$

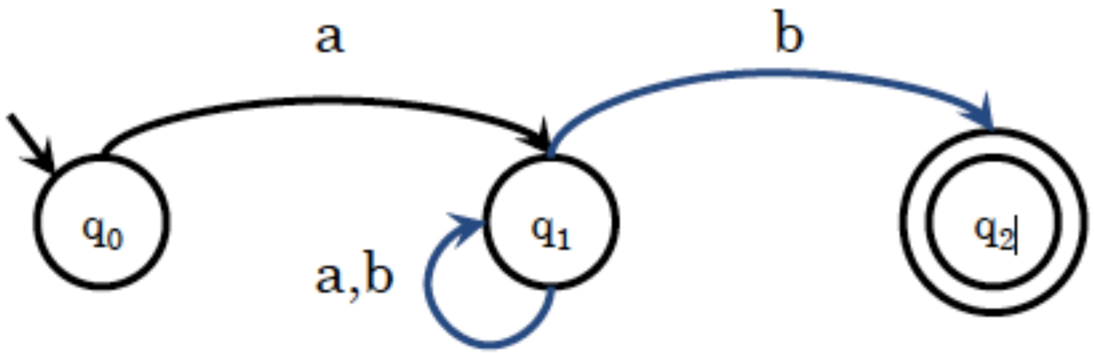
Soluzione

- Per ogni simbolo non terminale ho uno stato e uno per lo stato finale
- Quindi il mio automa ha due stati q0 (iniziale che corrisponde a S) e q1 (che corrisponde a N) e uno stato finale q2
- La produzione $S \rightarrow aN$ implica che nell'automa dallo stato q0 con input a vado nello stato q1
- Le produzioni $N \rightarrow aN$ e $N \rightarrow bN$ implicano che nell'automa dallo stato q1 con input a o b rimango nello stato q1
- La produzione $N \rightarrow b$ implica che nell'automa dallo stato q1 con input b vado nello stato q2

equivalenza fra le diverse definizioni di linguaggi regolari

- Scrivere l'Automa che riconosce il linguaggio generato dalla grammatica

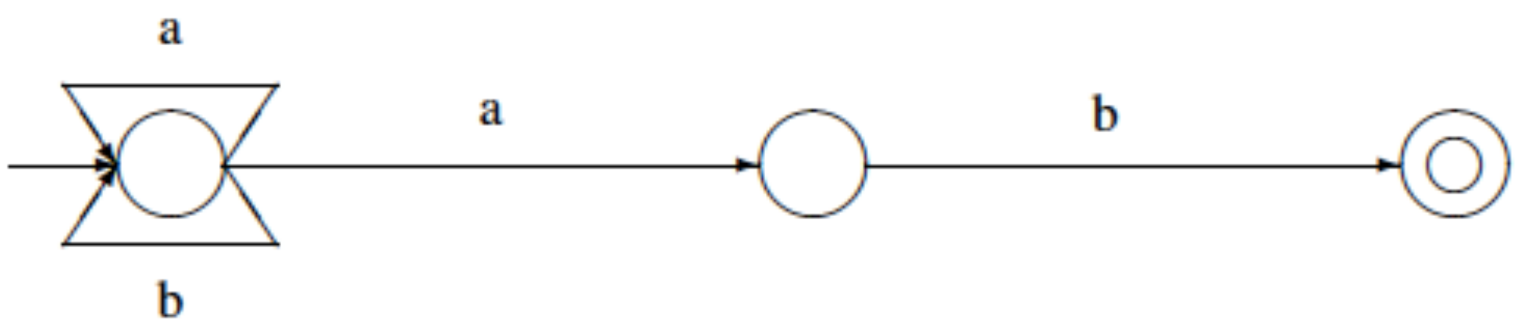
$S \rightarrow aN \qquad N \rightarrow a\,N \mid b\,N \mid b$



L'automa non è deterministico. Perché?

Esercizi su automi

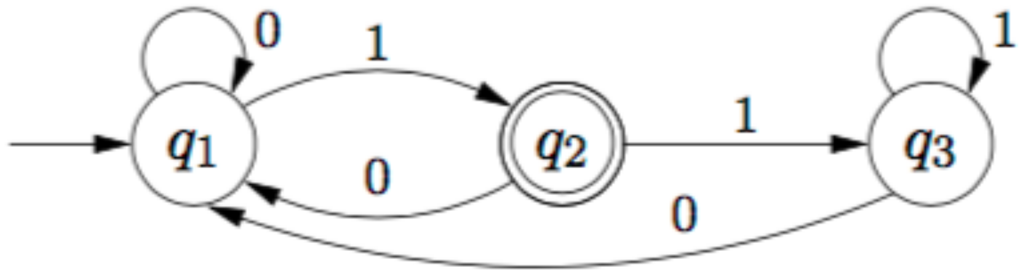
Sia dato il seguente automa

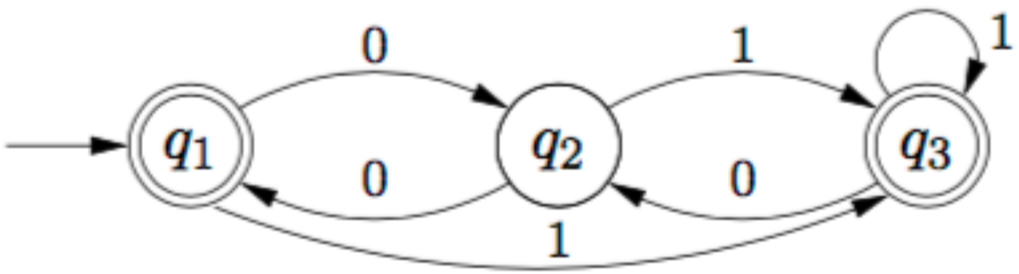


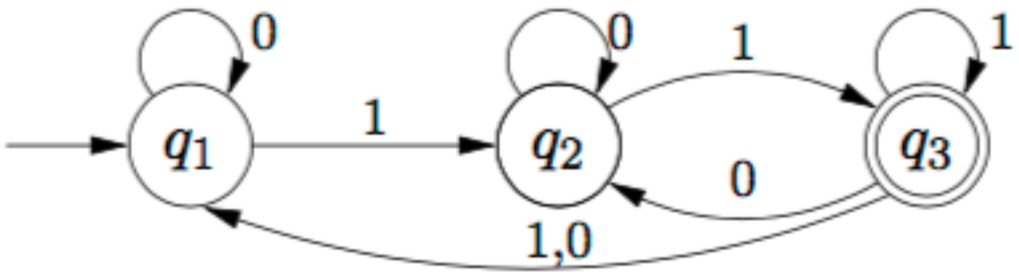
1. Facendo uso della tecnica di costruzione mediante sottoinsiemi, trasformare il seguente automa a stati finiti non deterministico in un automa deterministico equivalente.
2. Fornire una grammatica regolare che genera lo stesso linguaggio accettato dall'automa

Esercizi su automi

Ripetere l'esercizio precedente per i seguenti automi

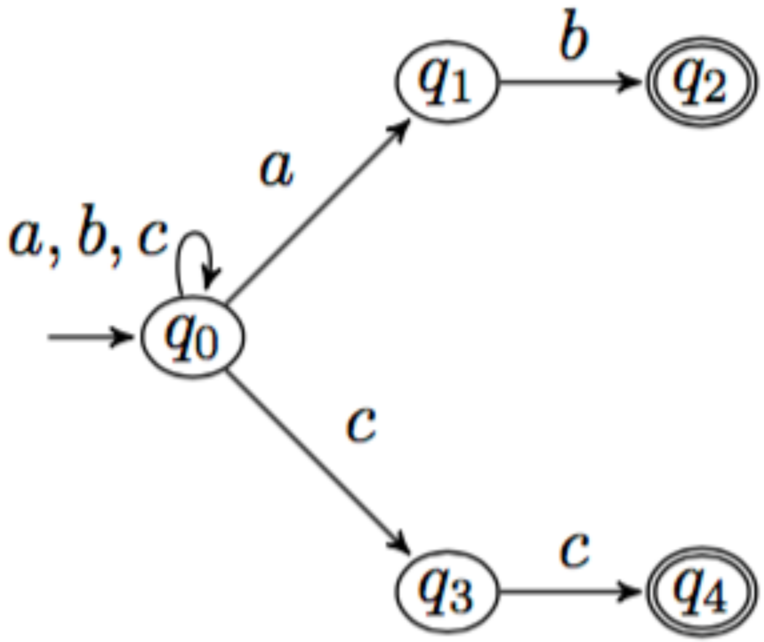






Esercizi

- Disegnare un automa non deterministico che riconosce le stringhe che terminano con bb o ba o baa
- Trasformare il seguente automa a stati finiti non deterministico in uno deterministico equivalente



Esercizi

Data la grammatica

$$\begin{array}{ll} 1 & E \rightarrow E + E \\ 2 & E \rightarrow E * E \\ 3 & E \rightarrow id \\ 4 & E \rightarrow (E) \end{array}$$

Fornire due alberi di derivazione diversi per la stringa

- id*id*id

Fornire tre alberi di derivazione diversi per le stringhe

- id*id + id *id
- id*id*id * id
- id+id * id+id

Fornire un albero di derivazione per la stringa

- (id*id)+ (id * id)

Inoltre giustificare perché non sono presenti altri alberi di derivazione

Esercizi

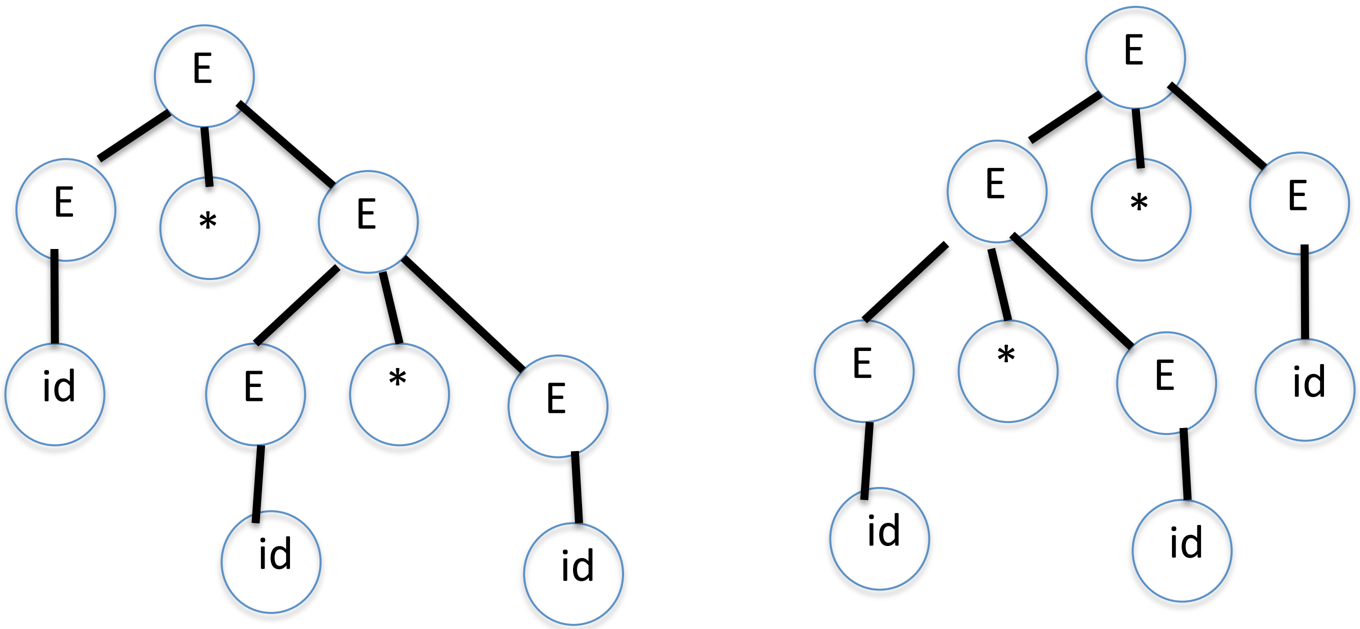
Data la grammatica

$$
\begin{aligned}
&1 \quad E \rightarrow E + E \\
&2 \quad E \rightarrow E * E \\
&3 \quad E \rightarrow id \\
&4 \quad E \rightarrow (E)
\end{aligned}
$$

Fornire due alberi di derivazione diversi per la stringa id*id*id

La prima produzione da utilizzare è certamente E→ E * E

Dopo al **primo** simbolo non terminale E in (E * E) possiamo applicare due produzioni E→id oppure E→ E * E Gli alberi che otteniamo sono i seguenti



NOTA: i due alberi hanno la stessa semantica ma sono diversi! La grammatica è ambigua

Esercizi

Data la grammatica

$$
\begin{array}{ll}
1 & E \rightarrow E + T \\
2 & E \rightarrow T \\
3 & T \rightarrow T * F \\
4 & T \rightarrow F \\
5 & F \rightarrow id \\
6 & F \rightarrow (E)
\end{array}
$$

Ricostruire l'albero di derivazione per le stringhe

1. id*id + id *id
2. id*id*id * id
3. id+id * id+id

Esercizi

Data la grammatica

Ricostruire l'albero di derivazione per la stringa id*id + id *id

1	$E \rightarrow E + T$
2	$E \rightarrow T$
3	$T \rightarrow T * F$
4	$T \rightarrow F$
5	$F \rightarrow id$
6	$F \rightarrow (E)$

NOTA: Per motivi di spazio invece dell'albero si mostra la sequenza di derivazioni (da cui è immediato derivare l'abero)

S→E+T→ T+T→ T * F + T→ F * F + T→ id * F + T → id * id + T ....

Nello scegliere la prima produzione da applicare la scelta di quella giusta (S→ E + T) è basata sul fatto che dopo la moltiplicazione abbiamo un'addizione e usato la precedenza fra + e *

Se avessimo scelto E→ T non avremmo potuto completare

NOTA: Se dobbiamo analizzare la stringa id * id * id * id * id * id + id dobbiamo esaminare la stringa di input fino all'ultimo simbolo di operazione per scegliere la prima produzione da applicare

Esercizi

Data la grammatica

Quale produzione dobbiamo applicare all'inizio?

E→ E+T o E→ T?

$$\begin{array}{ll} 1 & E \to E + T \\ 2 & E \to T \\ 3 & T \to T * F \\ 4 & T \to F \\ 5 & F \to id \\ 6 & F \to (E) \end{array}$$

- Se la stringa da analizzare fosse id * id * id * id * id * id + id

  La prima produzione da applicare è E→ E+T
- Se la stringa da analizzare fosse id * id * id * id * id * id * id

  La prima produzione da applicare è E→ T
- Per decidere se dobbiamo iniziare con E→ E+T o con E→ T dobbiamo esaminare la stringa di input e cercare un simbolo + dopo le moltiplicazioni fra id

Pertanto ci sono casi in cui dobbiamo esaminare quasi tutta la stringa per scegliere la prima produzione da applicare – questo rende lento l'analizzatore sintattico

Esercizi

Definire una grammatica per il linguaggio delle parentesi ben bilanciate ok: (())() no ())()(

Iniziamo con la seguente grammatica

1. S→() | (S) | SS

Altre due possibili grammatiche:

2. S→S (S) $S|\varepsilon$
3. S→() | S() | ()S | (S)

I linguaggi delle parentesi ben bilanciate e quello delle espressioni aritmetiche (di cui all'esercizio precedente) non sono regolari. Fornire una prova o una motivazione.

(Suggerimento: usare ragionamenti affini a quelli usati per il linguaggio $a^n b^n$)

Esercizi

Definire una grammatica per il linguaggio delle parentesi ben bilanciate ok: (())() no ())()(

Iniziamo con la seguente grammatica

1. $S \to () \mid (S) \mid SS$

Altre due possibili grammatiche:

2. $S \to S\,(S)\,S \mid \varepsilon$
3. $S \to () \mid S() \mid ()S \mid (S)$

Esercizi

Sia data la grammatica G = (N, T, P, S) con insieme dei simboli non terminali N = {S, A}, insieme simboli terminali T = {a, b}, assioma S e produzioni :

$S \rightarrow aS \qquad S \rightarrow aA$

$A \rightarrow bA \qquad A \rightarrow b$

- fornire un albero di derivazione della stringa aabbb
- specificare il linguaggio generato dalla grammatica
- discutere se la grammatica è ambigua o no. Se è ambigua fornire una stringa e due derivazioni per questa stringa; se non è ambigua motivre la risposta.

Esercizi

Sia data la grammatica insieme simboli non terminali S, A, simboli terminali a, b, assioma S e produzioni: S → aS | aA A → bA| b

3. discutere se la grammatica è ambigua o no.

SOL. La grammatica non è ambigua. Infatti in presenza di una qualunque stringa di input eseguiamo una derivazione sinistra (cioè espandiamo ogni volta il non terminale più a sinistra).

Ogni volta esaminando il carattere corrente e il carattere successivo possiamo decidere quale produzione applicare

Per espandere S consideriamo il simbolo corrente che è a

- Se il simbolo successivo è a applichiamo S→aS
- Se il simbolo successivo è b applichiamo S→aA

Per espandere A consideriamo il simbolo corrente che è b

- Se la stringa non è finita e il simbolo successivo è b allora applichiamo A→ bA
- Se la stringa è finita allora applichiamo A→ b

In ogni altro caso (caratteri diversi da a e b o stringhe del tipo aabbba riconosciamo abbiamo che la stringa non appartiene al linguaggio)

Quindi l'analisi può essere fatta senza backtracking.

Esercizi

Si consideri la seguente grammatica G (assioma S e simboli terminali {a, b}): $S \rightarrow aAb \mid aSb$ $A \rightarrow aaAbb \mid ab$

1. Fornire la sequenza di derivazioni che a partire dall'assioma deriva la stringa aaaabbbb
2. Descrivere il linguaggio generato dalla grammatica
3. La grammatica è ambigua; fornire due alberi di derivazione per la stringa aaaabbbb e scrivere una nuova grammatica non ambigua che genera lo stesso linguaggio
4. Se ci sono produzioni inutili semplificare la grammatica eliminando le produzioni inutili; motivare la risposta.

Esercizi

Si consideri la seguente grammatica G (assioma S e simboli terminali {a, b}): ***S→ aAb | aSb A→ aaAbb | ab***

4. Se ci sono produzioni inutili semplificare la grammatica eliminando le produzioni inutili; motivare la risposta.

Sol.: il linguaggio genera le stringhe del tipo $a^n b^n$ $n \geq 2$

NON possiamo eliminare

- S→aAb (altrimenti generiamo stringhe con solo simboli terminali e che hanno sempre S)
- A→ab (altrimenti non generiamo stringhe con solo simboli terminali)

Rimangono S→aSb e A→aaAbb

Se eliminiamo S→ aSb possiamo generare la stringa aaabbb?

Se eliminiamo A→aaAbb usando le produzioni ***S→ aAb | aSb*** possiamo generare stringhe del tipo possiamo generare a partire da A la stringa aaAbb usando le altre produzioni? Se la risposta è sì allora A→aaAbb è inutile altrimenti nessuna è inutile

Esercizi

1. Trovare la grammatica libera dal contesto per il linguaggio

   $L = \{a^i b^j c^k \mid i = j + k, j \geq 0 , k \geq 0\}$

2. Trovare l'albero di derivazione della stringa aabc

1.Suggerimento: utilizzare (due volte) la grammatica

$S \rightarrow aSb \mid \varepsilon$ che genera il linguaggio $a^n b^n$, $n \geq 0$

Esercizi

Trovare la grammatica libera dal contesto per il linguaggio

$L = \{a^i b^j c^k \mid j = i + k, j \geq 0 , k \geq 0\}$

- $S \rightarrow aSc \mid T$

  (le due produzioni generano a partire da S stringhe del tipo $a^k T c^k$ $k \geq 0$)

- $T \rightarrow aTb \mid \varepsilon$

  (le due produzioni generano da T stringhe del tipo $a^j b^j$ $j \geq 0$)

Trovare l'albero di derivazione della stringa aabc

Esercizi

Trova la grammatica libera dal contesto per il linguaggio

$L = \{a^i b^j c^k \mid j = i + k, i \geq 0 , k \geq 0\}$

- $S \rightarrow aTbR \mid TbRc \mid \varepsilon$ (le tre produzioni generano da S stringhe di tre tipi: con una a e una b, con una b e una c, la stringa vuota)
- $T \rightarrow aTb \mid \varepsilon$ (le due produzioni generano da T stringhe del tipo $a^j\ b^j$ $j \geq 0$)
- $R \rightarrow bRc \mid \varepsilon$ (le due produzioni generano da R stringhe del tipo $a^j b^j$ $j \geq 0$)

Variante (semplice): Trova la grammatica libera dal contesto per il linguaggio $L = \{a^i b^j c^k \mid j = i + k, i > 0 , k > 0\}$

Basta utilizzare (due volte) la grammatica che genera il linguaggio $a^n b^n$, $n > 0$

Esercizi

Dimostrare che i linguaggi liberi dal contesto sono chiusi rispetto alle operazione di unione, concatenazione e stella

1. Chiusura sotto unione – bisogna dimostrare che dati due linguaggi L1 e L2 liberi dal contesto anche il linguaggio unione (formato dalle stringhe che appartengono ad almeno uno dei due linguaggi) è libero dal contesto; siano L1 e L2 gli insiemi delle stringhe che appartengono ai linguaggi L1 e L2: l'insieme delle stringhe del linguaggio unione è l'insieme L1 $\cup$ L2

SOLUZIONE: Date due grammatiche che generano L1 e L2, assumi che i simboli iniziali per L1 e L2 siano rispettivamente S1 e S2 e rinomina i simboli non terminali di L1 e L2 in modo tale che siano diversi fra loro; allora possiamo definire la grammatica per la loro unione come segue.

- S → S1 | S2
- inoltre usiamo tutte le produzioni per generare L1 e L2
- Per definizione questo genererà qualsiasi stringa generata da S1 o da S2 (o entrambi), che è l'unione dei due linguaggi.

NOTA Se L1 e L2 non sono ambigui la grammatica che oteniamo per L1 $\cup$ L2 è ambigua nel caso che esista una stringa x che appartiene sia a L1 che a L2

Esercizi

Dimostrare che i linguaggi liberi dal contesto sono chiusi rispetto alle operazione di unione, concatenazione e stella

2. Chiusura sotto concatenazione – siano L1 e L2 gli insiemi delle stringhe che appartengono rispettivamente ai linguaggi L1 e L2: l'insieme delle stringhe del linguaggio concatenazione è l'insieme delle stringhe $\{w1\ w2 : w1 \in L1 \wedge w2 \in L2\}$

SOLUZIONE: Usando un argomento simile al precedente possiamo definire una grammatica per la concatenazione di L1 e L2 così. Assumiamo che S1 e S2 siano i simboli iniziali di L1 e L2 e che i simboli non terminali di L1 e L2 in modo tale che siano diversi fra loro; S simbolo iniziale del linguaggio unione

- $S \rightarrow S1\ S2$ (S nuovo simbolo iniziale del linguaggio unione)
- inoltre usiamo tutte le produzioni per generare L1 e L2
- Per definizione, questo genererà una stringa costituita da una stringa di L1 seguita da una stringa da L2, che è la concatenazione dei due linguaggi

Esercizi

Dimostrare che i linguaggi liberi dal contesto sono chiusi rispetto alle operazione di unione, concatenazione e stella (o chiusura di Kleene)

3. Chiusura sotto la stella - mostra che per ogni linguaggio L1 di tipo 2 il linguaggio L1* è di tipo 2. (L* è il linguaggio che non include alcuna stringa e tutte le stringhe del tipo x1 · x2 · x3... dove x1, x2, x3,... sono stringhe in L1 e · rappresenta la concatenazione di stringhe)

SOLUZIONE: Dobbiamo implementare l'operazione stella usando produzioni di tipo 2; assumiamo che S1 sia il simbolo di inizio di L1. Quindi possiamo definire la seguente grammatica:

1. S → S1 S | ε (S nuovo simbolo iniziale del linguaggio L*)
2. inoltre usiamo tutte le produzioni necessarie per generare L1
3. Usando 1 e 2 otteniamo da S la concatenazione di zero o più stringhe che appartengono a L1, che è la definizione di stella

Sia data la seguente grammatica con R simbolo iniziale e unico simbolo non terminale e produzioni

$R \rightarrow (R) | R+R | RR | R^* | a$

- Fornire una derivazione sinistra e una derivazione destra per la stringa (a+a)*a

$R \rightarrow RR \rightarrow Ra \rightarrow R^*a \rightarrow (R)^*a \rightarrow (R+R)^*a \rightarrow (R+a)^*a \rightarrow (a+a)^*a$

- Descrivere il linguaggio generato dalla grammatica
- La grammatica è ambigua?

Data la grammatica con S simbolo iniziale e B, D, altri simboli non terminale e produzioni

- $S \rightarrow Aa \quad A \rightarrow BD \quad B \rightarrow b \mid \lambda \quad D \rightarrow d \mid \lambda$
- Calcola First(X) e Follow(X) per ogni non terminale X
- Costruisci la tavola LL(1)

Definizione di FIRST(α) – α sequenza di simboli terminali e non terminali

- se X è l'insieme di tutte le forme β derivabili da α mediante derivaz. sinistre, per ogni β che inizia con un terminale x, x è in FIRST(α)
- se la stringa λ (stringa vuota) è generabile a partire da α allora appartiene a FIRST(α)

Calcoliamo gli insiemi FOLLOW (solo per i non terminali)

1. inizializzazione: FOLLOW(S) = { \$ }, FOLLOW(X) = $\emptyset$ per $X \neq S$
2. Calcolo di FOLLOW(B): individua le produzioni ove B compare nella parte destra
3. per ciascuna produzione $X \rightarrow \alpha B \beta$ FOLLOW(B) = FOLLOW(B) $\cup$ (FIRST(β) \ {λ})
4. per ciascuna produzione $X \rightarrow \alpha B \beta$ se FIRST(β) può generare la stringa vuota FOLLOW(B) = FOLLOW(B) $\cup$ FOLLOW(X)
5. per ciascuna produzione $X \rightarrow \alpha B$

   FOLLOW(B) = FOLLOW(B) $\cup$ FOLLOW(X)

Sia data la seguente grammatica con S simbolo iniziale e unico simbolo non terminale e produzioni

- $S \rightarrow Aa$ $A \rightarrow BD$ $B \rightarrow b|\lambda$ $D \rightarrow d|\lambda$
- Calcola First(X) e Follow(X) per ogni non terminale X
- Costruisci la tavola LL(1)

Definizione di FIRST(α) – α sequenza di simboli terminali e non terminali)

- se X è l'insieme di tutte le forme β derivabili da α mediante derivaz. sinistre, per ogni β che inizia con un terminale x, x è in FIRST(α)
- se la stringa λ (stringa vuota) è generabile a partire da α allora appartiene a FIRST(α)

First(S) : abbiamo una sola produzione $S \rightarrow Aa$

Ricorda in First abbiamo solo simboli terminali quindi A non appartiene a First(S) e si prosegue

Da A abbiamo solo una produzione $A \rightarrow BD$ e proseguiamo ; alla fine otteniamo

First(S) = {b,d,a}

Sia data la seguente grammatica con S simbolo iniziale e unico simbolo non terminale e produzioni

- $S \rightarrow Aa$ $A \rightarrow BD$ $B \rightarrow b|\lambda$ $D \rightarrow d|\lambda$
- Calcola First(X) e Follow(X) per ogni non terminale X
- Costruisci la tavola LL(1)

Definizione di FIRST(α) – α sequenza di simboli terminali e non terminali)

- se X è l'insieme di tutte le forme β derivabili da α mediante derivaz. sinistre, per ogni β che inizia con un terminale x, x è in FIRST(α)
- se la stringa λ (stringa vuota) è generabile a partire da α allora appartiene a FIRST(α)

First(S) = {b, d, a} First(A) = {b, d, λ}
First(B) = {b, λ} First(D) = {d, λ}

Sia data la seguente grammatica con S simbolo iniziale e unico simbolo non terminale e produzioni

- $S \rightarrow Aa$ $A \rightarrow BD$ $B \rightarrow b|\lambda$ $D \rightarrow d|\lambda$
- Calcola First(X) e Follow(X) per ogni non terminale X
- Costruisci la tavola LL(1)

Calcoliamo gli insiemi FOLLOW (solo per i non terminali) – ricorda $ rappresenta fine stringa

1. inizializzazione: FOLLOW(S) = { $ }, FOLLOW(X) = Ø per $X \neq S$
2. Calcolo di FOLLOW(B): individua le produzioni ove B compare nella parte destra
3. per ciascuna produzione $X \rightarrow \alpha B \beta$ FOLLOW(B) = FOLLOW(B) ∪ (FIRST(β) \ {λ})
4. per ciascuna produzione $X \rightarrow \alpha B \beta$ se FIRST(β) può generare la stringa vuota FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(X)
5. per ciascuna produzione $X \rightarrow \alpha B$

   FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(X)

Follow(S) = {$} Follow(A) = {a} Follow(B) = {d, a} Follow(D) = {a}

Sia data la seguente grammatica con S simbolo iniziale e unico simbolo non terminale e produzioni

- $S \rightarrow Aa$ $A \rightarrow BD$ $B \rightarrow b \mid \lambda$ $D \rightarrow d \mid \lambda$
- Costruisci la tavola LL(1)

First(S) = {b, d, a} First(A) = {b, d, λ} First(B) = {b, λ} First(D) = {d, λ}

Follow(S) = {\$} Follow(A) = {a} Follow(B) = {d, a} Follow(D) = {a}

Nota: abbiamo λ produzioni; quindi per scegliere quale produzione usare da B devo usare FOLLOW; analogamente per D

Esempio: ba $S \rightarrow Aa \rightarrow ba$

	a	b	d	$
S	Aa	Aa	Aa	
A	BD	BD	BD	
B	λ	b	λ	
D	λ		d	

Data la grammatica con prog   simbolo iniziale e {prog, stmt, stmts, block, expr} simboli non terminali e produzioni (terminali: if,while, id…)

prog → stmt

stmt → if expr then block | while expr do block | expr ;

expr→ term=>id|isZero? term|not expr|++id|−−id

term → id | const

block → stmt | { stmts }

stmts → istr stmts | λ

- Calcola First(X) e Follow(X) per ogni non terminale X
- Costruisci la tavola LL(1)

Data la grammatica con ‘prog’ simbolo iniziale e {prog, istr, stmts, block, expr} simboli non terminali e produzioni

scriviamo le produzioni in ordine

1. prog → stmt
2. stmt → if expr then block
3. stmt → while expr do block
4. stmt → expr ;
5. expr → term =>id
6. expr → isZero? term
7. expr → not expr
8. expr → ++id
9. expr → --id
10. term → id
11. term → const
12. block → stmt
13. block → { stmts }
14. stmts → stmt stmts
15. stmts → λ

First(prog) = First(stmt) (da prog una sola produz. prog→stmt)

First(term)= {id, const}

First(expr) = {id, ++,--,isZero?, not …. } (devo aggiungere i First(term) produzione 5

First(stmt)= {if, while, …. } (devo aggiungere i First(expr) – produzio e 4.

First(stmts) = {….}

Data la grammatica con prog simbolo iniziale e {prog, istr, stmts, block, expr} simboli non terminali e produzioni

scriviamo le produzioni in ordine

1. prog → stmt
2. stmt → if expr then block
3. stmt → while expr do block
4. stmt → expr ;
5. expr → term =>id
6. expr → isZero? term
7. expr → not expr
8. expr → ++id
9. expr → --id
10. term → id
11. term → const
12. block → stmt
13. block → { stmts }
14. stmts → stmt stmts
15. stmts → λ

First(prog) = First(stmt)

First(term)= {id, const}

First(expr) = {id, const, isZero?, not, ++, −−} (usiamo First(term))

First(stmt)= {if, while, id, const, isZero?, not, ++, −−}
(usiamo First(expr))

First(stmts) = { if, while, id, const, isZero?, not, ++, −−,}
(usiamo First(stmt))

Data la grammatica con prog simbolo iniziale e {prog, istr, stmts, block, expr} simboli non terminali e produzioni

scriviamo le produzioni in ordine

1. prog → stmt
2. stmt → if expr then block
3. stmt → while expr do block
4. stmt → expr ;
5. expr → term =>id
6. expr → isZero? term
7. expr → not expr
8. expr → ++id
9. expr → --id
10. term → id
11. term → const
12. block → stmt
13. block → { stmts }
14. stmts → stmt stmts
15. stmts → λ

Follow(prog) = {\$} (facile)

Follow(stmt) = {\$, } (abbiamo λ produzioni)

Follow(expr) = {then, do, ;} (facile)

Follow(term) = {=>, then, do, ;} (facile)

Follow(block) = {\$,.. }

Follow(stmts) = { '}' …. } (abbiamo λ produzioni)

Data la grammatica con prog simbolo iniziale e {prog, istr, stmts, block, expr} simboli non terminali e produzioni

scriviamo le produzioni in ordine

1. prog → stmt
2. stmt → if expr then block
3. stmt → while expr do block
4. stmt → expr ;
5. expr → term =>id
6. expr → isZero? term
7. expr → not expr
8. expr → ++id
9. expr → --id
10. term → id
11. term → const
12. block → stmt
13. block → { stmts }
14. stmts → stmt stmts
15. stmts → λ

Follow(stmts) = { '}' .... } (abbiamo λ produzioni)

Se stmts→ λ allora io posso avere

Block→ {stmts} → { } opuure

Block → {stmts} → {stmt stmts} →

Data la grammatica con prog simbolo iniziale e {prog, istr, stmts, block, expr} simboli non terminali e produzioni

scriviamo le produzioni in ordine

1. prog → stmt
2. stmt → if expr then block
3. stmt → while expr do block
4. stmt → expr ;
5. expr → term =>id
6. expr → isZero? term
7. expr → not expr
8. expr → ++id
9. expr → --id
10. term → id
11. term → const
12. block → stmt
13. block → { stmts }
14. stmts → stmt stmts
15. stmts → λ

Follow(prog) = {\$}
Follow(stmt) = {\$, if, while, id, const, isZero?, not, ++, −−}
Follow(expr) = {then, do, ;}
Follow(term) = {=>, then, do, ;}
Follow(block) = {\$, if, while, id, const, isZero?, not, ++, −−}
Follow(stmts) = {'}'}

Data la grammatica G con simboli terminali **{id , " , + }** e produzioni

S → id|"T" **T→SV** **V→λ|+SV**

- Qual è il inguaggio generato da G?
- Calcola First(α) per ogni produzione X → α e Follow(A) per ogni non terminale A
- Costruisci la tavola LL(1)

Definizione di FIRST(α) – α sequenza di simboli terminali e non terminali)

- definisce un insieme di simboli terminali a partire da α
- se X è l'insieme di tutte le forme β derivabili da α mediante derivaz. sinistre, per ogni β che inizia con un terminale x, x è in FIRST(α)
- se la stringa λ (stringa vuota) è generabile a partire da α allora appartiene a FIRST(α)

FIRST(S→id)= {id} FIRST(S→"T"}= { "} FIRST(T→SV)= ?

FIRST(V→λ

Sia data la seguente grammatica con A simbolo iniziale e {t,u,v,w,x} insieme dei simboli terminali:

A -> B D	B -> C w B | λ
D -> D x B | v	C -> t | t u

- Riscrivere la grammatica per renderla LL(1) (Ricorda: non eliminare mai produzioni che corrispondono al simbolo iniziale)

- Calcola FIRST e FOLLOW e costrusci la tavola di parsing

Riscrivere la grammatica per renderla LL(1)

1. Innanzitutto notiamo il ruolo di C che entra nelle due produzioni

B -> C w B | λ C -> t | t u

Eliminando il simbolo C otteniamo (in nero le nuove produzioni)

- A -> B D B -> t w B | t u w B | λ D -> D x B | v

2. Ora eliminiamo la ricorsione sinistra su D introducendo nonterm. D' e al posto di

D -> D x B | v introduciamo D -> v D' D' -> x B D' | λ

Abbiamo così ottenuto

A → B D B → t w B | t u w B | λ

D → v D' D' → x B D' | λ

Abbiamo ottenuto

$A \rightarrow B\ D$ $\qquad$ $B \rightarrow t\ w\ B \mid t\ u\ w\ B \mid \lambda$

$D \rightarrow v\ D'$ $\qquad$ $D' \rightarrow x\ B\ D' \mid \lambda$

3. Notiamo che esiste ancora un problema su B: ci sono due produzioni che iniziano con lo stesso terminale (t). Fattorizzando 't' introduciamo un nuovo non terminale B' e al posto di

$B \rightarrow t\ w\ B \mid t\ u\ w\ B$ scriviamo $B \rightarrow t\ B' \mid \lambda$ $\qquad$ $B' \rightarrow w\ B \mid u\ w\ B$

4. Inoltre possiamo riscrivere

$A \rightarrow BD$ come $A \rightarrow tB'D \mid D$ e $A \rightarrow D$ come $A \rightarrow vD'$

Abbiamo ottenuto la seguente grammatica equivalente

$A \rightarrow tB'D \mid vD'$ $\qquad$ $B \rightarrow t\ B' \mid \lambda$ $\qquad$ $B' \rightarrow w\ B \mid u\ w\ B$

$D \rightarrow v\ D'$ $\qquad$ $D' \rightarrow x\ B\ D' \mid \lambda$

Definizione di FIRST(α) – α sequenza di simboli terminali e non terminali)

- definisce un insieme di simboli terminali a partire da α
- se X è l'insieme di tutte le forme β derivabili da α mediante derivaz. sinistre, per ogni β che inizia con un terminale x, x è in FIRST(α)
- se la stringa λ (stringa vuota) è generabile a partire da α allora appartiene a FIRST(α)

Abbiamo ottenuto la seguente grammatica equivalente

$A \rightarrow tB'D \mid vD'$ $\quad B \rightarrow t\,B' \mid \lambda$ $\quad B' \rightarrow w\,B \mid u\,w\,B$

$D \rightarrow v\,D'$ $\quad D' \rightarrow x\,B\,D' \mid \lambda$

Calcoliamo FIRST()

- FIRST(A) = {t,v}
- FIRST(B) = { t, λ }
- FIRST(B') = { w, u }
- FIRST(D) = { v }
- FIRST(D') = { x, λ}

Definizione di FIRST(α) – α sequenza di simboli terminali e non terminali)

Abbiamo ottenuto la seguente grammatica equivalente

A -> B D   B -> t B' | λ B' -> w B | u w B

D -> v D'   D' -> x B D' | λ

Posso riscrivere la grammatica: sostituendo a A→BD

Prima la produzione A→ tB'D | D

e poi A→ D come A→ vD'

Ottengo come grammatica

A→tB'D | vD'   B -> t B' | λ B' -> w B | u w B

D -> v D'   D' -> x B D' | λ

- FIRST(A) = { t, v }
- FIRST(B) = { t, λ }
- FIRST(B') = { w, u } (facile)
- FIRST(D) = { v } (facile)
- FIRST(D') = { x, λ}

Abbiamo ottenuto la seguente grammatica equivalente

$A \rightarrow tB'D \mid vD'$ $B \rightarrow t\,B' \mid \lambda$ $B' \rightarrow w\,B \mid u\,w\,B$

$D \rightarrow v\,D'$ $D' \rightarrow x\,B\,D' \mid \lambda$

Calcoliamo gli insiemi FOLLOW (solo per i non terminali)

1. inizializzazione: FOLLOW(S) = { \$ }, FOLLOW(A) = Ø per A ≠ S
2. per calcolare FOLLOW(B) individua le produzioni ove B compare nella parte destra
3. per ciascuna prod. $X \rightarrow \alpha B \beta$ FOLLOW(B) = FOLLOW(B) ∪ (FIRST(β) \ {λ})
4. per ciascuna produzione $X \rightarrow \alpha B \beta$ se FIRST(β) può generare la stringa vuota FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(X)
5. per ciascuna produzione $X \rightarrow \alpha B$ FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(X)

- FOLLOW(A) = { \$ } (facile 1.)
- FOLLOW(B) = FOLLOW(B') ∪ FIRST(D) \ {λ} = { v, x, \$ } (Prod. B'→ wbB|uwB implicano che dobbiamo includere FOLLOW(B') vedi 5; A→BD implica che nel calcolo di FOLLOW(B) dobbiamo usare FIRST(D') vedi 3.)
- FOLLOW(B') = FIRST(D) ∪ FOLLOW(B) = { v, x, \$ } (Le produzioni A-> BD B -> λ e B'→wb|uwB implicano che nel calcolo di FOLLOW(B') dobbiamo usare FIRST(D')
- FOLLOW(D) = FOLLOW(A) = { \$ } (facile 5.)
- FOLLOW(D') = FOLLOW(A) ∪ FOLLOW(D) = { \$ } (facie 5.)

Abbiamo ottenuto la seguente grammatica equivalente

A -> B D   B -> t B' | λ   B' -> w B | u w B

D -> v D'   D' -> x B D' | λ

Calcoliamo gli insiemi FOLLOW (solo per i non terminali)

1. inizializzazione: FOLLOW(S) = { $ }, FOLLOW(A) = Ø per A ≠ S
2. per calcolare FOLLOW(B) individua le produzioni ove B compare nella parte destra
3. per ciascuna produzione X → αBβ FOLLOW(B) = FOLLOW(B) ∪ (FIRST(β) \ {λ})
4. per ciascuna produzione X → αBβ se FIRST(β) può generare la stringa vuota
   FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(X)
5. per ciascuna produzione X → αB FOLLOW(B) = FOLLOW(B) ∪ FOLLOW(X)

- FOLLOW(A) = { $ } (facile 1.)
- FOLLOW(B) = FOLLOW(B') ∪ FIRST(D) \ {λ} = { v, x, $ } (La produzione B -> λ implica che nel calcolo di FOLLOW(B) dobbiamo usare FIRST(D')
- FOLLOW(B') = FIRST(D) ∪ FOLLOW(B) = { v, x, $ } (Le produzioni
  A-> B D e B -> λ e B'→wb|uwB implicano che nel calcolo di FOLLOW(B')
dobbiamo usare FIRST(D')
- FOLLOW(D) = FOLLOW(A) = { $ } (facili)
- FOLLOW(D') = FOLLOW(A) ∪ FOLLOW(D) = { $ }

Abbiamo ottenuto la seguente grammatica equivalente

A -> B D B -> t B' | λ B' -> w B | u w B

D -> v D' D' -> x B D' | λ

Calcoliamo gli insiemi FIRST e FOLLOW

- FIRST(A) = { t, v } (Una sola produzione da A→BD se da B scegliamo B→tB' allora abbiamo t; se scegliamo B→ λ allora come primo simbolo in input abbiamo v - da D→vD')
- FOLLOW(A) = { $ } (il FOLLOW del simbolo iniziale è **sempre** $)
- FIRST(B) = { t, λ }
- FOLLOW(B) = FOLLOW(B') ∪ FIRST(D') \ {λ} = { v, x, $ } (La produzione B -> λ implica che nel calcolo di FOLLOW(B) dobbiamo usare FIRST(D')
- FIRST(B') = { w, u } (facile)
- FOLLOW(B') = FIRST(D) ∪ FOLLOW(B) = { v, x, $ } (Le produzioni

  A-> B DB -> λ e B'→wb|uwB implicano che nel calcolo di FOLLOW(B') dobbiamo usare FIRST(D')
- FIRST(D) = { v } FOLLOW(D) = FOLLOW(A) = { $ } (facili)
- FIRST(D') = { x, λ} FOLLOW(D') = FOLLOW(A) ∪ FOLLOW(D) = { $ }

Calcolo della tabella di parsing

A→ tB'D | vD'   B -> t B' | λ   B' -> w B | u w B

D →> v D'   D' -> x B D' | λ

- FIRST(A) = { t, v }
- FIRST(B) = { t, λ }
- FIRST(B') = { w, u }
- FIRST(D) = { v }
- FIRST(D') = { x, λ }

FOLLOW(A) = { $ }
FOLLOW(B) = { v, x, $ }
FOLLOW(B') = { v, x, $ }
FOLLOW(D) = { $ }
FOLLOW(D') = { $ }

NOTA (eps) rappresenta ε

	t	u	v	w	x	$
A	t B' D		v D'			
B	t B'		(eps)		(eps)	(eps)
B'		u w B		w B		
D			v D'			
D'					x B D'	(eps)

Sia data la seguente grammatica che genera indirizzi di posta (ind è il simbolo iniziale, ind e nome sono i simboli nonterminali, @ . e id i simboli terminali)

ind→ nome@ nome . id

nome → id | id . nome

La grammatica genera stringhe del tipo:

id@id.id   id.id@id.id.id.id   id.id.id@id.id

1. La grammatica non è LL(1) : motivare perché
2. Riscrivere la grammatica per renderla LL(1) (Ricorda: non eliminare mai produzioni che corrispondono al simbolo iniziale)
3. Calcola FIRST e FOLLOW della grammatica proposta
4. costruisci la tavola di parsing per la grammatica proposta

1. Spiegare perché non è LL(1)

Le produzioni nome → id | id . nome

creano problemi

ind→ nome@ nome . id
nome → id
nome → id . nome

- Consideriamo ad esempio la stringa id@id.id in input; iniziamo con ind → nome@ nome . id → id@ nome.id
- a questo punto come si prosegue? Dobbiamo espandere nome e abbiamo in input id. Come decidiamo quale produzione applicare fra nome → id e nome → id.nome?
- Ovviamente nel nostro caso dobbiamo applicare nome → id ; ma come scegliamo se vediamo solo un token in input (id)?
- Infatti se input fosse id@id.id.id avremmo dovuto applicare nome→id.nome
- Nota le due produzioni nome → id e nome → id.nome hanno lo stesso FIRST e quindi esaminando un solo carattere in input non possiamo decidere (conflitto FIRST fra due produzioni con stesso nonterminale a sinistra)

2. Scrivere una grammatica equivalente LL(1)

   ind → nome @ id . nome

   nome → id nome'

   nome' → ε | . id nome’

ind→ nome@ nome . id nome → id nome → id . nome

Un errore comune è fattorizzare le produzioni del nonterminale nome (ok) MA di dimenticare di modificare le altre produzioni

- Infatti supponiamo di scrivere

  ind→ nome@ nome . id

  nome → id nome'

  nome' → ε | nome’ . id (al posto di nome’→ ε | . id nome’)

- In questo caso per analizzare id@id.id.id abbiamo

  ind→ nome @ nome.id → id nome’ @nome.id → id @nome.id → id@ id. nome’ .id → id@id. ???

- A questo punto il token in input è id; come facciamo a sapere se l’input è id@id.id oppure id@id.id.id oppure id@id.id.id.id…?
- Nel primo caso dobbiamo applicare nome’ → ε negli altri

  nome’ → nome’ . Id

- Tecnicamente abbiamo un conflitto fra FIRST e FOLLOW

3. Calcolare FIRST e FOLLOW della grammatica proposta

ind → nome @ id . nome

nome → id nome'

nome' → ε | . id nome’

- Gli insiemi FIRST sono

  FIRST(ind) = { id }

  FIRST(nome) = { id }

  FIRST(nome') = { ε , . } (rispettivamente dati dalle produzioni nome' → ε e nome’ →. id nome’)

- Gli insiemi FOLLOW sono

  FOLLOW(ind) = { $ } (ind è simbolo iniziale, dopo aver espanso ind abbiamo finito l’input e $ rappresenta fine input)

  FOLLOW(nome) = { @, $ } (nome occorre due volte in ind → nome @ id . nome; la prima volta a nome segue @, la seconda $)

  FOLLOW(nome') = { @, $ }

4. Costruire la tabella Calcolare FIRST e FOLLOW della grammatica proposta

ind → nome @ id . nome

nome → id nome'

nome' → ε | . id nome’

- Gli insiemi FIRST sono

  FIRST(ind) = { id }   FIRST(nome) = { id }   FIRST(nome') = { ε , . }

- Gli insiemi FOLLOW sono

  FOLLOW(ind) = {$}   FOLLOW(nome) = {@,$}   FOLLOW(nome’) = {@,$}

	id	.	@	$
ind	nome@id.nome			
nome	id nome’			
nome’		. id nome’	ε	ε

Completare parse analisi bottom-up

$E \rightarrow F$

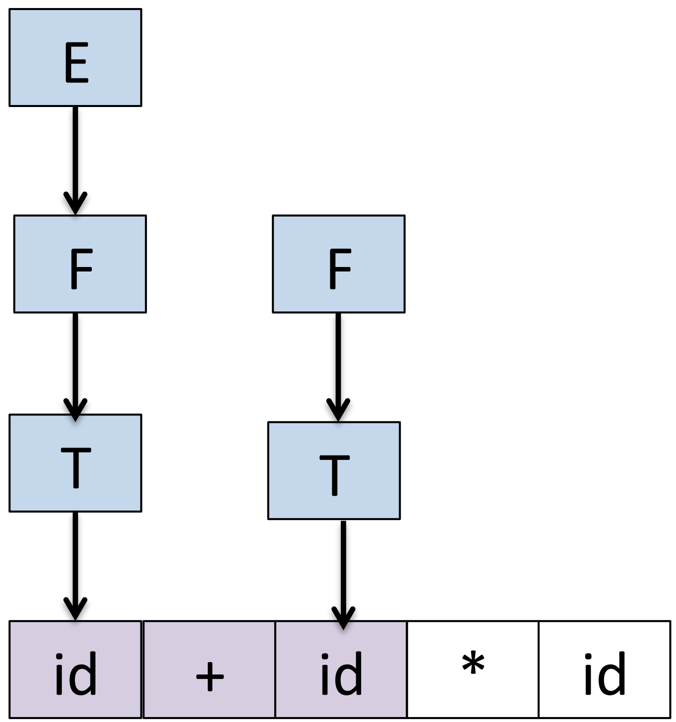
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow id$

$T \rightarrow (E)$



Completare parse analisi bottom-up

$E \rightarrow F$

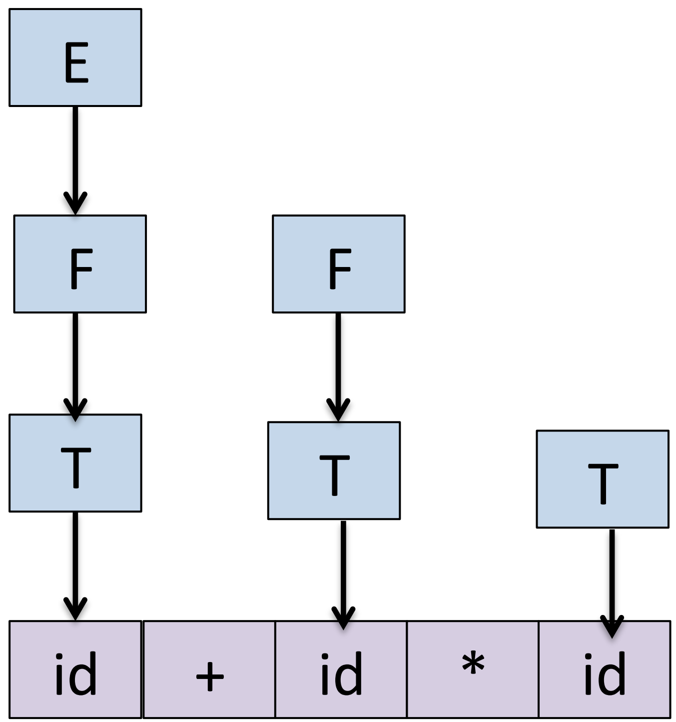
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow id$

$T \rightarrow (E)$



Completare parse analisi bottom-up

$E \rightarrow F$

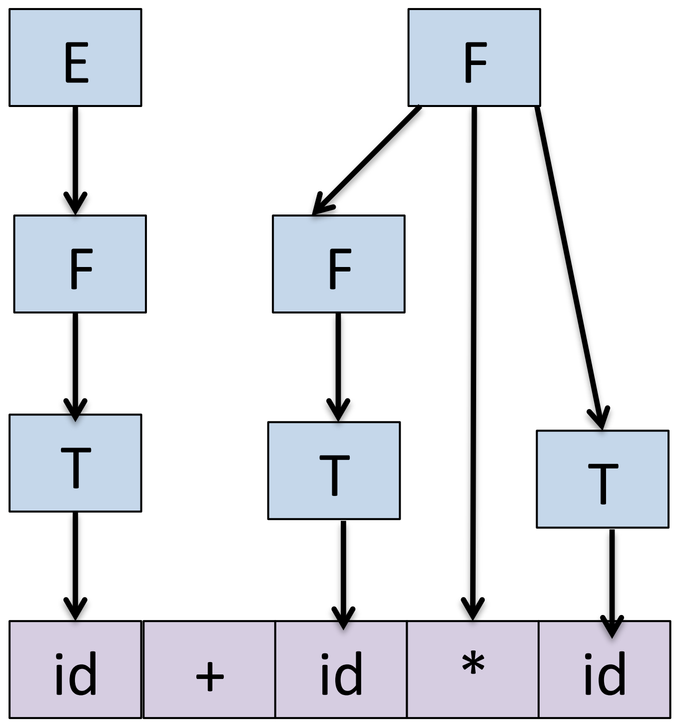
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow id$

$T \rightarrow (E)$



Completare parse analisi bottom-up

$E \rightarrow F$

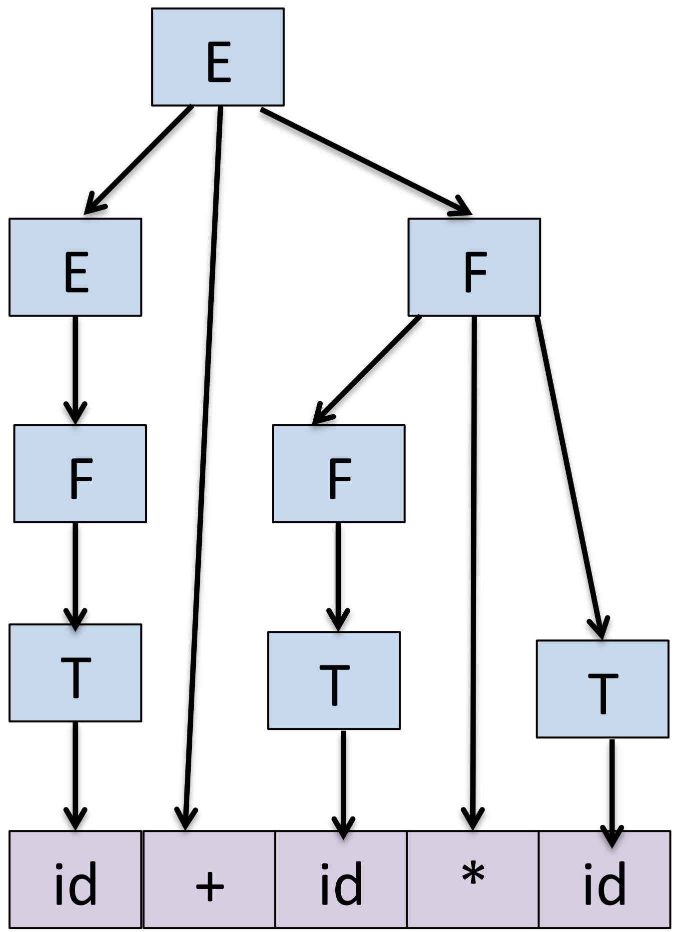
$E \rightarrow E + F$

$F \rightarrow F * T$

$F \rightarrow T$

$T \rightarrow id$

$T \rightarrow (E)$



Eseguire analisi bottom-up

$E \rightarrow F$
$E \rightarrow E + F$
$F \rightarrow F * T$
$F \rightarrow T$
$T \rightarrow id$
$T \rightarrow (E)$

Eseguire l'analisi passo passo segnalando ad ogni passo

- operazione eseguita (shift/reduce)
- il contenuto della pila

id	+	id	*	id	+	id

Derivazione destra

- Grammatica:
  1. S → S (S)   2. S → ε

1. Trovare derivazione destra di ()()
2. Linguaggio generato dalla grammatica: stringhe di parentesi

Ma posso generare ()(()) ?
(()))(?

Derivazione destra: ogni volta espandi il non terminale più a destra

Derivazione destra

- Grammatica:
  1. S → S (S)   2. S → ε
- Trovare derivazione destra di ()()

S ⇒ S (S)

⇒ S ( )

⇒ S (S) ( )

⇒ S ( ) ( )

⇒ ( ) ( )

Derivazione destra in ordine inverso
(con handle)

S ( ) ( ) ⇒ ( ) ( ) // handle stringa vuota

⇒ S (S) ( ) // handle stringa vuota

⇒ S ( ) // handle è S (S)

⇒ S (S) // handle è stringa vuota

⇒ S // handle è (S) S

- Grammatica:
  1. S → S (S) 2. S → ε
- Derivazione destra di ()()

S ⇒ S (S) // handle è S (S)

⇒ S ( ) // handle è stringa vuota

⇒ S (S) ( ) // handle è S (S)

⇒ S ( ) ( ) // handle è stringa vuota

⇒ ( ) ( ) // handle è stringa vuota

Data la seguente tabella eseguire parsing bottom-up

Stato	ACTION			GOTO
	(	)	$	S
0	r2	r2	r2	1
1	s2		Acc.	
2	r2	r2	r2	3
3	s2	s4		
4	r1	r1	r1	

Parsing con tavole. Nota non grammatica estesa

Grammatica:

1. $S \rightarrow S\ (S)$ 2. $S \rightarrow \varepsilon$

eseguire parsing bottom-up (Acc. = accetta)

Stato	ACTION			GOTO
	(	)	$	S
0	r2	r2	r2	1
1	s2		Acc.	
2	r2	r2	r2	3
3	s2	s4		
4	r1	r1	r1	

Stack	**Input**	**Action**
0	()()$	reduce (2) $S \rightarrow \varepsilon$ e push stato 1 su stack
0S1	()()$	shift (e push stato 2 su stack
0S1(2	)()$	reduce (2) $S \rightarrow \varepsilon$ e push stato 3
0S1(2S3	)()$	shift) e push stato 4
0S1(2S3)4	()$	reduce by (1) $S \rightarrow S(S)$ e push stato 1
0S1	()$	shift (e push stato 2
0S1(2	)$	reduce (2) $S \rightarrow \varepsilon$ e push stato 3
0S1(2S3	)$	shift) e push stato 4
0S1(2S3)4	$	reduce (1) $S \rightarrow S(S)$ e push stato 1
0S1	$	accetta

Costruzione tavole Action e Goto

- Definisci Grammatica estesa :

  1. $S' \rightarrow S$ 2. $S \rightarrow S(S)$ 3. $S \rightarrow \varepsilon$

- Notazione punto; si ottiene

1. $S' \rightarrow S$: $S' \rightarrow .S$ e $S' \rightarrow S.$

2. $S \rightarrow S(S)$:

   $S \rightarrow .S(S)$ | $S.(S)$ | $S(.S)$ | $S(S.)$ | $S(S).$

3. $S \rightarrow \varepsilon$: $S \rightarrow .$

Grammatica:

1. S’ → S   2. S → S (S)   3. S → ε

Considera S’ →S

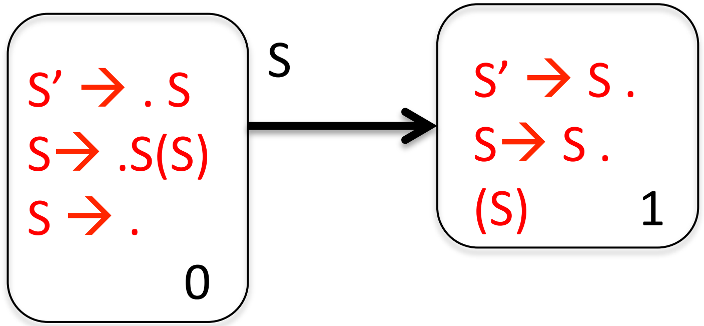
Stato 0 contiene

S’ → . S e la sua chiusura data da

S→ . S (S) e S → .

Da stato 0

- con input S stato 1: S’ → S. e S→ S. (S)



Grammatica:

1. S' → S   2. S → S (S)   3. S → ε

Considera S' →S

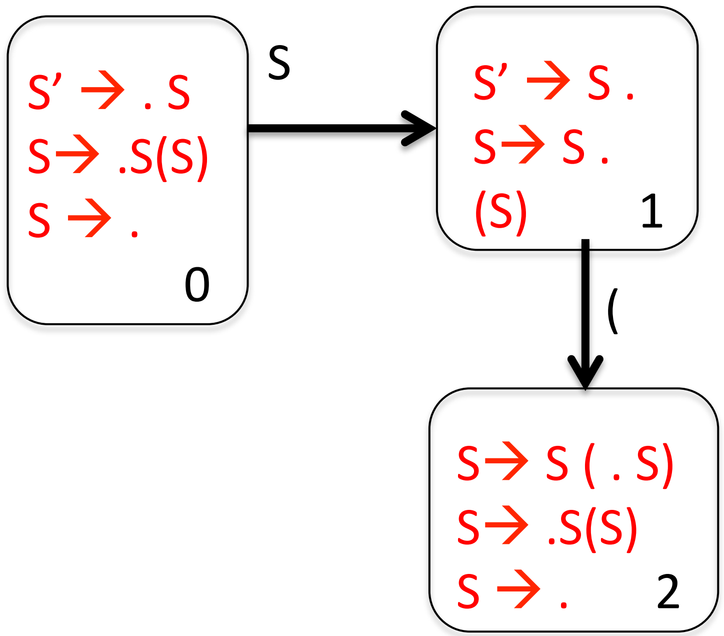
Stato 0 contiene

S' → . S e la sua chiusura data da

S→ . S (S) e S → .

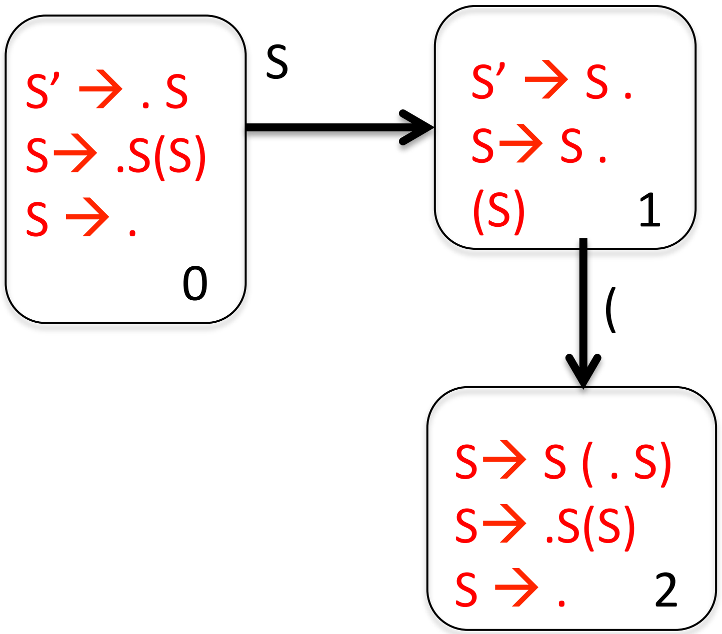
Da stato 0

- con input S stato 1: S' → S. e S→ S. (S)



Grammatica:

1. S' → S   2. S → (S)S   3. S → ε

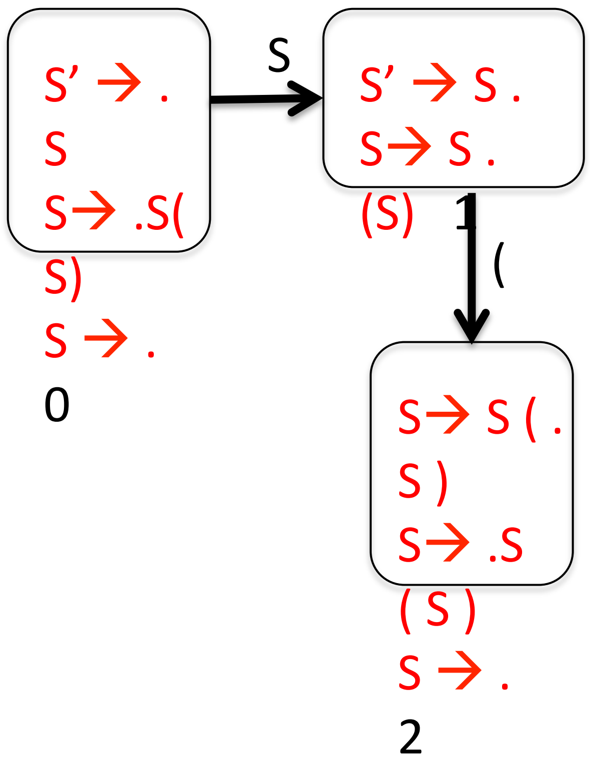


In ogni stato posso andare in un nuovo stato a segutio di nuovo input

Come proseguire?

Grammatica:

1. S' → S   2. S → (S)S   3. S → ε



Come proseguire?

Nota:

- da stato 0 proseguo solo con input S
- Da stato 1 proseguo solo con input (
- Da stato 2 proseguo solo con S

  e ottengo stato 3 con

  S→ S (S .) S→ S . (S)
- Da stato 3 proseguo
  - con (e ottengo stato 4 S→S (S).