Analisi sintattica – LL(k)

Prof. A. Morzenti

aa 2008-2009

ANALISI SINTATTICA

Data una grammatica G, l'analizzatore sintattico o parsificatore legge la stringa sorgente e, se appartiene al linguaggio L(G), ne produce una derivazione o un albero sintattico; altrimenti si ferma indicando la configurazione dove l'errore è comparso (diagnosi). Eventualmente prosegue l'analisi saltando le sottostringhe contaminate dall'errore (ripara l'effetto dell'errore).

Trascurando il trattamento degli errori ... un analizzatore è un riconoscitore arricchito della capacità di produrre la derivazione della stringa (l'automa, quando compie la mossa relativa alla regola grammaticale, deve salvare in una struttura dati l'etichetta che la identifica).

Se la stringa sorgente è ambigua il risultato è un insieme di alberi (o di derivazioni).

ANALISI DISCENDENTE E ASCENDENTE

Uno stesso albero corrisponde a diverse derivazioni (sinistra, destra, ...)

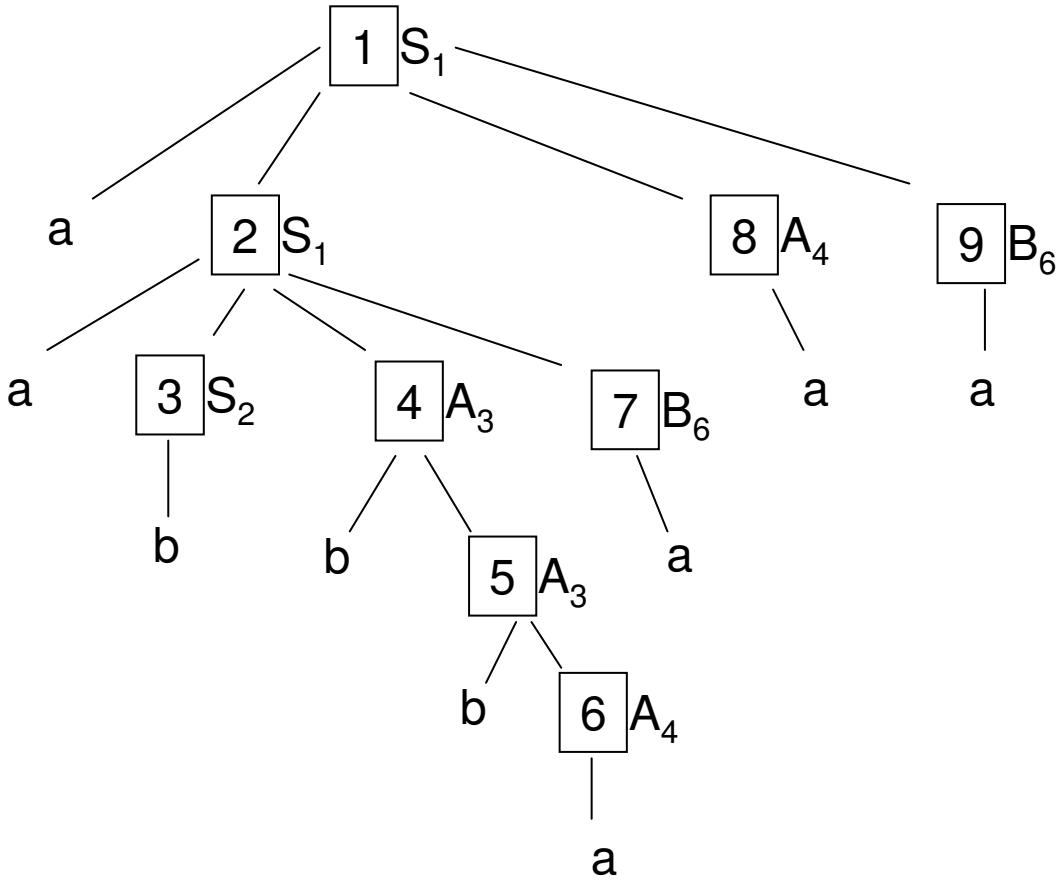
Due classi più importanti di analizzatori (dipendono dal fatto se si consideri una derivazione sinistra o una destra, e dall'ordine in cui essa è ricostruita)

ANALISI DISCENDENTE: costruisce la DERIVAZIONE SINISTRA, procedendo dall'assioma verso le foglie.

ANALISI ASCENDENTE: costruisce la derivazione ***destra*** della stringa in ordine riflesso (***dalle foglie alla radice*** dell'albero, attraverso una serie di ***riduzioni***).

ESEMPIO – ordini di visita dell'albero
frase: a *a b b b a a a a*

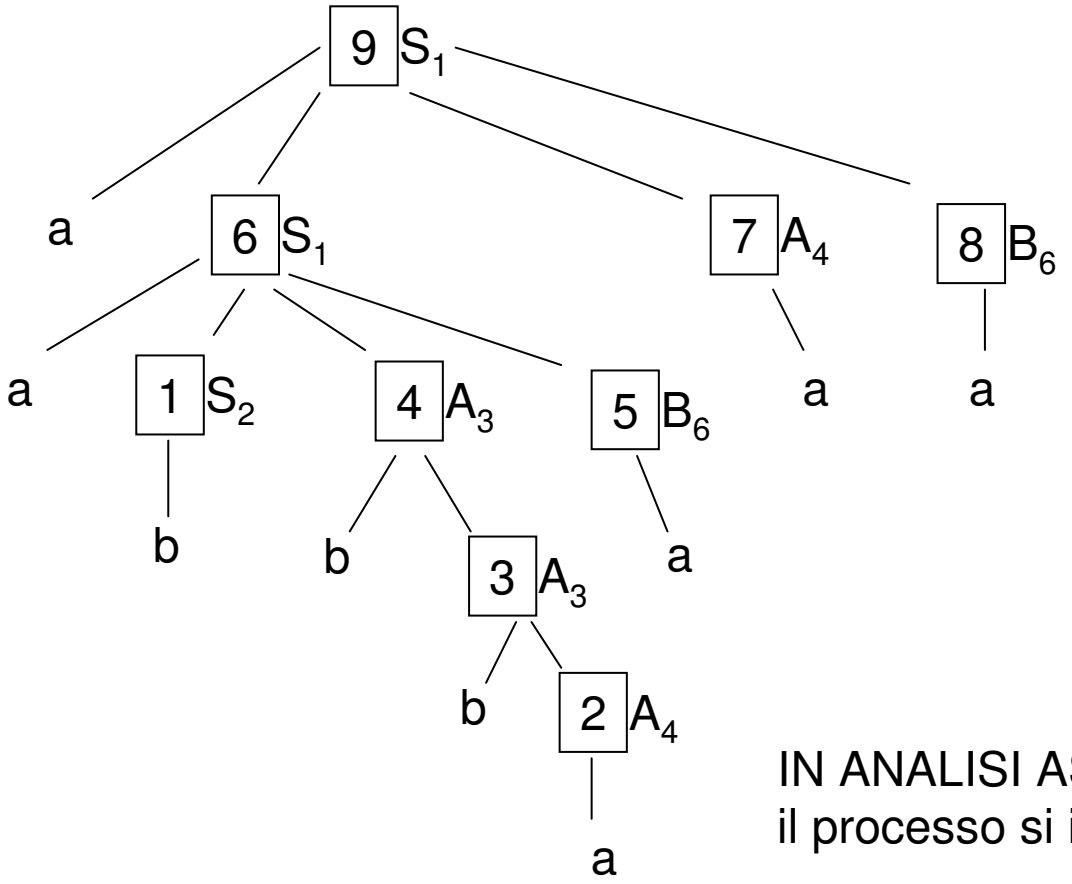
$1.\ S \to aSAB$	$2.\ S \to b$
$3.\ A \to bA$	$4.\ A \to a$
$5.\ B \to cB$	$6.\ B \to a$



ANALIZZ. DISCENDENTE: espande le parti sx delle regole con le corrispettive parti dx. Il processo termina quando tutti i simboli n.t. sono stati trasformati in simb. term. che combaciano con il testo (o in stringhe vuote).

ESEMPIO – ordini di visita dell'albero
frase: a *a b b b a a a a*

$1.\ S \to aSAB$	$2.\ S \to b$
$3.\ A \to bA$	$4.\ A \to a$
$5.\ B \to cB$	$6.\ B \to a$



ANALISI ASCENDENTE
Si riducono in un n.t. le parti dx delle regole, via via incontrate, prima nella stringa sorgente, poi nelle forme di frase da essa ottenute mediante le ***riduzioni***. Il processo termina quando l'intera stringa si riduce all'assioma.

IN ANALISI ASCENDENTE e DISCENDENTE
il processo si interrompe incontrando un errore.

LA GRAMMATICA COME RETE DI AUTOMI FINITI

Supponiamo G in forma estesa: ogni n.t. ha una sola regola

$A \rightarrow \alpha$ con α stringa di terminali e nonterminali.

α definisce un linguaggio regolare $\Rightarrow$ automa finito M_A, che riconosce α.

Transizione di M_A etichettata con n.t. B

interpretata come "chiamata" di automa M_B (se B=A ricorsione)

(chiamiamo - "macchine" gli automi finiti dei vari n.t.,
- "automa" quello a pila di tutto L(G))

TERMINOLOGIA

Σ alfabeto terminale, V={S, A, B, …} alfabeto nonterminale

$S \rightarrow \sigma$ $A \rightarrow \alpha$ $B \rightarrow \beta$ … regole della grammatica

R_S, R_A, R_B… linguaggi regolari in $\Sigma \cup V$ definiti da σ, α, β…

M_S, M_A, M_B, … macchine finite deterministiche che riconoscono R_S, R_A, R_B…

$\mathcal{M}$={M_S, M_A, M_B, …} rete di macchine

LA GRAMMATICA COME RETE DI AUTOMI FINITI - TERMINOLOGIA (segue)

Supponiamo stati delle macchine tutti distinti

$M_A\ Q_A\ q_{A,0}\ F_A$ macchna A, ins. stati, stato iniziale, ins. stati finali

$M_B\ Q_B\ q_{B,0}\ F_B$ …

…

$R(M_A,q)$ indica il linguaggio regolare $\subseteq(\Sigma\cup V)^*$ accettato da M_A partendo da stato q

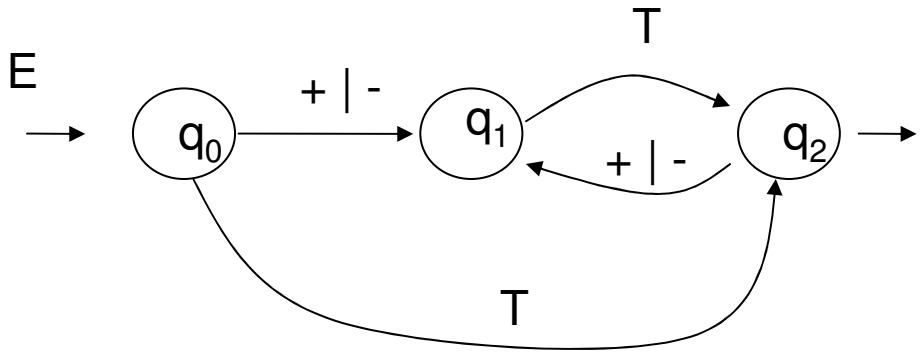
quindi $R(M_A,q_{A,0})=R_A$

Per il linguaggio **terminale**
$L(M_A,q)=\{y\in\Sigma^* \mid \eta\in R(M_A,q) \wedge \eta \Rightarrow^* y\}$ (in breve, L(q), se M_A chiara dal contesto)

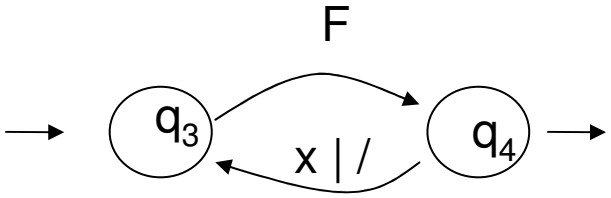
quindi $L(M_A,q_{A,0})=L_A(G)$
$L(M_S,q_{S,0})=L_S(G)=L(\mathcal{M})$

ESEMPIO – Espressioni aritmetiche

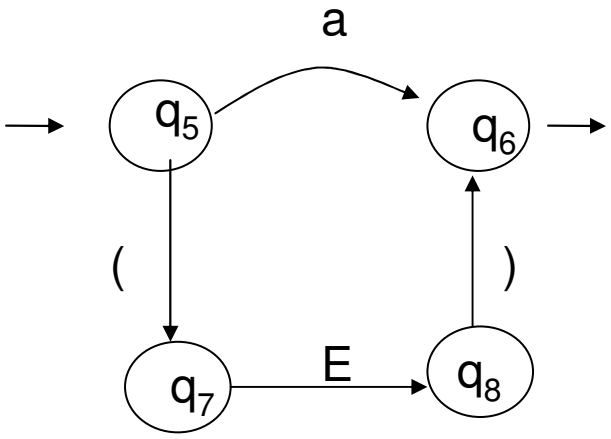
$$E \to [+|-]T((+|-)T)^*$$
$$T \to F((\times|/)F)^*$$
$$F \to (a|'('E')')$$



T:



F:



RICONOSCIMENTO

È immediato interpretare una rete di macchine ricorsive come lo schema di flusso dell'algoritmo ricorsivo di riconoscimento delle frasi del linguaggio.

L'algoritmo realizza un automa a pila, in generale non deterministico, con un solo stato interno, che accetta a pila vuota.

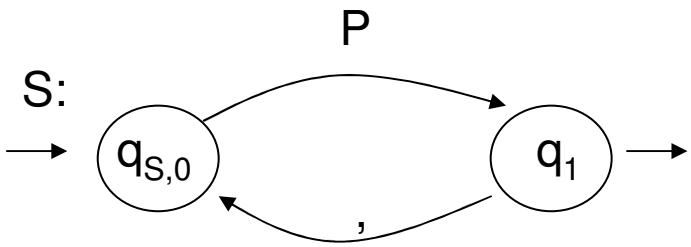
All'inizio la macchina attiva è quella dell'assioma che esamina il primo car.

1) Se la stringa è valida esisterà un calcolo che (ev. dopo aver invocato altre macc.) condurrà tale macchina in uno stato finale.
2) L'automa, a ogni passo, prosegue il calcolo all'interno della macchina corrente se il simbolo è terminale o (se il simbolo corrente denomina una macchina) salta al suo stato iniziale (può trattarsi della stessa macchina).
3) In caso di salto l'automa salva in pila lo stato della macchina sospesa.
4) Quando una macchina raggiunge lo stato finale l'automa ripristina lo stato della macchina che era stata sospesa per ultima, e da quel punto esegue la mossa che legge il nome della macchina terminata (come se il N.T. fosse una specie di ingresso).

Affinché la ricorsione termini è necessario che la grammatica non sia ricorsiva sinistra.

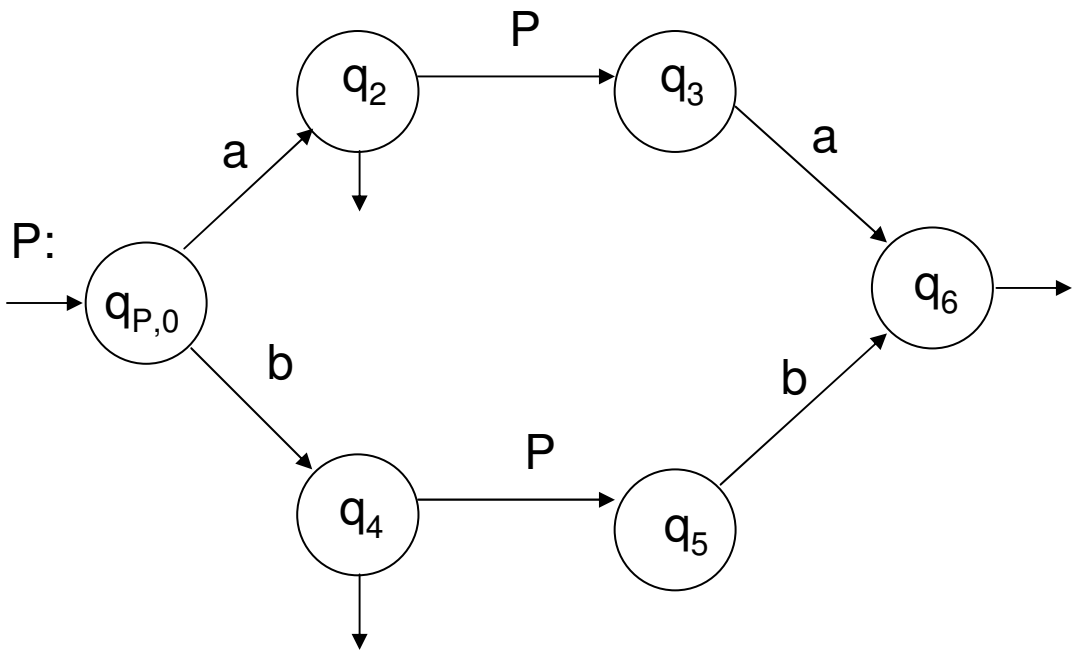
ESEMPIO – Lista di palindromi dispari

$$S \rightarrow P(,P)^*$$
$$P \rightarrow aPa \mid bPb \mid a \mid b \mid$$



ES: risocnoscimento della stringa a,b
(NB: a,b contiene **DUE** palindromi)

pila	stringa
$q_{S,0}$	a , b -|
$q_{S,0}$ $q_{P,0}$	a , b -|
$q_{S,0}$ q_2	, b -|
q_1	, b -|
$q_{S,0}$	b -|
$q_{S,0}$ $q_{P,0}$	b -|
$q_{S,0}$ q_4	-|
q_1	



ALGORITMO – Riconoscitore a discesa ricorsiva non deterministico
L'automa a pila è così definito:

begin

1. la stringa sorgente è *x* e *cc* il carattere corrente
2. i simboli della pila sono l'unione (disgiunta) $Q = Q_A \cup Q_B ...$ degli stati di tutte le mac.
3. l'automa a pila ha un solo stato interno, che si lascia sottinteso
4. inizialmente la pila contiene lo stato iniziale $q_{S,0}$ (macchina M_S dell'assioma)
5. transizioni; sia $s \in Q_A$ il simbolo in cima, cioè lo stato della macchina attiva M_A
   a) (mossa di scansione)
      se la mossa $\delta(s, cc) = s'$ è definita (nella macchina attiva M_A), ***consuma*** il carattere corrente e scrivi nella pila s' al posto di s
   b) (mossa di chiamata)
      se la mossa $\delta(s, B) = s'$ è definita per un nt B, effettua una mossa ***spontanea*** che depone sulla pila lo stato iniziale $q_{B,0}$ della macchina M_B, che così diviene attiva
   c) (mossa di ritorno) (NB: per l'automa a pila è una mossa spontanea)
      se s è uno stato finale di una macchina generica M_B cancellalo, effettua dallo stato r emerso sulla cima la mossa $\delta(r, B) = s'$ e scrivi sulla pila s' al posto di r
   d) (mossa di riconoscimento)
      se s è uno stato finale di M_S (la macchina dell'assioma) e cc = -|, accetta e termina
   e) in ogni altro caso rifiuta la stringa e termina

end

ANALISI SINTATTICA DISCENDENTE DETERMINISTICA

Molte grammatiche dei linguaggi artificiali sono progettate
per permettere un veloce riconoscimento deterministico.

Strada LL(k) (L = l'analisi esamina il testo da sinistra, L = la derivazione è sinistra,
k = lunghezza in caratteri della prospezione).

VEDREMO DUE VARIANTI IMPLEMENTATIVE DEL RICONOSCITORE:

1) automa a pila classico
2) il programma a discesa ricorsiva realizzato mediante la pila delle aree di attivazione dei sottoprogrammi

Analisi discendente LL(1)

Ci riferiamo a rete di automi per una grammatica: da $A \rightarrow \alpha$ deriva macchina M_A

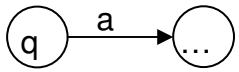
Definiamo

Insieme degli **INIZI** di uno **STATO** q, Ini(q): è definito come uguale a Ini(L(q))

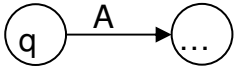
Insieme dei **SEGUITI** di un **NONTERMINALE** A

Seg(A) = {terminali che possono seguire A in una derivazione} (quindi $-| \in$ Seg(S))

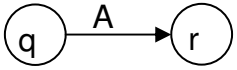
Algoritmo (in forma di clausole logiche) per calcolare Ini(q)

1. se (nella macchina) 

allora a $\in$ Ini(q)

2. se 

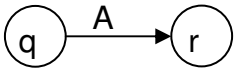
e $q_{A,0}$ stato iniziale di M_A allora Ini($q_{A,0}$) $\subseteq$ Ini(q)

3. se 

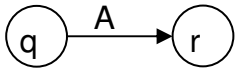
ed L($q_{A,0}$) è annullabile ($\varepsilon \in$ L($q_{A,0}$)) allora Ini(r) $\subseteq$ Ini(q)

Algoritmo (in forma di clausole logiche) per calcolare Seg(A). $A \in V$

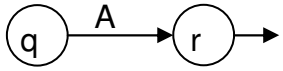
1. $\dashv \in Seg(S)$

2. se 

allora $Ini(r) \subseteq Seg(A)$

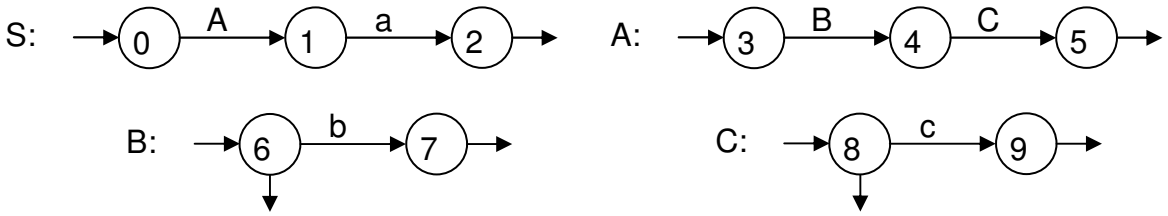
3. se in M_B 

, $B \neq A$ e $\varepsilon \in L(r)$ allora $Seg(B) \subseteq Seg(A)$

4. se 

(r finale in M_B, $B \neq A$), allora $Seg(B) \subseteq Seg(A)$

ESEMPIO: Grammatica $S \rightarrow Aa$ $A \rightarrow BC$ $B \rightarrow b|\varepsilon$ $C \rightarrow c|\varepsilon$

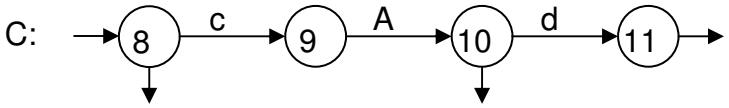


NB: A, B, C
sono annullabili

$Ini(0) = Ini(3) \cup Ini(1)$
$= Ini(6) \cup Ini(4) \cup \{a\}$
$= Ini(6) \cup Ini(8) \cup Ini(5) \cup \{a\}$
$= \{b\} \cup \{c\} \cup \varnothing \cup \{a\}$

$Seg(S) = \{-|\}$
$Seg(A) = Ini(1) = \{a\}$
$Seg(B) = Ini(4) \cup Seg(A)$
$= Ini(8) \cup Seg(A)$
$= \{c\} \cup \{a\}$
$Seg(C) = Seg(A) = \{a\}$

Se un nonterm. compare piu` volte nella rete i seguiti di ogni comparsa vanno uniti
Esempio: modifichiamo M_C e ricalcoliamo Seg(A)



$Seg(A) = Ini(1) \cup Ini(10) \cup Seg(C)$
$= \{a\} \cup \{d\} \cup Seg(A)$
$= \{a\} \cup \{d\}$

Insieme Guida:

per ogni ramo di un bivio dà l'insieme dei caratteri incontrati seguendolo

se in un bivio insiemi guida disgiunti il parsificatore può scegliere a colpo sicuro

$Gui(q\rightarrow\ldots) \subseteq \Sigma \cup \{\text{-}|\}$ definito per casi per tutte le frecce (mosse o accettazione)

1. $Gui(q \xrightarrow{b} r) = \{b\}$
2. $Gui(q \xrightarrow{A} r) = Ini(L(q_{A,0})L(r))$ se $\varepsilon \notin L(q_{A,0})L(r)$
3. se $q \xrightarrow{A} r$ in M_B e $\varepsilon \in L(q_{A,0})L(r)$ allora $Gui(q \xrightarrow{A} r) = Ini(L(q_{A,0})L(r)) \cup Seg(B)$
4. se $q \rightarrow$ (q stato finale) in M_B allora $Gui(q \rightarrow) = Seg(B)$

CONDIZIONE SUFFICIENTE AFFINCHÉ L'AUTOMA A PILA DELL'ALGORITMO PRECEDENTE SIA DETERMINISTICO:

CONDIZIONE LL(1) – Uno stato $q \in Q_A$ di una macchina M_A (per ipotesi deterministica) soddisfa la condizione LL(1) se, per ogni coppia di frecce uscenti da esso, gli insiemi guida sono disgiunti.

DEFINIZIONE – Una grammatica è LL(1) se ogni stato delle macchine della rete è LL(1).

ESEMPI – Grammatiche LL(1) e non

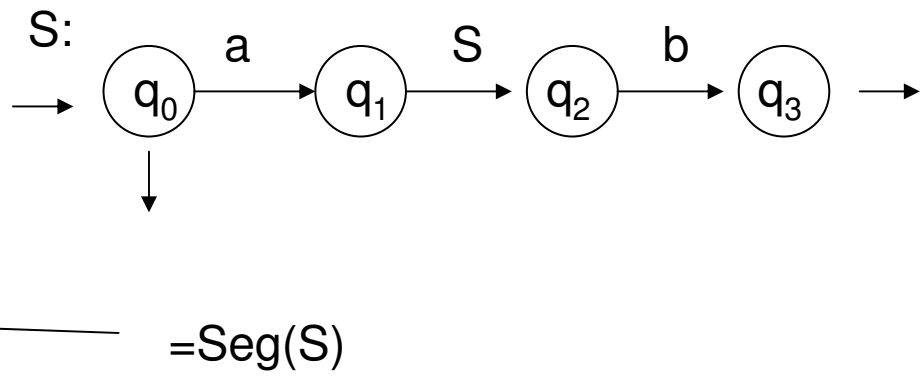
$G_1 = S \to aSb \mid \varepsilon$

$\{a^n b^n \mid n \geq 0\}$

$Gui(q_0 \xrightarrow{a} q_1) = \{a\}$

$Gui(q_0 \to) = \{b, \text{-|}\}$

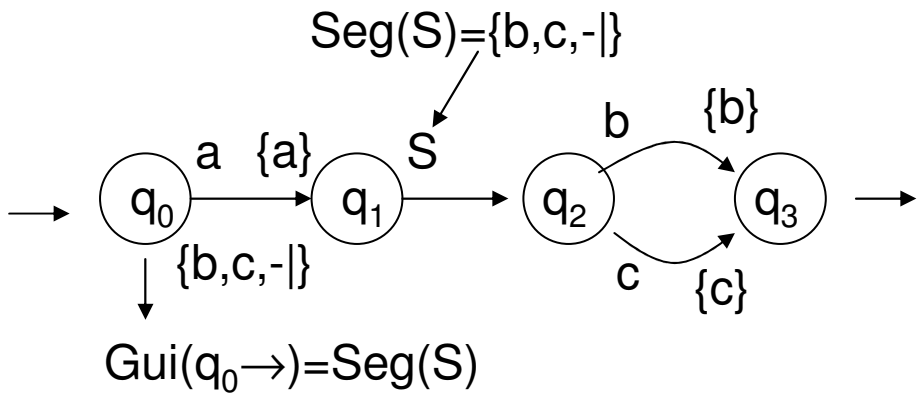
disgiunti: la gramm. è LL(1)



$G_2 : \; S \to aSb \mid aSc \mid \varepsilon$

$\left\{ a^n (b \mid c)^n \mid n \geq 0 \right\}$

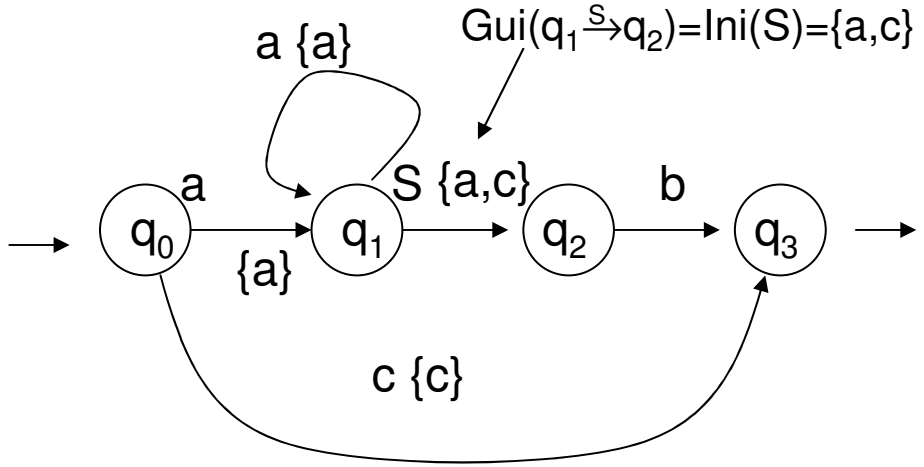
G_2 risulta $LL(1)$



$G_3 : \; S \to a^+ Sb \mid c$

$\left\{ a^* a^n c b^n \mid n \geq 0 \right\}$

G_3 viola la condizione $LL(1)$ in q_1



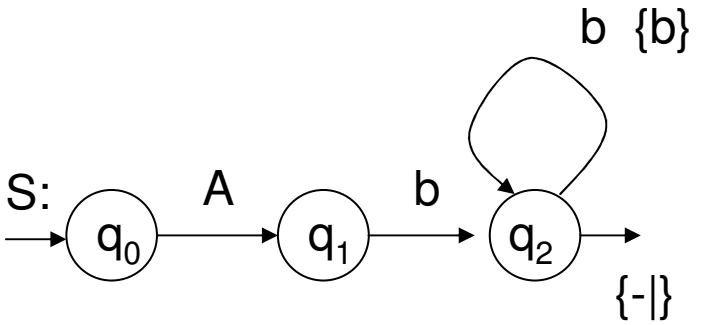
$$G_4: \quad S \to Ab^+$$

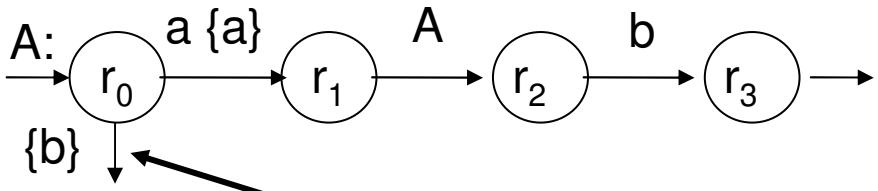
$$A \to aAb \mid \varepsilon$$

$$\left\{a^n b^n b^+ \mid n \geq 0\right\}$$

la condizione LL(1)

è rispettata





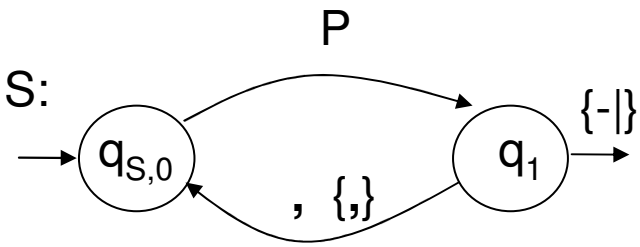
Gui($r_0 \to$)=Seg(A)={b}

Seg(A)=Ini(q_1)∪Ini(r_2)={b}

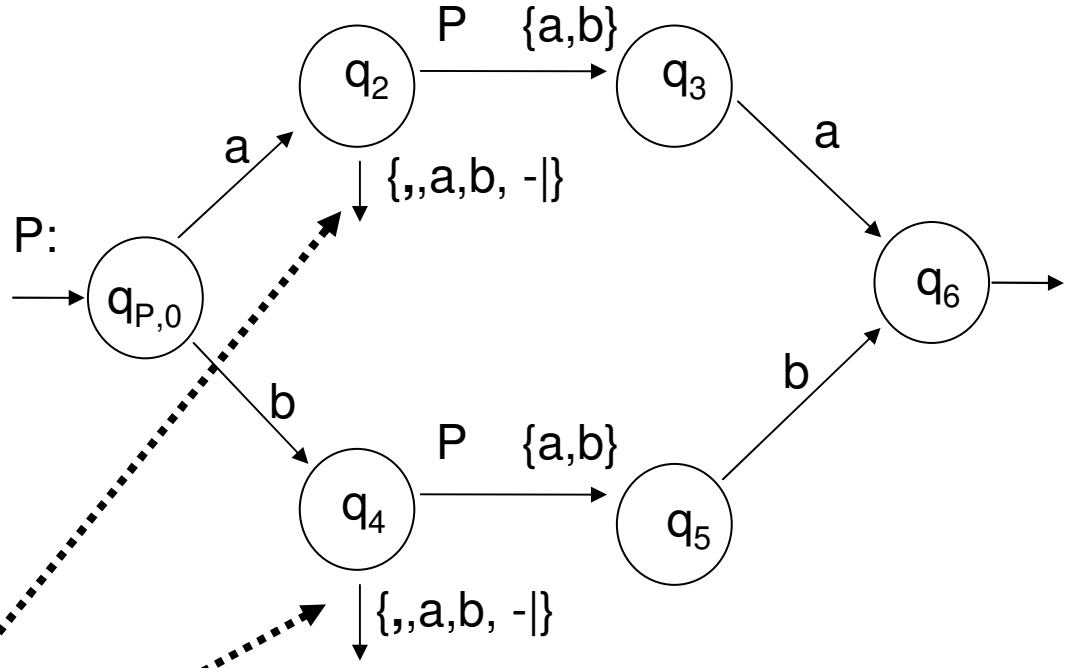
$S \rightarrow P(,P)^*$

$P \rightarrow aPa \mid bPb \mid a \mid b \mid$

la condizione LL(1) non è rispettata in q_2 e q_4



$Gui(q_2 \xrightarrow{P} q_3) = Ini(L(q_{P,0})) = \{a,b\}$



Seg(P)={,,a,b, -|}

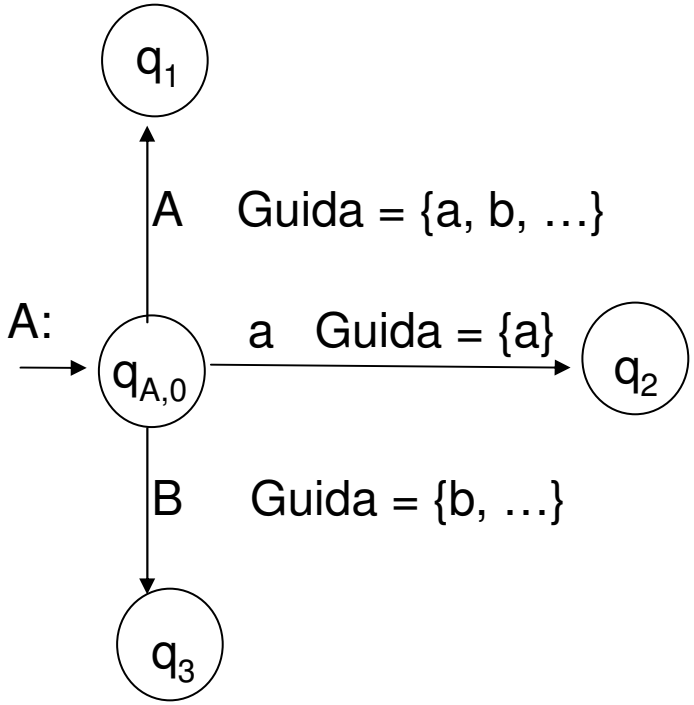
Gui(q_2→)= Gui(q_4→)=Seg(P)={,,a,b, -|}

PROPRIETÀ: una regola ricorsiva a sinistra non è LL(1) – L'insieme guida della mossa ricorsiva a sinistra contiene l'unione degli altri insiemi guida.

esempio:

$$A \rightarrow A.... \mid a.... \mid B$$

$$B \rightarrow b....$$



PROPRIETÀ – Se la grammatica è LL(1) l'algoritmo di riconoscimento illustrato diventa deterministico.

Infatti nell'algoritmo che segue il fatto che gli insiemi guida delle frecce uscenti dallo stato *s* siano per ipotesi disgiunti, rende - al passo 5 - le condizioni *a*, *b*, c, *d* ed *e* mutuamente esclusive.

ALGORITMO – Ric. a discesa ricorsiva det. - L'automa a pila è così definito:
begin

1. la stringa sorgente è *x* e *cc* il carattere corrente
2. i simboli della pila sono l'unione (disgiunta) $Q = Q_A \cup Q_B ...$ degli stati di tutte le mac.
3. l'automa a pila ha un solo stato interno, che si lascia sottinteso
4. inizialmente la pila contiene lo stato iniziale $q_{S,0}$ (della mac. M_S ass. all'assioma)
5. transizioni: sia $s \in Q_A$ il simbolo in cima, cioè lo stato della macchina attiva M_A
   a) (mossa di scansione) se la mossa $\delta(s, cc) = s'$ è definita (nella macchina attiva M_A), consuma il carattere corrente e scrivi nella pila s' al posto di s
   b) (mossa di chiamata) se la mossa $\delta(s, B) = s'$ è definita per un n.t. B, **e cc∈ Gui(s$\stackrel{B}{\rightarrow}$s')** effettua una mossa spontanea che depone sulla pila lo stato iniziale $q_{B,0}$ della macchina M_B, che così diviene quella attiva
   c) (mossa di ritorno) se *s* è uno stato finale di una macchina generica M_B, **e cc∈ Gui(s→)** cancellalo, poi effettua dallo stato *r* emerso sulla cima la mossa spontanea $\delta(r, B) = s'$ e infine scrivi sulla pila s' al posto di *r*
   d) (mossa di riconoscimento) se *s* è uno stato finale di M_S (la macchina dell'assioma) e *cc* = -|, accetta e termina
   e) in ogni altro caso rifiuta la stringa e termina

end

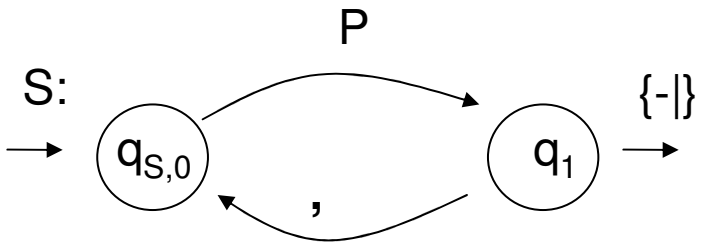
NB: - le condizioni a), b), c), d) sono mutuamente esclusive
- la complessita` del calcolo è $\Theta(|x|)$ (mosse spontanee limitate da grammatica)

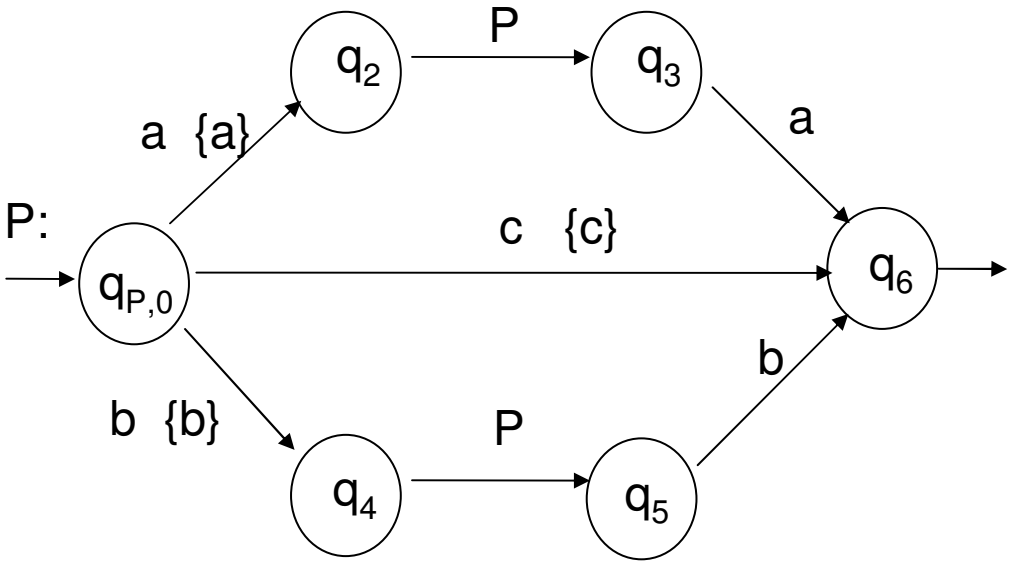
IMPLEMENTAZIONE DEL PARSIFICATORE MEDIANTE PROCEDURE RICORSIVE

Il parsificatore viene realizzato come insieme di procedure che corrispondono alle macchine della grammatica.
Ogni procedura ha lo schema a blocchi che corrisponde alla macchina associata a quel n.t.

$$S \rightarrow P(,P)^*$$
$$P \rightarrow aPa \mid bPb \mid c$$

ESEMPIO





IMPLEMENTAZIONE DEL PARSIFICATORE MEDIANTE PROCEDURE RICORSIVE

```
procedure S
begin
1.  call P;
2.  while (not cc = '-|') do
        if cc = ',' then
            cc := Prossimo;
            call P;
        else Errore;
    end while
3.  if cc = '-|' then accetta e termina;
    else Errore;
end
```

```
procedure P
begin
1.  if cc = 'a' then
        cc := Prossimo;
        call P;
        if cc = 'a' then
            cc := Prossimo
        else Errore;
2.  else if cc = 'b' then
        cc := Prossimo;
        call P;
        if cc = 'b' then
            cc := Prossimo;
        else Errore;
3.  else if cc = 'c' then
        cc := Prossimo;
4.  else Errore;
end
```

OSSERVAZIONI

NEL PARSIFICATORE IMPLEMENTATO:

1) La funzione Prossimo realizza la scansione lessicale.
2) Se l'arco è non terminale viene invocata la corrispondente procedura.
3) Se il carattere non sta in nessun insieme guida si ha un errore e la stringa sorgente è rifiutata.
4) In caso di errore questo parsificatore termina subito senza fornire spiegazioni, mentre nella pratica è richiesto un messaggio diagnostico.
5) Confrontando i caratteri attesi in un certo stato (quelli appartenenti agli insiemi guida delle frecce uscenti) con il carattere corrente è possibile produrre una semplice diagnostica (i caratteri attesi sono .... invece di ....).
6) Un buon analizzatore deve essere in grado di proseguire il calcolo dopo il primo errore, al fine di esaminare l'intera stringa sorgente con una sola compilazione, anche se essa contiene degli errori.

ALLUNGAMENTO DELLA PROSPEZIONE

Per ottenere un parsificatore deterministico, quando la condizione LL(1) cade, si esamina non solo il carattere corrente, ma anche quello successivo (l'algoritmo applica una prospezione di lunghezza 2). Se così facendo la scelta della mossa da eseguire nello stato considerato è unica, si dice che lo stato soddisfa la condizione LL(2).

Se si estende la lunghezza della prospezione a qualsiasi valore finito $k \geq 1$, si ottiene la definizione della famiglia delle grammatiche LL(k).

In pratica i parsificatori che operano con il metodo LL(k) usano lunghezze diverse di prospezione: in ogni stato si usa la minima lunghezza necessaria a risolvere la scelta tra mosse uscenti dallo stato.

Sono stati sviluppati parsificatori che - nel caso non basti la prospezione di lunghezza finita - eseguono un'esplorazione in avanti sulla stringa sorgente fino a incontrare un terminale che permetta di fare la scelta tra le mosse.

ESEMPIO: per distinguere l'id di una label dall'id di una variabile a sinistra di un assegnamento occorre guardare il token dopo l'id: se è un ':' è una label, se è un '=' è il nome di una variabile assegnata

$progr \rightarrow [label :] stat(;stat)^*$ $\quad stat \rightarrow assign\_stat \mid for\_stat \mid ....$

$assign\_stat \rightarrow id = expr$ $\quad for\_stat \rightarrow for...$

$label \rightarrow id$ $\quad expr \rightarrow ...$