Analisi sintattica: metodo di Early

Prof. A. Morzenti

aa 2008-2009

UN METODO GENERALE DI ANALISI SINTATTICA: EARLY

Il METODO DI EARLY tratta qualsiasi grammatica libera

Costruisce tutte le derivazioni delle frasi ambigue.
Complessità: cubo lunghezza stringa, si riduce se grammatica non è ambigua,
ulteriormente se deterministica.

Early simile a LR(k), ma non usa pila
invece usa vettore di insiemi:
rappresenta efficientemente più pile con parti comuni

simula automa a pila indeterministico, ma senza complessità esponenziale

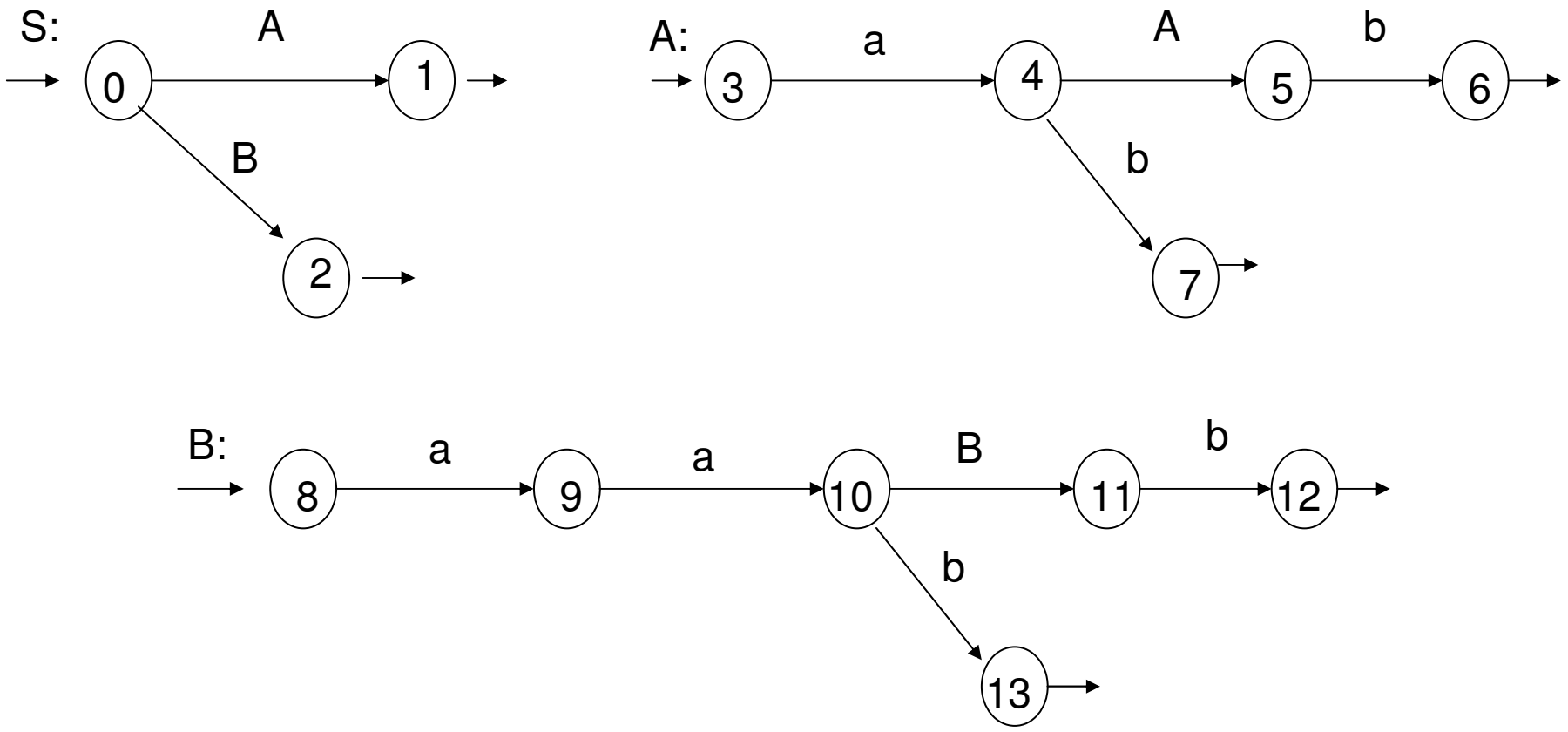
NB: anche metodo LR simula piu` calcoli in parallelo, ma solo fino alla riduzione.

IPOTESI INIZIALE: LA GRAMMATICA È PRIVA DI REGOLE VUOTE
L'algoritmo con o senza prospezione senza modifica della complessità
Per semplicità non usiamo prospezione.

ESEMPIO - grammatica non LR(k)

$$L = \{a^n b^n \mid n \geq 1\} \cup \{a^{2n} b^n \mid n \geq 1\}$$

$$S \to A \mid B \qquad A \to aAb \mid ab \qquad B \to aaBb \mid aab$$



analisi della stringa: aabb

$$L = \{a^n b^n \mid n \geq 1\} \cup \{a^{2n} b^n \mid n \geq 1\}$$

$$S \to A \mid B \quad A \to aAb \mid ab \quad B \to aaBb \mid aab$$

E[0]

(0≡S→●(A|B),p=0)
(3≡A→●aAb|●ab,p=0)
(8≡B →●aaBb|●aab,p=0)

a E[1]

(4≡A→a●Ab|a●b,p=0)
(9≡B →a●aBb|a●ab,p=0)
(3≡A→●aAb|●ab,p=1)

a E[2]

(10≡B →aa●Bb|aa●b,p=0)
(4≡A→a●Ab|a●b,p=1)
(8≡B →●aaBb|●aab,p=2)
(3≡A→●aAb|●ab,p=2)

b E[3]

(13≡B →aab●,p=0)
(7≡A→ab●,p=1)
(2≡S→B●,p=0)
(5≡A→aA●b,p=0)

perchè in 0 c'è (S→ ●B, 0)

perchè in 1 c'è (A→ a●Ab, 0)

b E[4]

(6≡A→aAb●,p=0)
(1≡S→A●,p=0)

perchè in 0 c'è (S→ ●A, 0)

Se la stringa in analisi fosse *aab,* verrebbe accettata per la presenza in E[3] della coppia (s = 2,p = 0). Ma la stringa in analisi è *aabb* che viene accettata perché in E[4] figura la coppia (s = 1,p = 0)

L'algoritmo è un analizzatore discendente che sviluppa simultaneamente tutte le possibili derivazioni sinistre della stringa. L'algoritmo legge la stringa $x_{1...n}$ da sinistra a destra e, quando esamina il carattere x_i, produce strutture a due campi (coppie) aventi la forma:

<stato della rete, puntatore> scritte come *(s , p = …)* con $0 \le p \le i$

Lo stato può essere scritto come regola grammaticale marcata $A \rightarrow \alpha \bullet \beta$

Intuitivamente la coppia $(s \equiv A \rightarrow \alpha \bullet \beta, j)$ rappresenta un'asserzione e un obiettivo

Asserzione: è stata trovata una sottostringa $x_{j+1...i}$ con $1 \le j \le i$ che deriva da α

Obiettivo: trovare tutte le posizioni k tali che la sottostringa $x_{i+1...k}$ derivi da β

Se l'algoritmo trova tale posizione k, può asserire che dal nonterminale A deriva la sottostringa $x_{j+1...k}$

$$\alpha \overset{*}{\Rightarrow} x_{j+1...i}$$

$$\beta \overset{*}{\Rightarrow} x_{i+1...k}$$

$$A \overset{*}{\Rightarrow} x_{j+1...k}$$

Una coppia $(q \equiv A \rightarrow \alpha \bullet, j)$ in cui lo stato q è finale (marca in fondo), è detta completata

ALGORITMO – Riconoscitore di Early

Il compito del riconoscitore è trovare la derivazione dell'intera stringa *x*, ma l'algoritmo produce più di quanto richiesto: per ogni prefisso proprio della stringa, dice se esso appartiene al linguaggio. Es.: in E[3] c'è (2≡S→B●,p=0), dice che aab∈L

L'algoritmo costruisce un vettore E[0…n] dimensionato sulla lunghezza della stringa sorgente i cui elementi sono insiemi di coppie

PASSO 0: *Inizializzazione* - predispone gli obiettivi per trovare ogni prefisso di *x* derivabile dall'assioma S. L'insieme iniziale è riempito con coppie ricavate dall'assioma; tutti gli altri insiemi sono inizialmente vuoti.

E[0] := {(q_α,0) | α e` lo stato iniziale della rete}

E[i] := ∅; per ogni i = 1, …, n

i := 0;

Poi si applicano nell'ordine naturale *0, 1, …, n* le operazioni di *predizione, completamento e scansione* per calcolare tutti gli insiemi E[i].

L'algoritmo al passo *i* può aggiungere elementi solo all'insieme corrente E[i] e a quello successivo E[i + 1].

Se nessuna delle operazioni ha aggiunto nuove coppie a E[i], si passa al calcolo di E[i + 1].

L'algoritmo termina prematuramente e rifiuta la stringa se E[i + 1] è vuoto e $i < n$

L'algoritmo termina con successo e accetta la stringa quando l'insieme E[n] è stato completato e contiene (almeno) una coppia completata ($q_\omega \equiv S \rightarrow \alpha \bullet, 0$) dell'assioma.

OPERAZIONE DI PREDIZIONE (analoga a CHIUSURA)

Ogni obiettivo presente nell'insieme E[i] può aggiungere altri sotto obiettivi all'insieme stesso; al puntatore è assegnato l'insieme corrente.

per ogni coppia *(q ≡ A→α ●Bγ, j)* presente in E[i],
aggiungi all'insieme E[i] la coppia *(r, i)*
dove *r* è lo stato iniziale della macchina B

OPERAZIONE DI SCANSIONE (analoga a SPOSTAMENTO)

Aggiorna gli obiettivi dell'insieme E[i + 1] in funzione del prossimo carattere sorgente. Il puntatore è posto eguale a quello della coppia esaminata.

per ogni coppia *(q ≡ A→α ●aγ, j)* presente in E[i],
se a = x_{i+1}
aggiungi la coppia *(r ≡ A→α a●γ, j)* all'insieme E[i + 1]

Ciascun carattere x_{i+1} della stringa è esaminato dall'algoritmo una sola volta nella scansione, la testina di lettura non torna mai indietro

OPERAZIONE DI COMPLETAMENTO (analoga a RIDUZIONE)

Una coppia completata *(q ≡ A→α ●, j)* asserisce che è stata trovata la derivazione della stringa $x_{j+1\ldots i}$ dal nonterminale A. Occorre aggiornare l'insieme E[i] con tale asserzione.

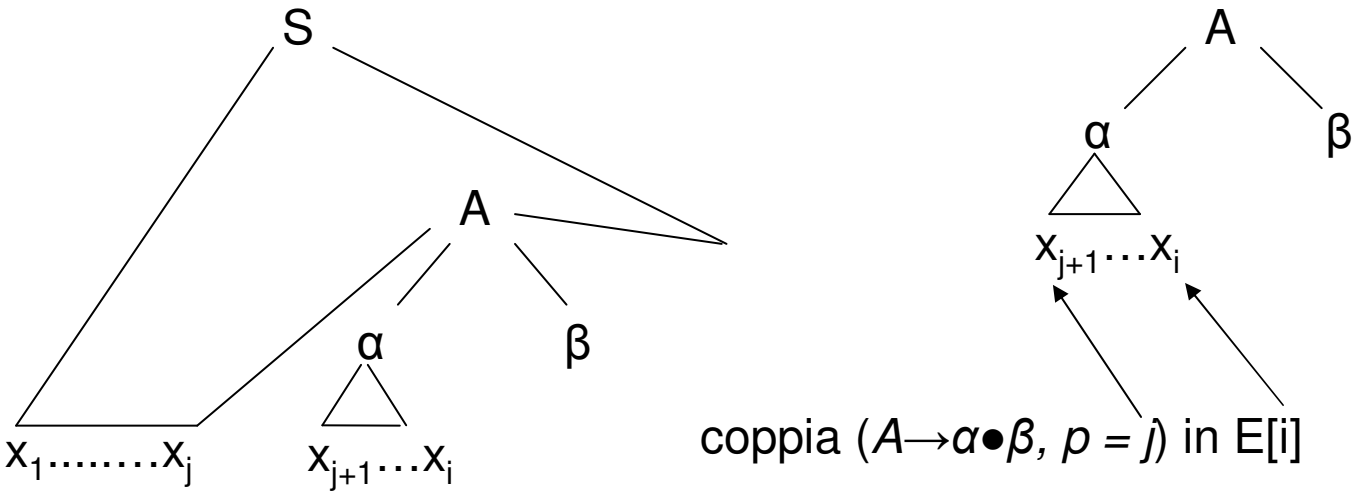
per ogni coppia completata *(q ≡ A→α ●, j)* presente in E [i],
per ogni coppia *(r ≡ B→β ●Aγ, k)* presente in E [j],
aggiungi a E[i] la coppia *(s ≡ B→βA●γ, k)*

Complessità quadratica,
=> algoritmo cubico

Nel completamento, se nella coppia *(r ≡ B→β ●Aγ, k)* ∈ E[j] la stringa *γ* è vuota, la nuova coppia *(s ≡ B→βA●, k)* aggiunta a E[i] è completata, quindi occorre iterare di nuovo.

È facile mostrare che l'algoritmo accetta solo stringhe appartenenti al linguaggio L(G): una coppia viene aggiunta all'insieme solo se la derivazione da essa asserita è possibile. Più complessa è la dimostrazione che ogni frase del linguaggio è riconosciuta dall'algoritmo.

LA COSTRUZIONE PROGRESSIVA DELL'ALBERO SINTATTICO



È significativo ripensare le tre operazioni come costruzione progressiva di alberi sintattici: a ogni insieme corrisponde un insieme di alberi.

In E[i] vi è la coppia ($A \rightarrow \alpha \bullet \beta$, $p = j$) se e solo se, per la grammatica esiste un albero sintattico della forma presentata in figura. L'albero di destra mostra l'albero associato alla coppia considerata.

Al termine dell'algoritmo, la stringa è accettata se, e solo se, tra gli alberi associati all'ultimo insieme (E[n]) vi è un albero sintattico completo con radice nell'assioma.

E[0]

(0≡S→●(A|B),p=0)
(3≡A→●aAb|●ab,p=0)
(8≡B →●aaBb|●aab,p=0)

a E[1]

(4≡A→a●Ab|a●b,p=0)
(9≡B →a●aBb|a●ab,p=0)
(3≡A→●aAb|●ab,p=1)

a E[2]

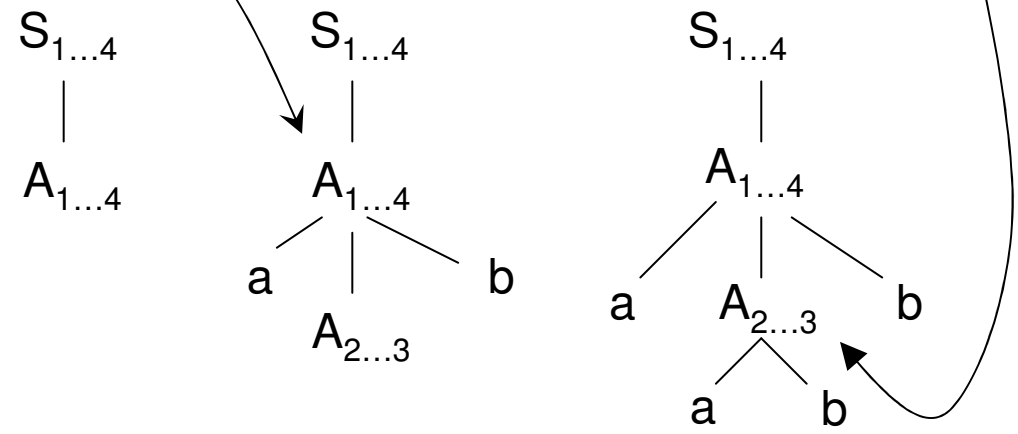
(10≡B →aa●Bb|aa●b,p=0)
(4≡A→a●Ab|a●b,p=1)
(8≡B →●aaBb|●aab,p=2)
(3≡A→●aAb|●ab,p=2)

b E[3]

(13≡B →aab●,p=0)
(7≡A→ab●,p=1)
(2≡S→B●,p=0)
(5≡A→aA●b,p=0)

b E[4]

(6≡A→aAb●,p=0)
(1≡S→A●,p=0)



Se la grammatica è ambigua, l'analisi di una frase produce tutti gli alberi possibili, rappresentati in modo fattorizzato.

ESEMPIO – parsificazione di un linguaggio ambiguo – La grammatica e ricorsiva bilateralmente, quindi ambigua. Analizzata stringa a + a + a

$$S \to E \qquad E \to E + E \qquad E \to a$$

E[0]

S→●E,0
E→●E+E| ●a,0

a E[1]

E→a●,0

S→E●,0
E→E●+E,0

+ E[2]

E→E+●E,0
E→●E+E| ●a,2

a E[3]

E→a●,2

E→E+E●,0
E→E●+E,2

S→E●,0
E→E●+E,0

+ E[4]

E→E+●E,0
E→E+●E,2
E→●E+E| ●a,4

a E[5]

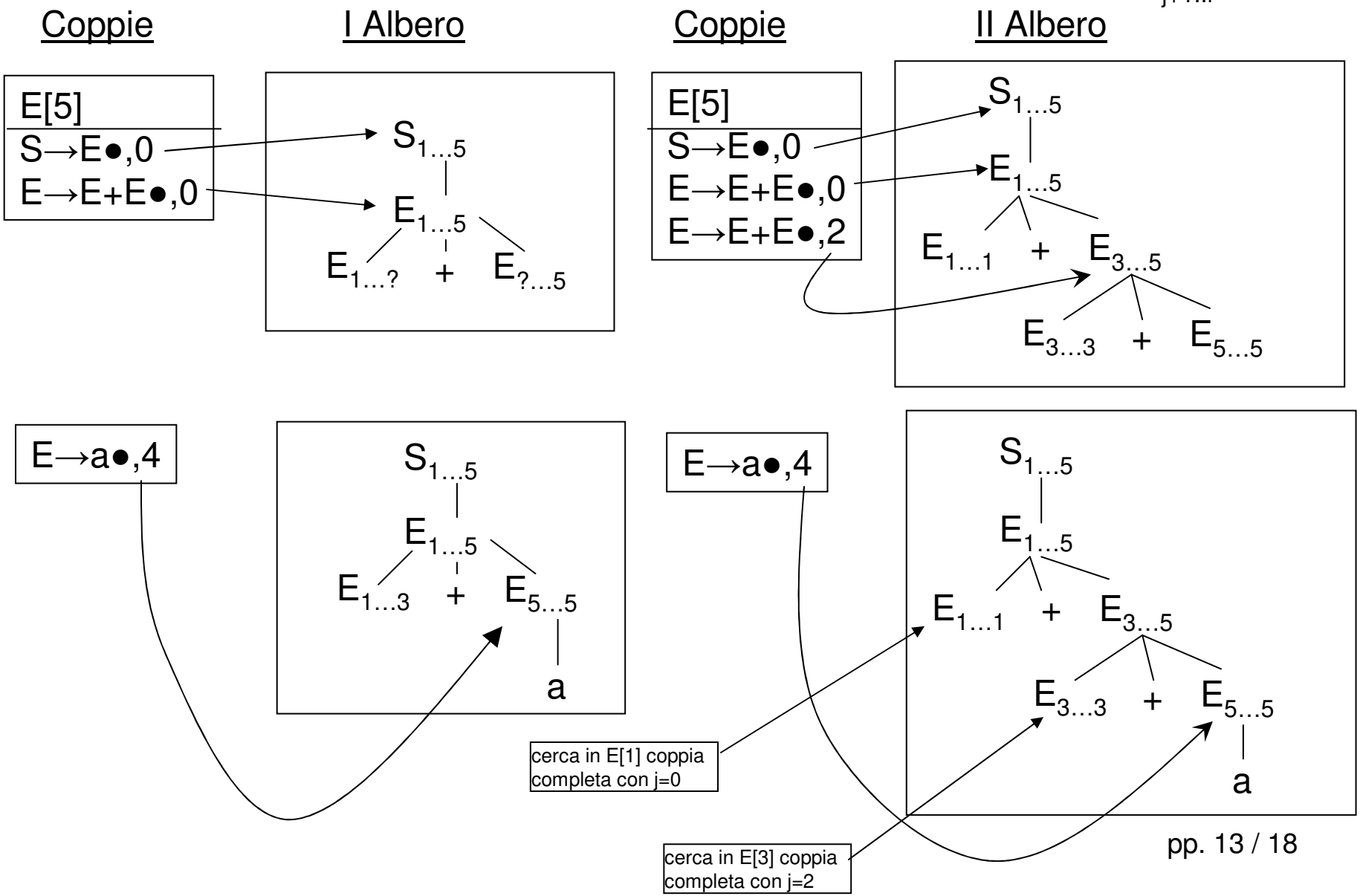
E→a●,4

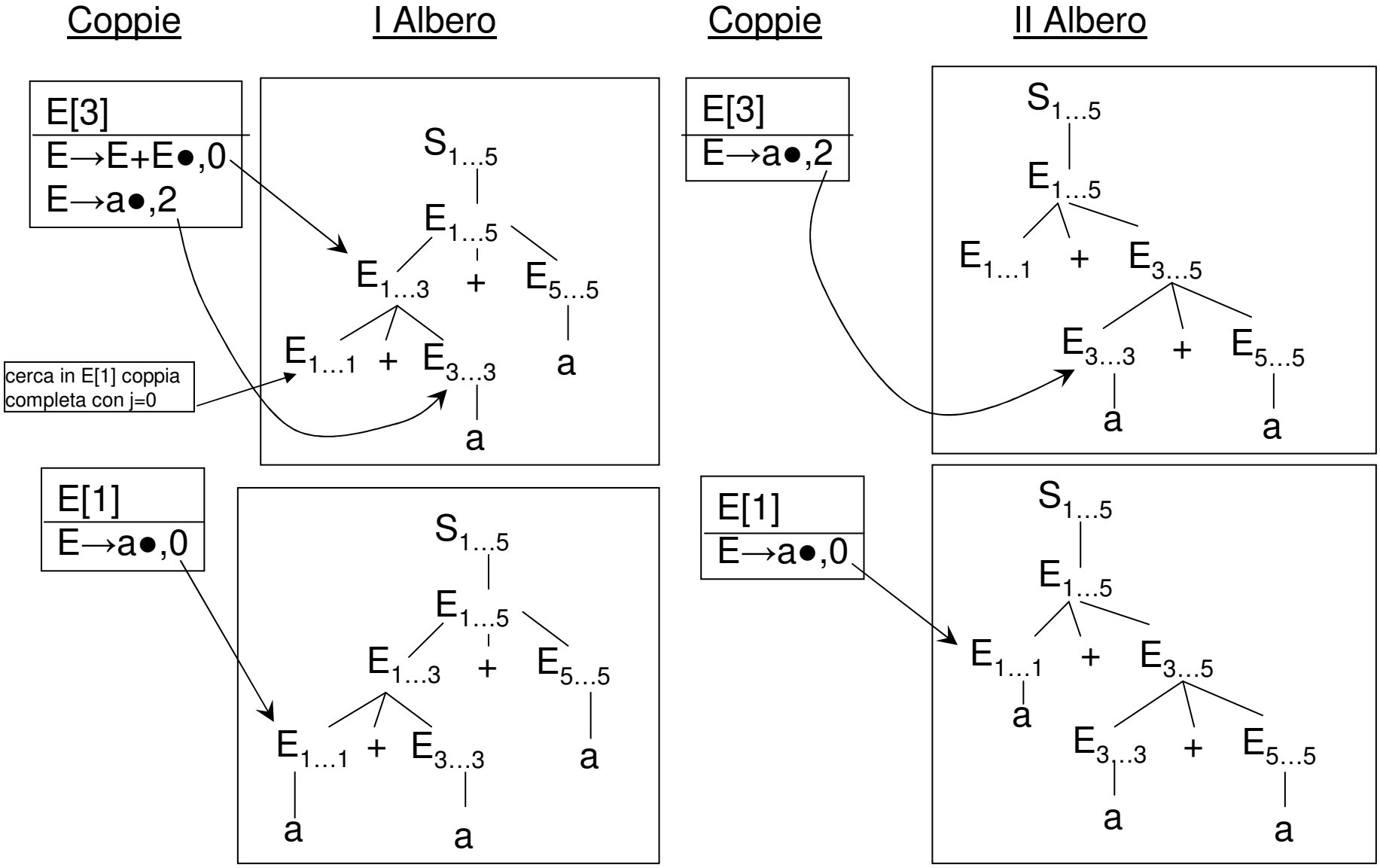
E→E+E●,2
E→E●+E,4
E→E+E●,0

E→E●+E,2
S→E●,0
E→E●+E,0

NB: linea orizzonta separa successive iterazioni del completamento

NB: se in E[i] c'è (A→ α, j) allora albero $A_{j+1..i}$ con $\alpha_{j+1..i}$




Coppie
I Albero
Coppie
II Albero
E[3]
E→E+E●,0
E→a●,2
cerca in E[1] coppia completa con j=0
E[1]
E→a●,0
S1…5
E1…5
E1…3
+
E5…5
E1…1
+
E3…3
a
E3…5

TRATTAMENTO DELLE REGOLE VUOTE

L'algoritmo presentato, costruisce due insiemi di coppie: E[i] che riceve coppie dai passi di predizione e completamento e l'insieme E[i + 1] che riceve coppie dai passi di scansione.

In presenza di ε-regole il *completamento* dovrebbe esaminare l'insieme E[i] parzialmente costruito, poi chiamare la *predizione*, la quale dovrebbe riattivare il *completamento,* e così via fino a quando nessuna delle due operazioni ha più nulla da aggiungere all'insieme. Questo metodo risulta corretto ma inefficiente.

Il metodo di *Aycock e Horspool* risulta più efficiente e modifica così l'operazione di predizione:

OPERAZIONE DI PREDIZIONE (con ε-regole)

per ogni coppia *(q ≡ A→α ●Bγ, j)* presente in E[i],
aggiungi all'insieme E[i] la coppia *(r ≡ B→●δ, i)*
dove *r* è lo stato iniziale della macchina B
se B è annullabile
aggiungi all'insieme E[i] anche le coppie *(s ≡ A→αB●γ, j)*

Il passo sposta la marca a destra di un simbolo nonterminale, se da esso può derivare la stringa vuota, in accordo con il fatto che la derivazione farebbe sparire il nonterminale stesso

ESEMPIO – Grammatica con tutti i nonterminali annullabili: ALTAMENTE AMBIGUA

$$S' \to S \quad S \to AAAA \quad A \to a \quad A \to E \quad E \to \varepsilon$$

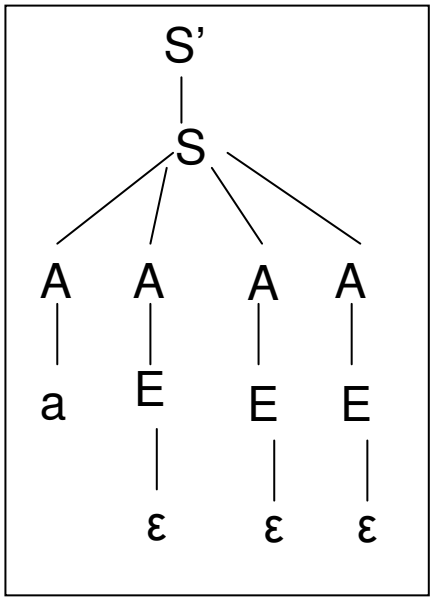
Traccia del riconoscimento della stringa *a*

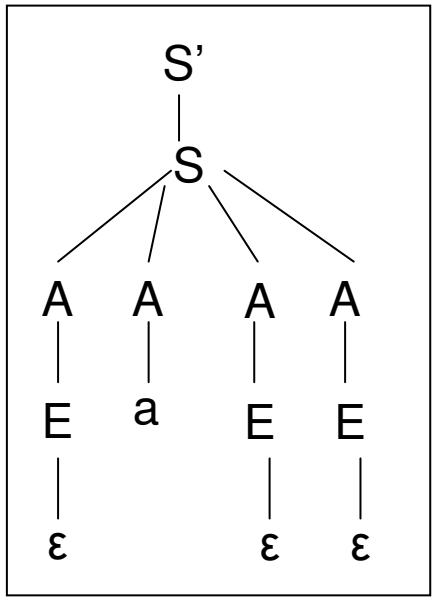
E[0]

S’→●S,0
S→●AAAA,0
S’→S●,0
A→●a,0
A→●E,0
S→A●AAA,0
E→●,0
A→E●,0
S→AA●AA,0
S→AAA●A,0
S→AAAA●,0

aE[1]

A→a●,0
S→A●AAA,0
S→AA●AA,0
S→AAA●A,0
S→AAAA●,0
A→●a,1
A→●E,1
S’ →S●,0
E→●,1
A→E●,1





eccetera …..

SVILUPPI DEL METODO DI EARLY

1) USO DELLA PROSPEZIONE

Le coppie presenti negli insiemi possono essere arricchite con la prospezione, calcolate come nel metodo LR(1).

Affiancando a ogni coppia un insieme di prospezione l'algoritmo evita di inserire negli insiemi coppie corrispondenti a scelte destinate al fallimento. Ma in questo modo il numero delle coppie per certe grammatiche può aumentare invece che diminuire. I vantaggi della prospezione sono controversi.

Le implementazioni più efficienti del metodo di Early non usano la prospezione.

2) GRAMMATICHE BNF ESTESE

L'algoritmo di Early può essere modificato per accettare anche regole grammaticali BNF estese.