ALGORITMO DI EARLEY

COSTRUZIONE DI UN ANALIZZATORE SINTATTICO: L'ALGORITMO DI EARLEY

In questa lezione faremo vedere quali sono i passi che si devono compiere per costruire un analizzatore sintattico che verifica se una stringa appartiene a un dato linguaggio, espresso sotto forma di CFG.

In particolare l'algoritmo di Earley costruisce un analizzatore sintattico a partire da una grammatica context free generica.

Vedremo in seguito altri algoritmi che permettono di costruire parser a partire da alcune grammatiche context free particolari. La scelta di tali grammatiche permette una costruzione più efficiente e un riconoscimento più intuitivo

L'ALGORITMO DI EARLEY

L'idea è quella di costruire progressivamente tutte le possibili derivazioni leftmost (o rightmost) compatibili con la stringa in input.

Durante il procedimento si analizza la stringa in input da sinistra verso destra scartando via via le derivazioni in cui non vi sia corrispondenza tra i simboli derivati e quelli della stringa.

Se esistono due derivazioni leftmost (o rightmost) possibili, l'algoritmo restituisce i due alberi di derivazione.

La complessità di calcolo è proporzionale al cubo della lunghezza della stringa da analizzare e si riduce al quadrato se la grammatica non è ambigua e ancora di più se è deterministica.

IN DETTAGLIO

Data una stringa $x_1x_2...x_n$, l'algoritmo la scandisce da sinistra verso destra e per ogni x_i costruisce gli stati S_j. Uno stato è costituito da un insieme di coppie

(dotted_rule, puntatore).

Una **dotted rule** è una produzione di *G* avente sul lato destro un punto che ne marca una posizione.

Il **puntatore** è un intero che indica la posizione dell'input a partire dalla quale è iniziato l'esame della produzione contenuta nella dotted rule.

In altre parole un elemento dello stato S_j è del tipo:

(A->α.β,i) con 0<=i<=j, α,β∈(N U T)* , A->αβ regola della grammatica

Ciò significa che:

- Si è iniziato l'esame della produzione **A->αβ** a partire dalla posizione i+1 dell'input;
- È già stata esaminata la parte **α** che precede il punto;
- È già stato verificato che **α** genera $\mathbf{x_{i+1}..x_j}$

L'ALGORITMO

Per semplicità si aggiunge il simbolo $ alla fine della stringa x e la produzione S'->S$.

input $x=x_1 \ldots x_n$

x_{n+1}=$

```
S[0]={(S'->.S$,0)}    /*stato iniziale
for j=0 to n do
   elabora ogni coppia (A->α.β,i) di S[j] applicando una delle
   3 operazioni: scansione, predizione, completamento.
if S[n+1]={(S'->S$.,0)} then
   accetta
else  rifiuta.
```

SCANSIONE (SCANNER)

Esaminiamo uno stato $(A \to \alpha . \beta, i) \in S_j$

SCANSIONE:

Si applica quando β inizia con un terminale a, cioè $\beta = a\beta'$ (cioè $A \to \alpha . a\beta'$), e $a = x_{j+1}$, allora si aggiunge la coppia $(A \to \alpha a . \beta', i)$ a S_{j+1} (cioè lo stato successivo)

La regola (che era stata applicata all'i-esimo passo) ha quindi riconosciuto il carattere j+1-esimo, lo stato successivo si dovrà preoccupare di esaminare il prossimo carattere della produzione.

Se il carattere a è diverso dal carattere x_{j+1} letto nell'input, questa produzione fallisce e non va più avanti.

PREDIZIONE (PREDICTOR)

Esaminiamo $(A \to \alpha . \beta, i) \in S_j$

PREDIZIONE:

Si applica quando β inizia con un non terminale B, cioè $\beta = B\beta'$, allora, per ogni produzione $B \to \gamma$ della grammatica, aggiungi la coppia $(B \to .\gamma, j)$ a S_j (n.b. stesso stato in cui si trova $(A \to \alpha . \beta, i)$)

Significa che si predice l'espansione di B a partire dalla posizione j

COMPLETAMENTO (COMPLETED)

$(A \to \alpha . \beta, i) \in S_j$

COMPLETAMENTO:

Si applica quando $\beta = \varepsilon$, $(A \to \alpha ., i)$ allora per ogni coppia $(C \to \eta . A\delta, h)$ di S_i (ossia ogni coppia in cui *A* appare a destra del punto nello stato i in cui è iniziata la predizione di *A*) si aggiunge $(C \to \eta A . \delta, h)$ a S_j

Ciò significa che la predizione $A \to .\alpha$ iniziata allo stato i ha avuto successo, quindi possiamo continuare ad analizzare le produzioni da cui questa è derivata (quelle del tipo $(C \to \eta . A\delta, h)$), a partire dal simbolo successivo. Si individuano allora in S_i tutti gli stati che avevano attivato la predizione $A \to .\alpha$

Significa che $\eta \Rightarrow^* x_{h+1} \dots x_i$, in più $A \to \alpha \Rightarrow^* x_{i+1} \dots x_j$, allora possiamo concludere che $\eta\alpha \Rightarrow^* x_{h+1} \dots x_j$

La derivazione (e quindi l'albero di derivazione) può essere ricostruita a ritroso.

Applicazione dell'algoritmo di Earley al riconoscimento di (a)$

S'->E$
E->E+E
E->E*E
E->(E)
E->a

- **scanner** se β inizia con un terminale a, cioè β=aβ', allora se $a=x_{j+1}$, aggiungi la coppia (A->αa.β',i) a S_{j+1}.
- **predictor** : se β inizia con un non terminale B, cioè β=Bβ', allora, per ogni produzione B->γ, aggiungi la coppia (B->.γ,j) a S[j]
- completer se β = ε allora data la coppia (A->α.,i), per ogni coppia (C->η.Aδ,h) di S_i si aggiunge (C->ηA.δ,h) a S_j

(a) $

	S_1	S_2	S_3	S_4
S'->.E$,0 •	E->(.E),0 •	E->a.,1 •	E->(E).,0 •	S'->E$.,0
E->.E+E,0 •	E->.E+E,1 •	E->(E.),0 •	S'->E.$,0 •	
E->.E*E,0 •	E->.E*E,1 •	E->E.+E,1 •	E->E.+E,0 •	
E->.(E),0 •	E->.(E),1 •	E->E.*E,1 •	E->E.*E,0 •	
E->.a,0 •	E->.a,1 •			

Nell'operazione di scansione si inserisce un puntatore tra un insieme e un altro.
Nell'operazione di completamento e di predizione all'interno dello stesso insieme.

Applicazione dell'algoritmo di Earley al riconoscimento di (a)$

S_0	S_1	S_2	S_3	S_4
S'->.E$,0 •	E->(.E),0 •	E->a.,1 •	E->(E).,0 •	S'->E$.,0
E->.E+E,0 •	E->.E+E,1 •	E->(E.),0 •	S'->E.$,0 •	
E->.E*E,0 •	E->.E*E,1 •	E->E.+E,1 •	E->E.+E,0 •	
E->.(E),0 •	E->.(E),1 •	E->E.*E,1 •	E->E.*E,0 •	
E->.a,0 •	E->.a,1 •			

S'->E$

E->E+E

E->E*E

E->(E)

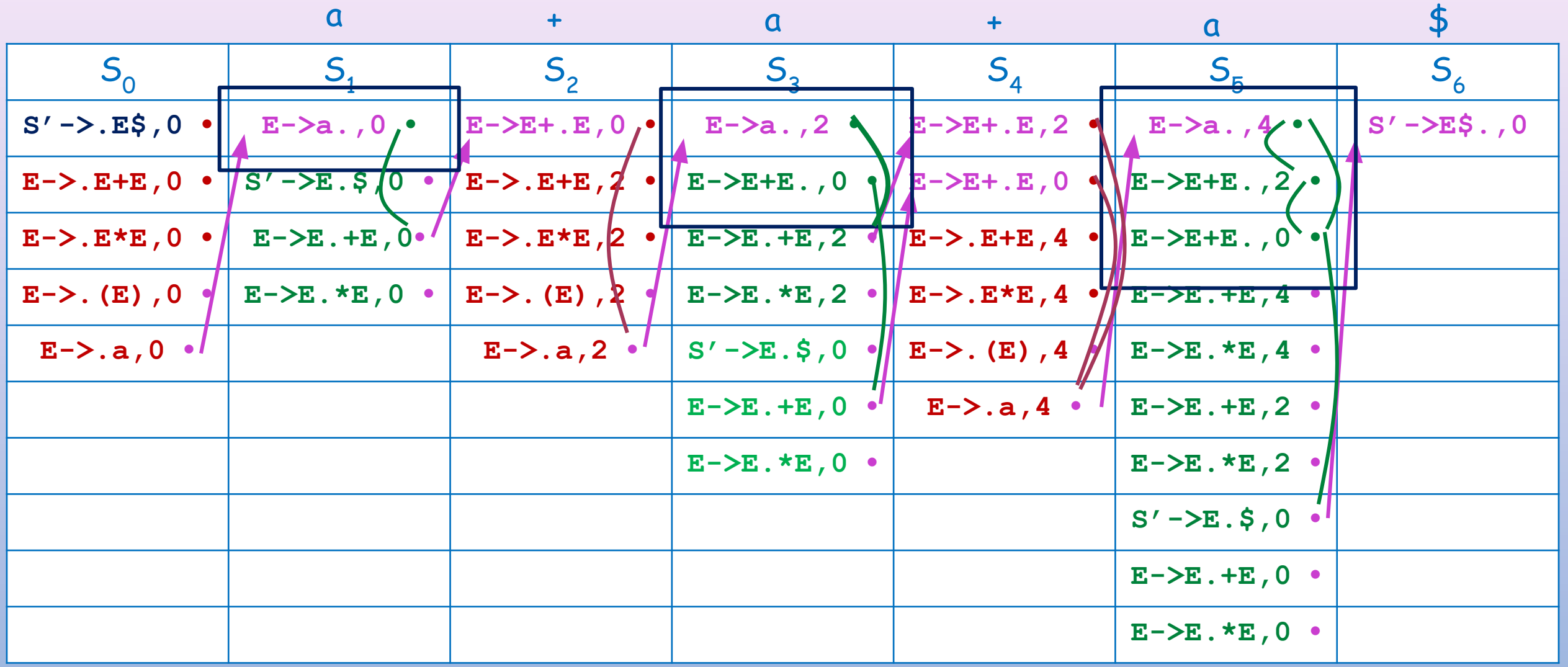
E->a

RICOSTRUZIONE DELLA DERIVAZIONE

A partire dallo stato S_4, vedo da dove ho ricavato S'->E$.,0 **ossia** S'->E.$,0 in S_3. Questo a sua volta proviene da E->(E).,0. Questo proviene da E->(E.),0 in S_2, che a sua volta proviene da E->a.,1

S'->E$->(E)$->(a)$

Esercizio: provare il parser di a+a+a



S'->E$
E->E+E
E->E*E
E->(E)
E->a

COMPLESSITÀ DELL'ALGORITMO

Ciascun insieme S_j può avere un numero di coppie che cresce linearmente con j, quindi $O(n)$;

Le operazioni di scansione e predizione su ogni coppia sono indipendenti da n;

L'operazione di completamento richiede $O(j)$ per ogni coppia, quindi in totale $O(n^2)$

Sommando i passi per ogni i si ha $O(n^3)$.

In pratica l'algoritmo è più veloce: per molte Grammatiche è $O(n)$ e per ogni grammatica non ambigua è $O(n^2)$…. Troppo per l'analisi sintattica di un Compilatore!

ESERCIZIO

Eseguire l'algoritmo di Earley per la grammatica

S->A | B

A->aAb | ab

B->aaBb| aab

e la stringa aabb e poi per la stringa aaabb

ESERCIZIO

Siano date la grammatica G e la stringa aaaa. Si esegua il riconoscimento della stringa utilizzando l'algoritmo di Earley.

G: S -> aaS | Saaa | a | ε

Nota: se ci sono produzioni A->ε, si effettua subito il completamento

(La dot_rule è A->.)

ESERCIZIO

Siano date la grammatica G e la stringa aabb. Si esegua il riconoscimento della stringa utilizzando l'algoritmo di Earley.

S -> aAbB | C

A-> aA | a

B-> bB | b

C-> aCb |ab