Algoritmo di Earley

Lunedì 4 Novembre

In un generico automa a pila sono presenti tre forme di non-determinismo

- Incertezza tra più mosse di lettura: per stato q, carattere a e simbolo A della pila, $\delta(q,a,A)$ ha più di un valore;
- Incertezza tra una mossa spontanea (senza lettura) e una mossa di lettura;
- Incertezza tra più mosse spontanee;

Se nessuna delle tre forme è presente, l'automa a pila è deterministico, e il linguaggio riconosciuto è detto context-free deterministico

Parser deterministici e non deterministici

• Un parser è l'implementazione di un automa a pila che riconosce il linguaggio generato da una certa grammatica.

• Se un parser è deterministico allora ogni frase è riconosciuta o non riconosciuta con un solo calcolo (l'automa a pila è deterministico).

Linguaggi deterministici e non deterministici

- *Un linguaggio riconosciuto da parser deterministico si dice linguaggio context-free deterministico e può essere generato da una grammatica non ambigua.*
- *La famiglia dei linguaggi context-free deterministici è strettamente contenuta in quella dei linguaggi context-free.*
- *Se un linguaggio è inerentemente ambiguo allora il parser non può mai essere deterministico.*

Per esempio: $L=\{a^ib^jc^k$ con i=j oppure j=k$\}$

$L=\{a^ib^ic^* \mid i>=0\} \cup \{a^*b^ic^i \mid i>=0\}$.

- Esistono linguaggi non ambigui ma il cui parsing è effettuato da parser non deterministici (due calcoli diversi ma solo uno termina con successo):
  Esempio: $L=\{a^nb^n \mid n>0\} \cup \{a^nb^{2n} \mid n>0\}$

Domanda

- E' deterministico il linguaggio delle parole palindrome generato dalla grammatica?
  S->aSa | bSb | a | b | ε

Tipi di parser

- Metodi Universali
  - (Cocke-Younger-Kasami 1967): usa la programmazione dinamica per stabilire se una data stringa appartiene ad un dato linguaggio context-free. Richiede che la grammatica sia in forma normale di Chomsky.
    L'algoritmo richiede tempo $O(n^3)$.
  - (Earley 1970): in grado di trattare qualsiasi grammatica CF.
    L'algoritmo richiede tempo $O(n^3)$.

- Metodi Lineari in $O(n)$, su certe grammatiche riconosciute da automi a pila deterministici:
  - analisi discendente (top-down), più intuitiva, ben adatta a grammatiche semplici;
  - analisi ascendente (bottom-up), più sofisticata, più utilizzata dai generatori automatici di analizzatori sintattici, poichè necessita poche manipolazioni della grammatica.

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 l'insieme S[i], costituito da coppie (dot_rule, puntatore).
- Una dot_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 dot_rule.

In altre parole se un elemento di S[j] è del tipo:

$(A \to \alpha.\beta, i)$ con $0 <= i <= j$

Ciò significa che:

-si è iniziato l'esame della produzione $A \to \alpha\beta$ a partire dalla posizione i+1 dell'input;

- è già stata esaminata la parte che precede il punto;
- è già stato verificato che α genera $x_{i+1}..x_j$

L'ALGORITMO

Per semplicità si aggiunge $ a x e la produzione S'->S$.

Input $x=x_1...x_n$

$x_{n+1}=\$$

S[0]={(S'->.S$,0)}

for j=0 **to** n+1 **do**

elabora ogni coppia $(A->\alpha.\beta,i)$ di S[j] applicando una delle 3 operazioni: scansione, completamento, predizione.

if S[n+1]={(S'->S$.,0)} **then**

accetta

else rifiuta.

SCANSIONE, COMPLETAMENTO, PREDIZIONE

Esaminiamo uno stato (A->α.β, i) appartenente a S[i].

- SCANSIONE: 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].
- PREDIZIONE: 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](si predice l'espansione di B a partire dalla posizione j)
- COMPLETAMENTO: Se β è la parola vuota allora data la coppia (A->α., i)

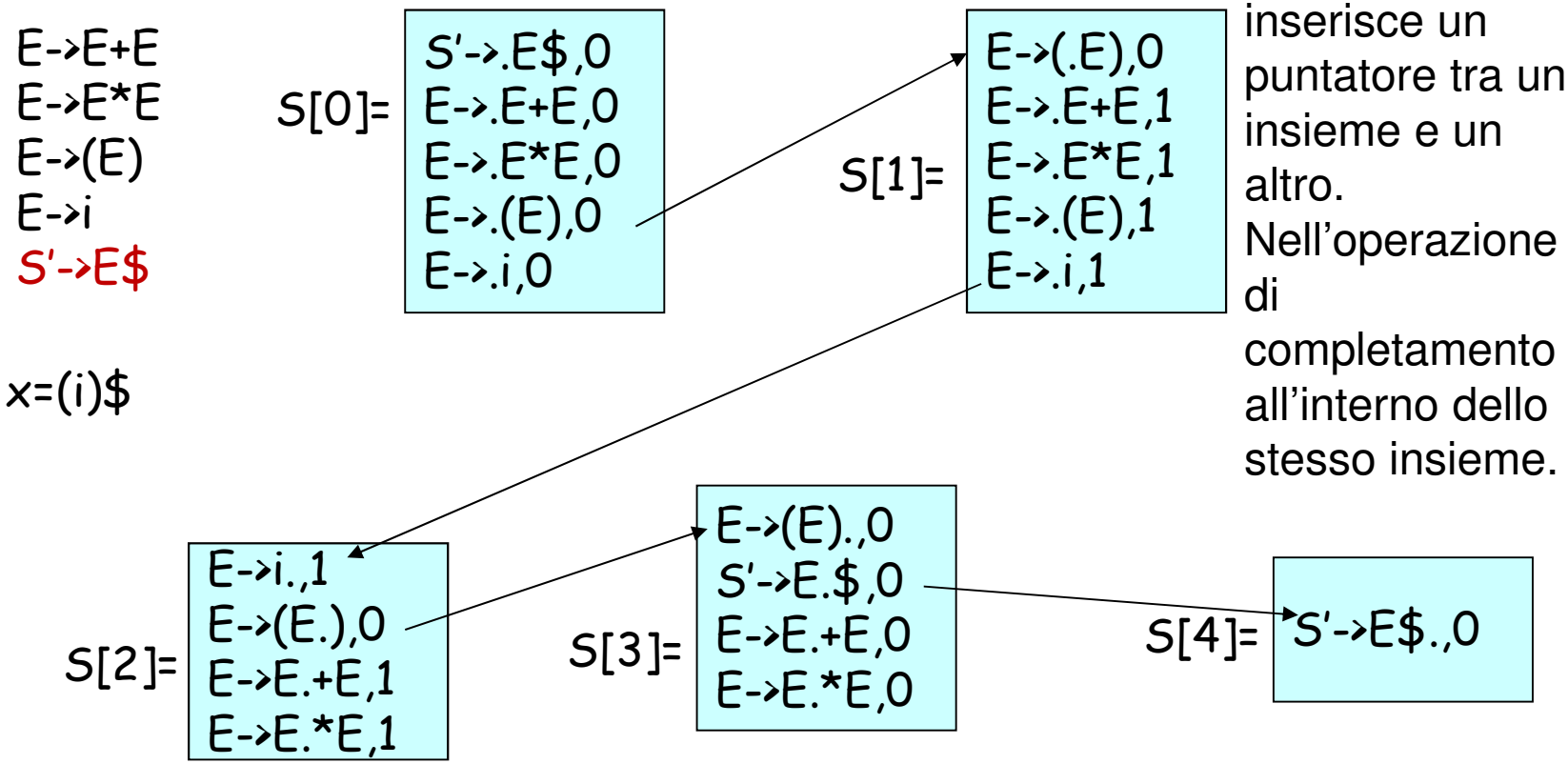
  per ogni coppia (C->η.Aδ, h) di S[i] si aggiunge (C->ηA.δ, h) a S[j]
  (la predizione A-> α si è verificata quindi si può allungare la parte riconosciuta. Si individuano allora in S[i] tutti gli stati che avevano fatto la predizione
  $\eta \Rightarrow x_{h+1}...x_i$. Siccome $A \Rightarrow \alpha \Rightarrow x_{i+1}...x_j$, allora $\eta A \Rightarrow x_{h+1}...x_j$

Proseguendo a ritroso si costruisce la derivazione e quindi l'albero di derivazione.

Esempio

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

x=(i)$



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

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

Esercizio: provare il parser di i+i+i

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 $O(n^2)$

Esercizio

- Costruire l'algoritmo di Earley per la grammatica

S->A | B

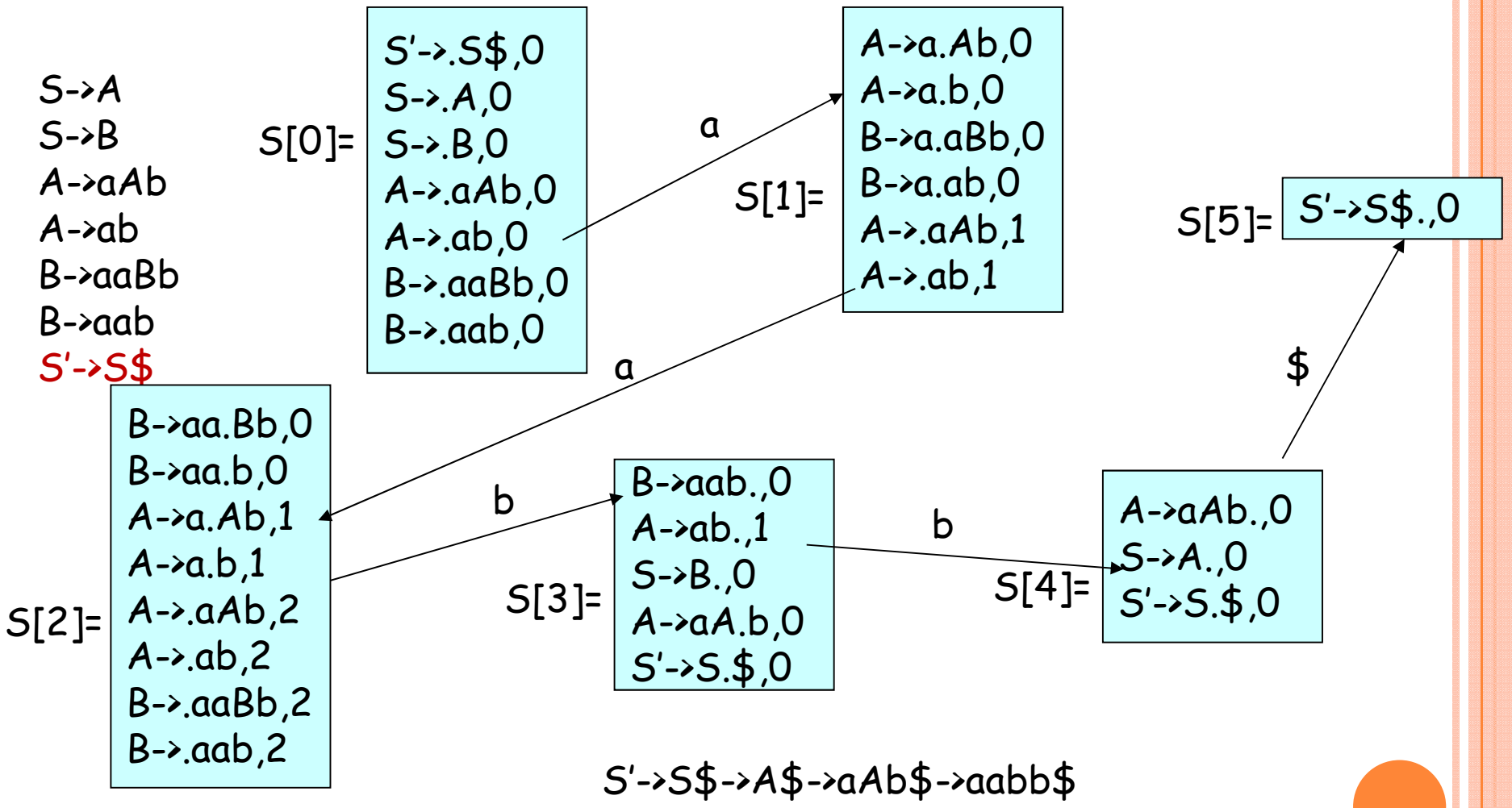
A->aAb | ab

B->aaBb| aab

e la stringa aabb

ESEMPIO

x=aabb$



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 *aaaa*. Si esegua il riconoscimento della stringa utilizzando l'algoritmo di Earley.

- G: S -> aaS | Saaa | a | ε

Esercizio

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

G:

S -> aAbB | C

A-> aA | a

B-> bB | b

C-> aCb |ab

Esercizi con Flex

- Scrivere un programma in FLEX che valuti una espressione aritmetica in forma postfissa

Soluzione (I parte)

```
/*
VALUTAZIONE IN NOTAZIONE POSTFISSA
*/

%{
#include <stdlib.h>
typedef struct elemento{
        int inf;
        struct elemento *pun;
        } ELEMENTO;

void push(ELEMENTO **p, int ele){
        ELEMENTO  *paus;
        paus=(ELEMENTO*)malloc(sizeof(ELEMENTO));
        if(paus==NULL);                         //se non c'è abbastanza memoria
        else {
        paus->pun= (*p);
        paus->inf= ele;
        (*p) = paus;
        }
        }
```

Soluzione (II parte)

```
int pop(ELEMENTO **p)
        {
        ELEMENTO *paus;
        int t=0;
        if(*p!=NULL){
        t=(*p)->inf;
        paus= (*p);
        (*p) = (*p)->pun;
        free(paus);
        }
        return t;
        }

void visualizzaPila(ELEMENTO *p){
      printf("\n<------Testa della pila ");
      while(p!=NULL) {
      printf("\n%d ", p->inf);
      p = p->pun;
      }
      }
```

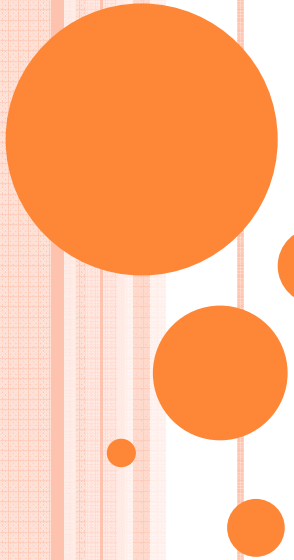
Soluzione (III parte)

```
ELEMENTO *stack=NULL;
int num;
%}

%option main

nat                                          0|[1-9][0-9]*
%%

{nat}
                                             {push(&stack,atoi(yytext));}
[+]
                                             {num=pop(&stack)+pop(&stack);push(&stack,num);}
[-]
                                             {num=pop(&stack)-pop(&stack);push(&stack,num);}
[*]
                                             {num=pop(&stack)*pop(&stack);push(&stack,num);}
[/]
                                             {num=pop(&stack)/pop(&stack);push(&stack,num);}
.
                                             ;
\n                                           {printf("= %d\n",pop(&stack));}
```



USO DI EOF IN FLEX

<<EOF>>

- La regola speciale <<EOF>> indica azioni che sono eseguite quando si incontra un end-of-file e yywrap() restistuisce un valore non nullo, ovvero indica che non ci sono altri file da processare. Le azioni devono terminare preferibilmente facendo una delle seguenti cose:
  - Assegnare yyin a un nuovo file in input.
  - Eseguire un'azione di return
  - Eseguire yyterminate().
- Nel caso di input con file multipli, si deve gestire la funzione yywrap.
- La regola <<EOF>> non dovrebbe essere usata con altri pattern. Dovrebbe solo essere qualificata con una lista di start conditions. Se non è qualificata da nessuna start condition allora si applica a tutte le start conditions.

Esercizio

Sia preso in considerazione un linguaggio di programmazione, chiamato InitC, in cui l'inizializzazione e l'assegnazione dei tipi delle variabili viene effettuata in base alle caratteristiche lessicali degli identificatori delle variabili stesse. In particolare, gli identificatori delle variabili sono parole costituite da lettere e cifre che possono cominciare solo con lettere minuscole e le definizioni dei tipi e le inizializzazioni vengono effettuate secondo le regole seguenti:

a) se l'identificatore è formato da una lettera seguita da un numero arbitrario positivo di cifre da 0 a 9, allora la variabile sarà di tipo intero e verrà inizializzata ad un valore intero uguale a quante delle suddette cifre sono uguali a '0';

b) se l'identificatore è costituito solo da lettere minuscole, allora la variabile sarà di tipo intero e viene inizializzata ad un valore che rappresenta il numero di vocali presenti a partire dalla seconda posizione; ad esempio "abdbeccdio" viene inizializzata a 3;

c) se l'identificatore è costituito da una lettera minuscola seguita da lettere maiuscole allora la variabile sarà di tipo reale e la variabile sarà inizializzata a 0;

Ogni parola formata in modo diverso da quanto descritto nei punti a), b) e c) è un elemento lessicale non identificato.

Scrivere un programma in Flex che preso in input un file di testo riconosca gli identificatori delle variabili del linguaggio InitC e restituisca in output per ogni identificatore riconosciuto le seguenti informazioni:

Variabile:……

Riga:…… (riga in cui compare)

Tipo:……

Valore:…… (valore di inizializzazione)

Inoltre deve riportare il numero degli identificatori riconosciuti e il numero delle parole che non sono elementi del lessico di InitC.