TOP-DOWN

costruiamo un **parser a discesa ricorsiva**

una procedura P_A per ogni nonterminale A

$A ::= X_1\ X_2 \ldots X_n$

procedure $P_A()$;
begin
** codice per X_1 **
** codice per X_2 **
.....
** codice per X_n **
end;

INVARIANTE

$P_A()$, applicata a stato con input γ,
termina con *successo* se e solo se $\gamma \in L(A)$

codice per X_i

P_{Xi} *se* Xi e' nonterminale

match(Xi) *se* Xi e' terminale

```
procedure match(X:token);
        begin
          if lookahead = X
          then lookahead:= nexttoken
          else fail()
        end;
```

Parser a discesa ricorsiva: Struttura Completa

```
Procedure Parser();
  **dichiarazioni procedure: Ps, PA, PX1 ,... **
 begin
   Ps;
   match(eof/$);
   accept()
 end;
```

Parser a discesa ricorsiva: Applichiamolo ad una grammatica

E::= T + T
T::= id

```
Procedure P_E();
 begin
   P_T;
   match(+);
   P_T
 end;
```

```
Procedure P_T();
 begin
   match(id)
 end;
```

```
Procedure Parser();
 Procedure  P_E();,,,,;
 Procedure  P_T();…;
 begin
   P_E;
   match(eof/$);
   accept()
 end;
```

invocazioni:

Parser()
  P_E
    P_T
      match(id)
        nexttoken
    match(+)
      nexttoken
    P_T
      match(id)
        nexttoken
    match($)
    accept()

Input:

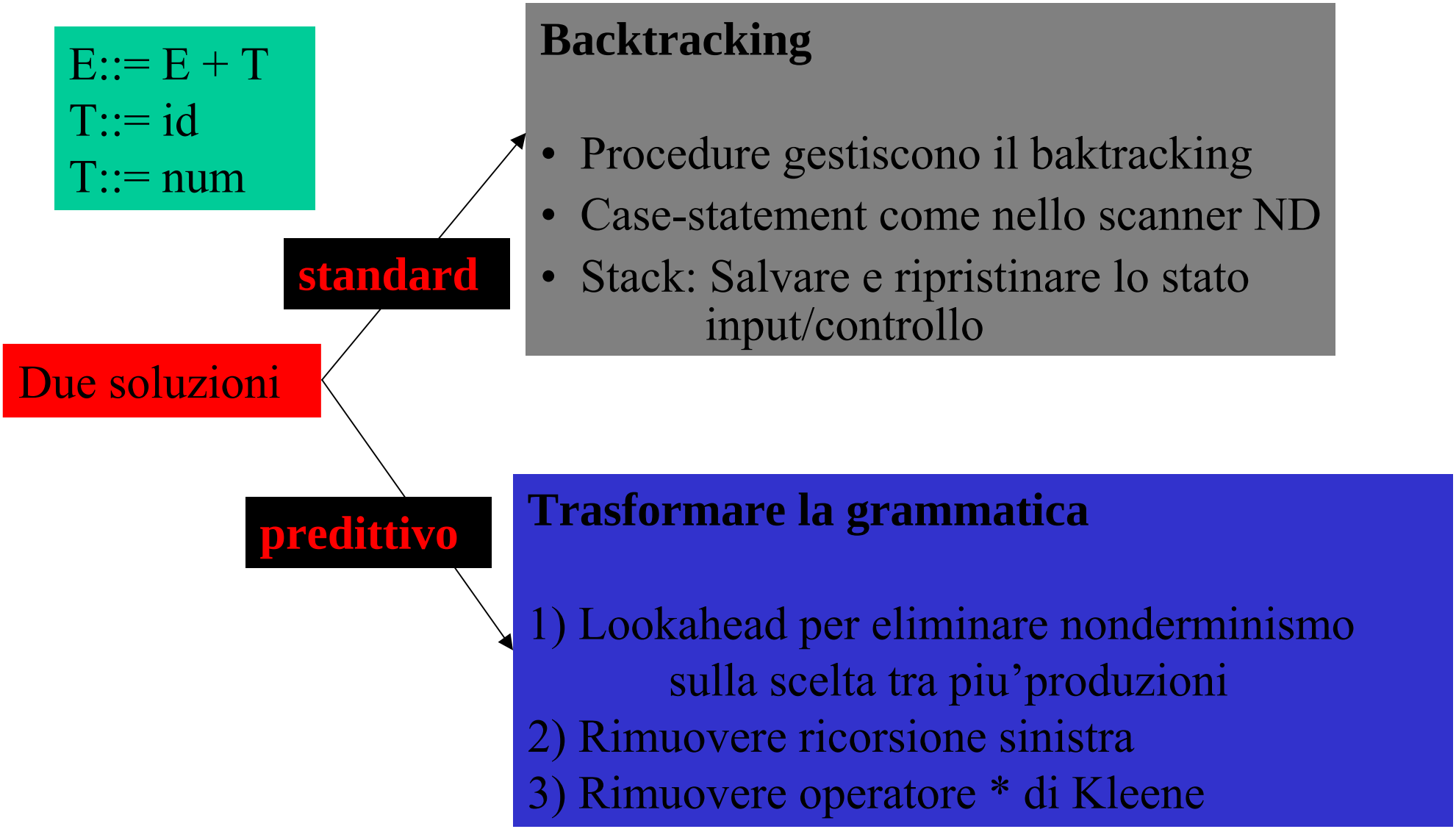
Id+id$

+id$

Id$

$

Piu' produzioni con stesso non terminale



1) Eliminare il nondeterminismo sulla scelta

guardiamo il simbolo (k-simboli) di *lookahead* e decidiamo l'alternativa

T::= id
T::= num

```
procedure P_T();
begin
   case lookahead of
            num: match(num);
            id: match(id)
   end
end;
```

K > 1 e' complicato

K = 1 non basta quando:

$E ::= \alpha\ \beta_1$
$E ::= \alpha\ \beta_2$

$|\ \alpha\ | > 1$ (|-| = lunghezza)

Left Factoring

Rimpiazziamo con:

$E ::= \alpha\ B$
$B ::= \beta_1$
$B ::= \beta_2$

2) **Eliminare ricorsione sinistra**

E::=E+id
E::=id

```
procedure P_E();
begin
   case lookahead of
            id:
   end
end;
```

$P_E()$;

match(num);

Rimuovi Ricorsione

$A ::= A\ \alpha$
$A ::= \beta$

$A ::= \beta\ \underline{A}$
$\underline{A} ::= \alpha\ \underline{A}$
$\underline{A} ::= \varepsilon$

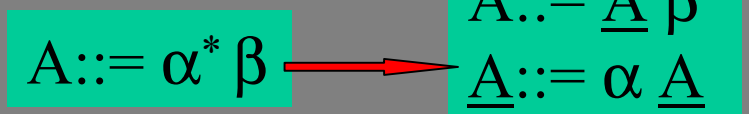
Rimuovi Mutua Ricorsione

$A ::= B\ \alpha \mid \gamma$
$B ::= A\ \beta \mid \delta$

$A ::= B\ \alpha \mid \gamma$
$B ::= (B\ \alpha \mid \gamma)\ \beta \mid \delta$

3) Eliminare stella di Kleene

Rimuovi stella



A::= A β
A::= α A
A ::= ε

Ora siamo in grado di costruire un parser **predittivo a discesa ricorsiva** che:

guarda il simbolo di lookahead e applica la procedura di riconoscimento corretta

Quale la sua complessita' ????

ovviamente O(n)

Adatto per compilazione una-passata ????

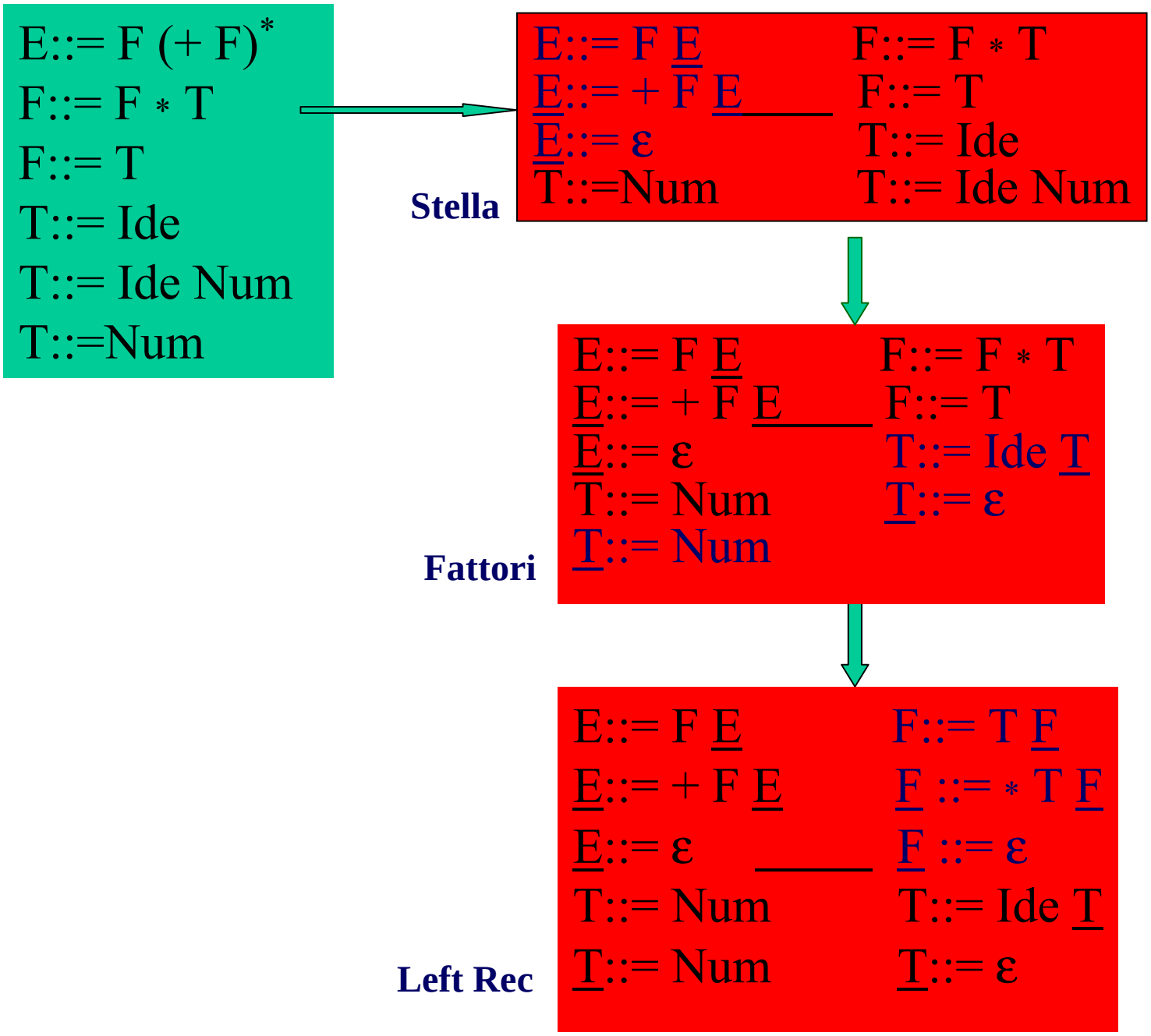
ovviamente ideale

Parser Predittivo a discesa ricorsiva:

Esempio

E::= F (+ F)*
F::= F * T
F::= T
T::= Ide
T::= Ide Num
T::= Num

Dobbiamo applicare
Il passo di trasformazione


E::= F (+ F)*
F::= F * T
F::= T
T::= Ide
T::= Ide Num
T::=Num
E::= F E
E::= + F E
E::= ε
T::=Num
F::= F * T
F::= T
T::= Ide
T::= Ide Num
Stella
E::= F E
E::= + F E
E::= ε
T::= Num
T::= Num
F::= F * T
F::= T
T::= Ide T
T::= ε
Fattori
E::= F E
E::= + F E
E::= ε
T::= Num
T::= Num
F::= T F
F ::= * T F
F ::= ε
T::= Ide T
T::= ε
Left Rec

E::= F E F::= T F

E::= + F E F ::= * T F

E::= ε F ::= ε

T::= Num T::= Ide T

T::= Num T::= ε

Dobbiamo creare le procedure con lookahead

```
procedure P_E();
begin
 P_F
 P_E;
 end;
```

```
procedure P_T();
begin
  case lookahead of
            quando?: match(Num);
            quando?: begin match(Ide); P_T end
  end
end;
```

Guardare il **primo** simbolo di cio' che derivano le *LSF*:

Num

Ide T

Guardare il **primo** simbolo di cio' che derivano

```
procedure P_T();
begin
  case lookahead of
            Num : match(Num);
            Ide: begin match(Ide); P_T end
  end
end;
```

E::= F E	F::= T F
E::= + F E	F ::= * T F
E::= ε	F ::= ε
T::= Num	T::= Ide T
T::= Num	T::= ε

```
procedure P_E();
begin
  case lookahead of
            + : begin match(+); P_F; P_E end
            quando?: nop
  end
end;
```

Non si applica a ε

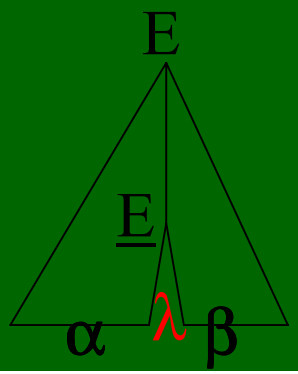
Guardare simbolo che puo' **seguire** una derivazione alla stringa vuota

E::= F E	F::= T F
E::= + F E	F ::= * T F
E::= ε	F ::= ε
T::= Num	T::= Ide T
T::= Num	T::= ε

ε => ?? non e' un simbolo

Capiamo da cio' che segue E

carattere di fine stringa $



Come determinare in modo sistematico il **primo** simbolo derivabile da una stringa su $\Sigma \cup V$

first(c)={c} *per ogni terminale*

first(s)=$\cup_{s::=\alpha \in P}$ **first**(α)

first(ε)={ε}

first($x_1 \ldots x_n$)=[$\cup_{i<k}$ (**first**(x_i)-{ε})]$\cup$**first**(x_k)

per k *massimo intero tale che*:

$x_1 \ldots x_{k-1}$ =>* ε

oppure k = n

E::= (F E) F::=id

E::= F E

E::= + F E

E::= ε

Come determinare in modo sistematico il **primo** simbolo che puo' seguire un non terminale

$$\mathbf{follow}(A) = \bigcup_{\{B::=\alpha A\beta\}} (\mathbf{first}(\beta)-\{\varepsilon\}) \cup \bigcup_{\{B::=\alpha A\beta \mid \beta =>^* \varepsilon\}} \mathbf{follow}(B) \cup \bigcup_{\{B::=\alpha A\}} \mathbf{follow}(B)$$

$\{eof/\$\} \in \mathbf{follow}(S_0)$ allorche' S_0 sia il distinto

Ora siamo in grado di costruire un parser **predittivo** a discesa ricorsiva

Per ogni nonterminale A, sia $\{\alpha_i \mid 1 \leq i \leq n, A ::= \alpha_i \in P\}$

```
procedure P_A();
        begin
        case lookahead of
                **caseof(α1)**: **codeof(α1)**;
                **caseof(α2)**: **codeof(α2)**;
                ........
                **caseof(αn)**: **codeof(αn)**;
                end
        end;
```

$$\text{caseof}(\alpha_i) = \begin{cases} \textbf{first}(\alpha_i) & \textit{se } \varepsilon \notin \textit{first}(\alpha_i) \\ (\textbf{first}(\alpha_i) - \{\varepsilon\}) \cup \textbf{follow}(A) & \textit{altrimenti} \end{cases}$$

$\text{codeof}(\alpha_i)$= *esattamente come prima*

Completiamo $P_{\underline{E}}()$;

```
procedure P_E();
begin
   case lookahead of
            + : begin match(+); P_F; P_E end
            $ : nop
   end
end;
```

E::= F E

E::= + F E

E::= ε

Parser **predittivo a discesa ricorsiva:** Considerazioni e conclusioni

Complessita': lineare O(n) *mai backtrack*
fallimento = non appartenenza

Una-passata: adatto *ad ogni simbolo ricevuto corrisponde una o piu derivazioni leftmost*

Applicabilita': LL(K) *non sempre in grado di prevedere*

Adattabilita': nessuna *cambia grammatica allora riscrivere completamente*

Applicabilita' ovvero $LL(K)_{/k=1}$

proprieta'1: Per ogni $A ::= \alpha \mid \beta$ (ovvero $\{A ::= \alpha , A ::= \beta\}$)

$$\textbf{first}(\alpha) \cap \textbf{first}(\beta) = \{\}$$

proprieta'2: Per ogni $A ::= \alpha \mid \beta$

$$\textbf{se } \alpha =>^* \varepsilon \textbf{ allora } \textbf{first}(\beta) \cap \textbf{follow}(A) = \{\}$$

Th. Data una grammatica G:

- $G \in LL(1)$ se e solo se entranbe prop.1 e prop.2
- G ha parser predittivo (lineare) 1-simbolo se e solo se entranbe prop.1 e prop.2

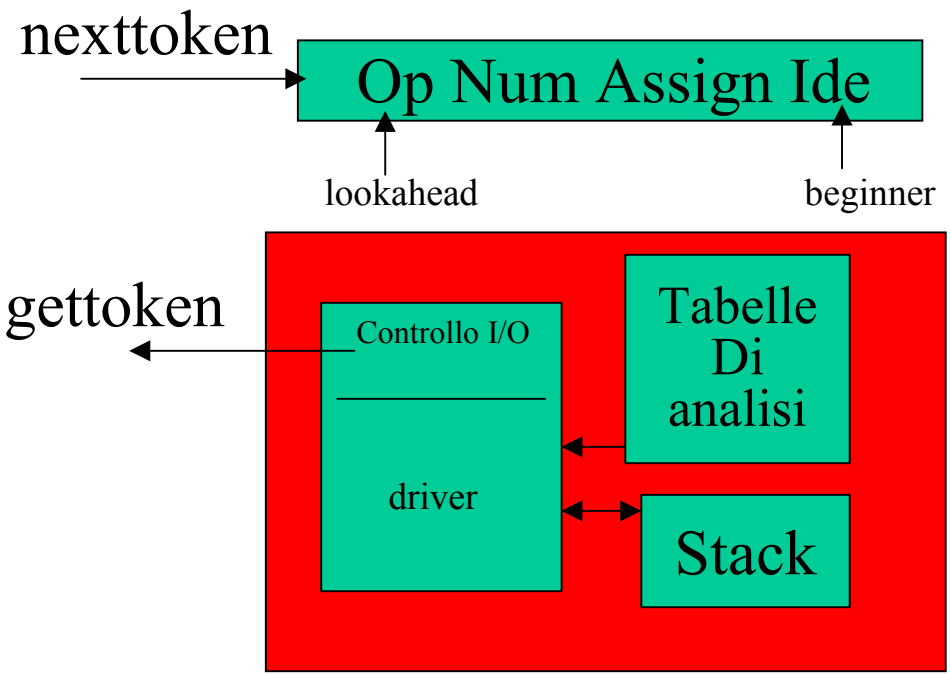
Parser Predittivo

Ricorsione vs. iterazione: *tail recursive*

Error Recovery: *rimediare all'errore*

Adattabilita': *il nostro problema*

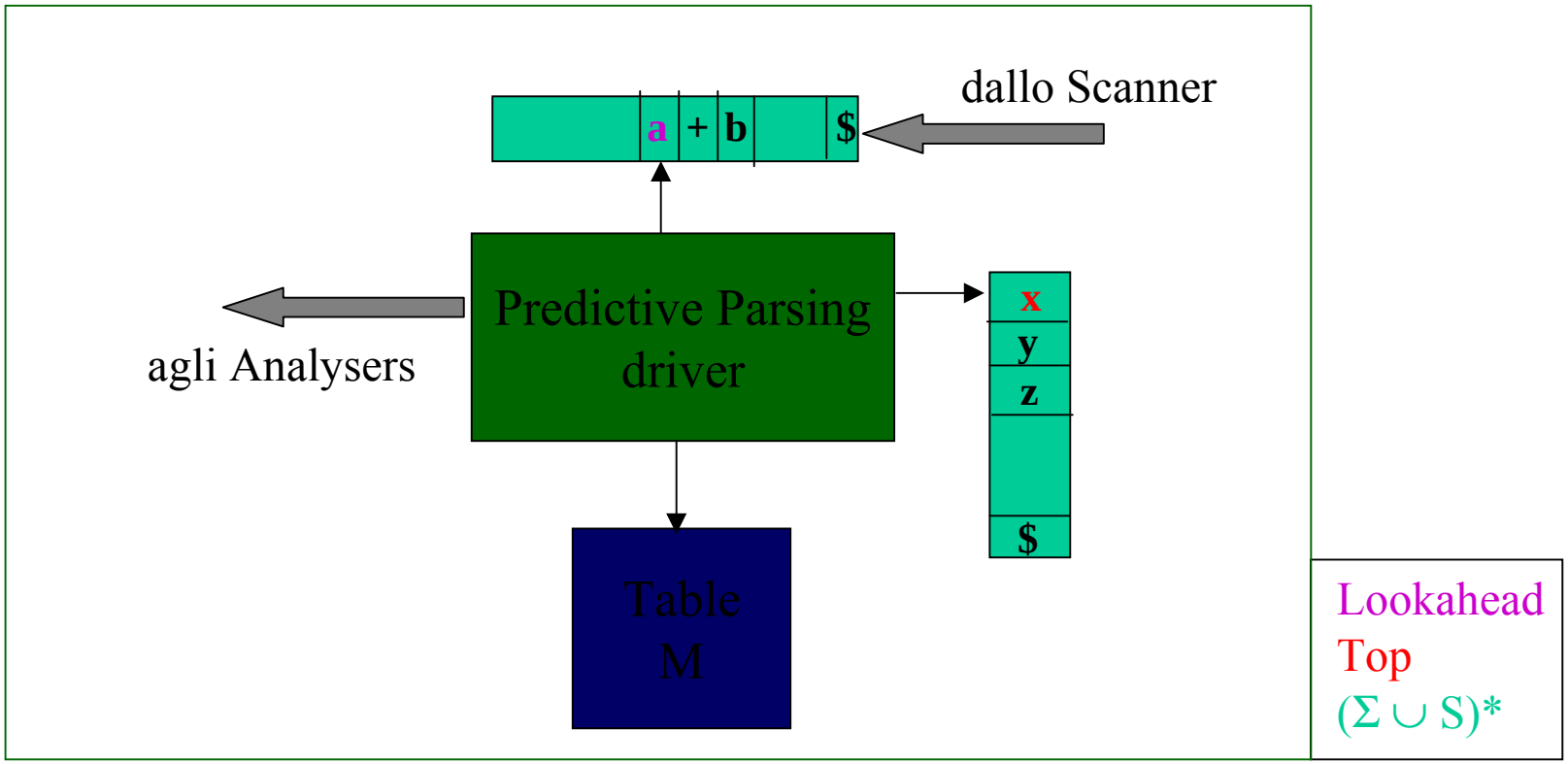
automi a pila: driver e tabelle


nexttoken
Op Num Assign Ide
lookahead
beginner
gettoken
Controllo I/O
driver
Tabelle
Di
analisi
Stack

Automa a pila (*pushdown automaton*)

$<S, \Sigma, M: S \times \Sigma \rightarrow S, D: M \times (\Sigma \cup S)^* \rightarrow (\Sigma \cup S)^*, s_0 \in S, F \subseteq S>$

per S finito



La funzione D per LL(1)

- top = lookahead = $: stop
- top = lookahead ≠ $: pop;
  lookahead:= nexttoken
- top ∈ S: pop;
  push(α)
  dove M(top,lookahead) = top::=α

Costruiamo la tabella M

Per ogni produzione $A::=\alpha_i$

+ per ogni $a \in (first(\alpha_i)-\varepsilon)$,
  $M(A,a):= A::= \alpha_i$

+ se $\varepsilon \in first(\alpha_i)$ allora:
  per ogni $b \in follow(A)$,
  $M(A,b):= A::= \alpha_i$

+ tutte le rimanenti entries sono error

Parser Predittivo Generatore/Adattivo

Esempio

Applichiamo alla grammatica gia trasformata

0.E::= F E
1.E::= + F E
2.E::= ε
3.T::= Num
4.T::= Num
5.F::= T F
6.F ::= * T F
7.F ::= ε
8.T::= Ide T
9.T::= ε

First(F E) = {Ide, Num}
First(+ F E) = {+}
First(T F) = {Ide, Num}
First(* T F) = {*}

Fw(E) = Fw(E) = {$}
Fw(F) = Fw(F) = First(E$) = {+,$}
Fw(T) = Fw(T) = First(F)∪Fw(F)
= {*}∪{+,$}

	'+	'*	Ide	Num	$
E			0	0	
E	1				2
F			5	5	
F	7	6			7
T			8	3	
T	9	9		4	9

Top Down: Conclusioni

1. Considera il linguaggio $\mathbf{T} = \{u^n v^k z^m \mid n,k,m > 0,\ n < m\}$
   a) fornire G tale che L(G) = **T**
   b) G ∈ LL(1) ?
   c) eventuali trasformazioni di G hanno un parser predittivo ?

2. Considera il linguaggio $\mathbf{T} = \{u^n v^k z^m \mid n,k,m > 0,\ n > m\}$
   a) fornire G tale che L(G) = **T**
   b) G ∈ LL(1) ?
   c) eventuali trasformazioni di G hanno un parser predittivo ?

3. Per ogni linguaggio **T** e' decidibile esistenza G tale che:
   L(G) = **T** ed in piu' G ∈ LL(1) ?

Top Down: Conclusioni (continua)

4. Sono piu' le grammatiche LL(K) di quelle LL(1) ?

5. Sono piu' i linguaggi LL(K) di quelli LL(1) ?

6. Quali condizioni per LL(k)

$$(\forall\ A ::= \beta_1 | \beta_2)\ \text{ e } ((\forall\ \gamma): S \Rightarrow_l^* \alpha A \gamma)$$
$$\text{first}_k(\beta_1\ \gamma) \cap \text{first}_k(\beta_2\ \gamma) = \{\}$$