UNIVERSITÀ
DEGLI STUDI
DI TORINO

Analisi discendente: grammatiche LL(1)

a.a. 2018-2019

Riferimenti bibliografici

Le slide seguenti presentano, in modo più dettagliato, gli argomenti trattati sul testo:

Compilatori: principi, tecniche e strumenti, A.V. Aho, M. S. Lam, R. Sethi, J.D. Ullman

Analisi sintattica [cap. 4]

4.4 Parsing top-down
- 4.4.3 Grammatiche LL(1)
- 4.4.4 Parsing predittivo non ricorsivo

Analisi sintattica top-down deterministica

L'obiettivo è costruire un analizzatore deterministico, basato sul modello dell'automa a pila, che riconosca il linguaggio scegliendo in modo deterministico le produzioni da usare.
Esamineremo il caso in cui sia possibile scegliere deterministicamente, ad ogni passo, la produzione da usare, conoscendo solo un simbolo di input. Possiamo chiamarlo automa con "*look ahead*" di lunghezza 1.

La scelta importante è determinare la produzione da applicare quando un nonterminale si trova in cima alla pila. Quando invece in cima alla pila si trova un terminale basterà verificare la corrispondenza di questo col simbolo sul nastro di input e avanzare la testina di lettura. In mancanza di questa corrispondenza la stringa in input deve essere rifiutata.

Quali sono le condizioni cui deve soddisfare la grammatica per costruire un analizzatore deterministico discendente usando 1 simbolo in input? Cioè quando una grammatica è LL(1)?

Analisi top-down deterministica: insieme guida

Per poter scegliere la riscrittura di una variabile in modo deterministico associamo ad ogni produzione un insieme di simboli terminali, chiamato «**insieme guida**».
Per riscrivere una variabile che si trova sul top dello stack viene scelta la produzione il cui insieme guida contiene il simbolo in esame sul nastro di input.

Una grammatica per $\{0^n1^n \mid n > 0\}$:

Produzioni	Insiemi guida
$S \rightarrow 0A$	{0}
$A \rightarrow S1$	{0}
$A \rightarrow 1$	{1}

La grammatica è LL(1)

Nel seguito considereremo la stringa sul nastro di input *terminata da un simbolo di fine stringa*: $

Analisi top-down deterministica: tabella

Visualizziamo le produzioni da usare in funzione delle variabili e dei simboli in input mediante una tabella che memorizza, per ogni coppia

< variabile A, simbolo in input *a* >

la produzione $A \to \alpha$ se avendo in input il simbolo terminale *a*, la variabile A sullo stack, la produzione da usare è $A \to \alpha$.

Produzione	Insieme guida
$S \to 0A$	{0}
$A \to S1$	{0}
$A \to 1$	{1}

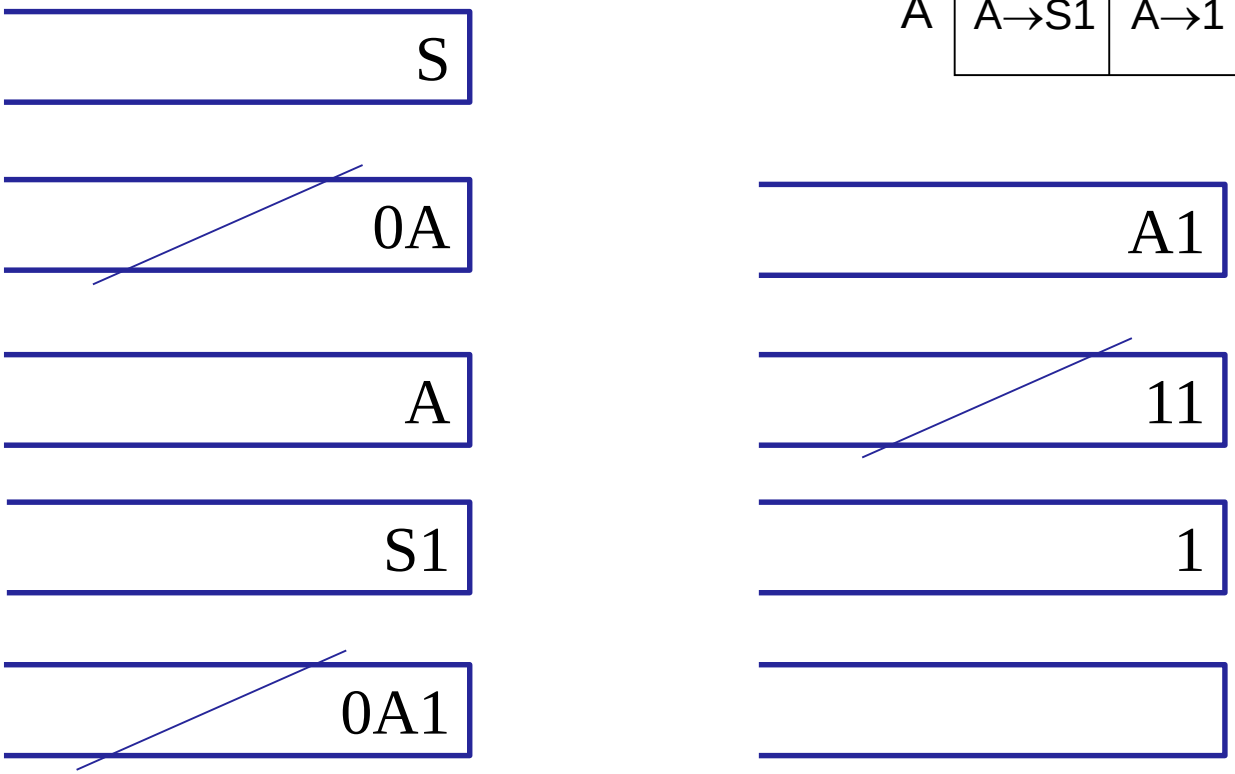
Sulla tabella si può facilmente verificare che la grammatica è LL(1)

	0	1	$
S	S→0A		
A	A→S1	A→1	

Esempio

Stringa 0 0 1 1 $

	0	1	$
S	S→0A		
A	A→S1	A→1	



$S \Rightarrow 0A \Rightarrow 0S1 \Rightarrow 00A1 \Rightarrow 0011$

Il linguaggio di disegno

- Assegnazione: ← (esempio: A ← B)
- Espressioni numeriche e booleane

- Dati composti:
  - i-esimo elemento array A: A[i]
  - Liste: - *null* costruttore di lista vuota;
    - *cons* (elemento, nomelista) aggiunge un elemento in testa alla lista.
  - Coppie: C = <a,b> chiamiamo p_1 e p_2 le proiezioni (cioè $p_1(C)$ e $p_2(C)$ individuano la prima e la seconda componente della coppia C).

- Dichiarazione e chiamata di funzione (metodo): nome (*param 1*, *param 2*, ...)
- Ritorno da una funzione: return (*valore*)
  (opzionale se non c'è valore da ritornare)
- Parametri alle funzioni passati per valore.

Il linguaggio di disegno

Statement:

struttura di blocco (sequenza): indentazione o parentesi graffe

costrutti iterativi:

costrutto condizionale:

```
if (condizione)
        azioni
[elseif (condizione) azioni]
[elseif (condizione) azioni]
…
[else azioni]
```

```
while (condizione)
        azioni
```

Esempio:

```
function A( )
 if (c1)  S1
          S2
 elseif (c2)
          S3
          S4
 else S5
```

Automa push-down “look ahead 1”

PROSS( ) è una funzione che restituisce il prossimo simbolo in input.
STACK è la pila sul cui top è memorizzato inizialmente lo start symbol della grammatica.
ERRORE(. . .) è la funzione che gestisce le segnalazioni di errore e abortisce il programma.

	0	1	$
S	S→0A		
A	A→S1	A→1	

```
main  zero-uno ( )
      cc ← PROSS( )
      X ← top (STACK)
      while (not empty (STACK))
                if (X ∈ {0,1} and X = cc)
                          cc ← PROSS( )
                          pop (STACK)
                elseif (X = S) …
                ..
                elseif (X = A) …
                …
                else ERRORE (…)
                X ← top (STACK)
      if (cc = ‘$’) stampa (“stringa accettata”)
      else ERRORE (…)
```

Esempio

```
main  zero-uno ( ) {
    . . .
        elseif (X = S)
                if (cc = 0)
                        cc ← PROSS( )
                        pop (STACK)
                        push (A, STACK)
                        stampa (S → 0A)
                else ERRORE (…)

        elseif (X = A)
                if (cc = 0)
                        pop (STACK)
                        push (S1, STACK)
                        stampa (A → S1)
                elseif (cc = 1)
                        cc ← PROSS( )
                        pop (STACK)
                        stampa (A → 1)
                else ERRORE (…)
    . . .
```

	0	1	$
S	S→0A		
A	A→S1	A→1	

Analizzatore iterativo

Algoritmo che implementa l'automa a pila look ahead 1.

Input: 1) Stringa da parsificare, seguita da un simbolo di fine stringa: $

2) Tabella M che memorizza, per ogni coppia
< variabile A, simbolo in input *a* >
la produzione $A \rightarrow \alpha$ se $a \in Gui(A \rightarrow \alpha)$
' ' se $a \notin Gui(A \rightarrow \alpha)$.

Output: Stringa accettata o segnalazione di errore;
Elenco delle produzioni usate per generare la stringa.

Analizzatore iterativo

```
AUTOMA( )
        cc ← PROSS( )                    /*1^ simbolo della stringa in input */
        X ← top (STACK)
        while (not empty (STACK))
                if (X ∈ Σ and X = cc)
                        cc ← PROSS( )
                        pop (STACK)
                elseif (X ∈ V and M[X, cc] = X → α and α ≠ cc.β )
                        pop (STACK)
                        push (α, STACK)
                        stampa (X → α)
                elseif (X ∈ V and M[X, cc] = X → α and α = cc.β )
                        cc ← PROSS( )
                        pop (STACK)
                        push (β, STACK)
                        stampa (X → α)
                else ERRORE (…)
                X ← top (STACK)
        if (cc = '$') stampa ("stringa accettata")
        else ERRORE (…)
```

Analizzatore iterativo: esempio

```
main parentesi( )
    cc ← PROSS( )
    X ← top (STACK)
    while (not empty (STACK))
            if (X ∈ { ], >} and X = cc)
                cc ← PROSS( )
                pop (STACK)
            elseif (X = S)
                if (cc = '[' ) cc ← PROSS( )
                        pop (STACK)
                        push ( S], STACK)
                elseif (cc = '<' ) cc ← PROSS( )
                        pop (STACK)
                        push ( S>, STACK)
                elseif (cc ∈ { ], >, $} pop (STACK)
                else ERRORE (…)
            else ERRORE (…)
            X ← top (STACK)
    if (cc = '$') stampa ("stringa accettata")
    else ERRORE (…)
```

Grammatica	Insiemi guida
$S \rightarrow [\ S\ ]$	{ [}
$S \rightarrow <\ S\ >$	{ < }
$S \rightarrow \varepsilon$	{], >, $}

Analizzatore a discesa ricorsiva

Ad ogni variabile A con produzioni

$A \rightarrow \alpha_1$

$A \rightarrow \alpha_2$

...

$A \rightarrow \alpha_k$

si associa una funzione:

function A( )

if (cc $\in$ Gui($A \rightarrow \alpha_1$)) body (α_1)

elseif (cc $\in$ Gui($A \rightarrow \alpha_2$)) body (α_2)

....

elseif (cc $\in$ Gui($A \rightarrow \alpha_k$)) body (α_k)

else ERRORE (…)

Analizzatore a discesa ricorsiva

Se $\alpha = \varepsilon$, body(ε) = do nothing

Se $\alpha = X_1 \ldots X_m$, body($X_1 \ldots X_m$) è così definito:

body($X_1 \ldots X_m$) = act (X_1) act (X_2) … act (X_m)

$$act(X) = \begin{cases} X( ) & \text{se } X \in V \\ cc \leftarrow \text{PROSS}( ) & \text{se } X = cc \\ \text{ERRORE}(\ldots) & \text{altrimenti} \end{cases}$$

```
main discesa_ricorsiva( )
        cc ← PROSS( )
        S( )
        if (cc = '$')  stampa "stringa accettata"
        else ERRORE(…)
```

Analizzatore ricorsivo: esempio

	0	1	$
S	S→0A		
A	A→S1	A→1	

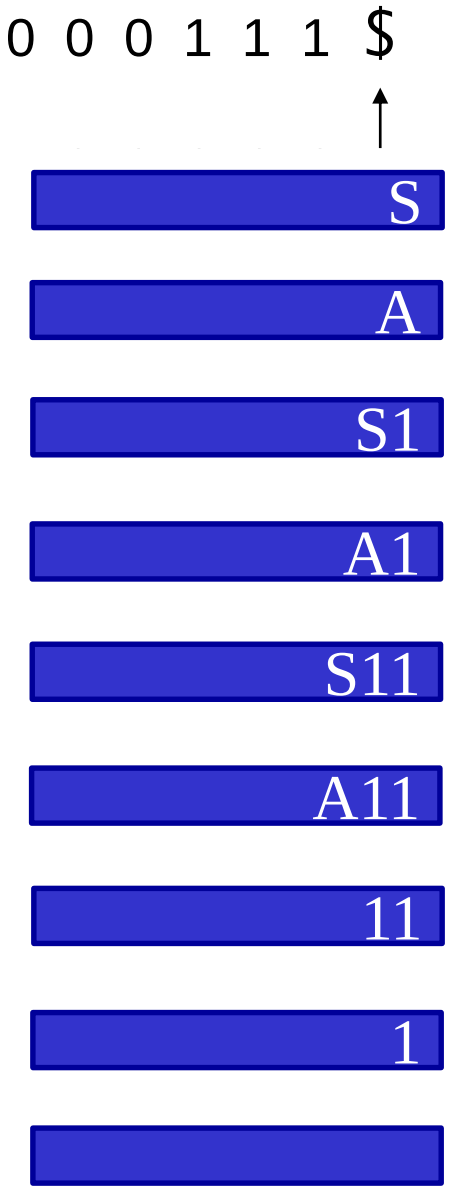
Input: 0011$

```
main  zero-uno( )
        cc ← PROSS( )
        S( )
        if (cc = '$') "stringa accettata"
        else ERRORE(…)
```

```
function  S( )
        if (cc = 0)
              cc ← PROSS( )
              A( )
        else ERRORE(…)
```

```
function  A( )
        if (cc = 0)
              S( )
              if (cc = 1) cc ← PROSS( )
              else ERRORE (…)
        elseif (cc = 1)
              cc ← PROSS( )
        else ERRORE (…)
```

Analizzatore iterativo - analizzatore ricorsivo



$S \rightarrow 0A \quad \{0\}$

$A \rightarrow S1 \quad \{0\}$

$A \rightarrow 1 \quad \{1\}$

```
if (cc = 0)
    cc ← PROSS( )
    A( )
      if (cc = 0)
        S( )
          if (cc = 0)
            cc ← PROSS( )
            A( )
              if (cc = 0)
                S( )
                  if (cc = 0)
                    cc ← PROSS( )
                    A( )
                      if (cc = '1')
                        cc ← PROSS( )
                  if (cc = 1) cc ← PROSS( )
        if (cc = 1) cc ← PROSS( )
```

Analizzatore a discesa ricorsiva: esempio

Grammatica

$S \rightarrow [\ S\ ]$

$S \rightarrow <\ S\ >$

$S \rightarrow \varepsilon$

Insiemi guida

{ [}

{< }

{], >, $}

```
main discesa_ricorsiva( )
        cc ← PROSS( )
        S( )
        if (cc = '$') "stringa accettata"
        else ERRORE(…)
```

```
function S( )
    if (cc ='[')   cc ← PROSS( )
                   S( )
                   if (cc = ']') cc ←  PROSS( )
    else ERRORE(…)
    elseif (cc ='<')
                   cc ← PROSS( )
                   S( )
                   if (cc = '>') cc ← PROSS( )
                   else ERRORE(…)
    elseif (cc ∈ {']', '>', '$'})  do nothing
    else ERRORE (…)
```

Esempio

Grammatica	Insieme guida
1. $S \rightarrow PQ$	{a, b, c}
2. $Q \rightarrow \&PQ$	{&}
3. $Q \rightarrow \varepsilon$	{$}
4. $P \rightarrow aPb$	{a}
5. $P \rightarrow bPa$	{b}
6. $P \rightarrow c$	{c}

```
Program discesa_ricorsiva ( )
        cc ← PROSS( )
        S ( )
        if (cc = '$') "stringa accettata"
        else ERRORE (…)
```

```
function S( )
    if (cc ∈ {'a', 'b', 'c'})
        P( )
        Q( )
    else ERRORE (…)
```

Esempio

```
function P( )
        if (cc = ‘a’)
                cc ← PROSS( )
                P( )
                if (cc = ‘b’) cc ← PROSS( )
                else ERRORE(…)
        elseif (cc = ‘b’)
                cc ← PROSS
                P( )
                if (cc = ‘a’) cc ← PROSS( )
                else ERRORE(…)
        elseif (cc = ‘c’) cc ← PROSS( )
        else ERRORE(…)
```

$Q \rightarrow \&PQ$	{&}
$Q \rightarrow \varepsilon$	{$}
$P \rightarrow aPb$	{a}
$P \rightarrow bPa$	{b}
$P \rightarrow c$	{c}

```
function Q( )
        if (cc = ‘&’)
                cc ← PROSS( )
                P ( )
                Q ( )
        elseif (cc =‘$’) do nothing
        else ERRORE (…)
```