Parsing top down

a.a. 2020-2021

Corso di Fondamenti di Informatica - 1 modulo

Corso di Laurea in Informatica

Università di Roma "Tor Vergata"

Prof. Giorgio Gambosi

Parsing predittivo

In una derivazione sinistra di una stringa, una forma di frase è necessariamente del tipo $V_T^+(V_T \cup V_N)^*$.

Esempio: la grammatica

$$\begin{aligned} T &\longrightarrow R \mid aTc \\ R &\longrightarrow RbR \mid \varepsilon \end{aligned}$$

e la produzione sinistra

$$T \Longrightarrow \underline{a}Tc \Longrightarrow \underline{aa}Tcc \Longrightarrow \underline{aa}Rcc \Longrightarrow \underline{aa}RbRcc \Longrightarrow \underline{aa}RbRbRcc \Longrightarrow$$
$$\underline{aab}RbRcc \Longrightarrow \underline{aab}RbRbRcc \Longrightarrow \underline{aabb}RbRcc \Longrightarrow \underline{aabbb}Rcc \Longrightarrow \underline{aabbbcc}$$

Parsing predittivo

- Nel corso di un parsing predittivo con input x, alla forma di frase $wA\alpha$, con $w \in V_T^*$, $A \in V_N$, $\alpha \in (V_T \cup V_N)^+$, corrisponde una situazione in cui il parser ha letto il prefisso w della stringa x e deve determinare, sulla base di esso e della rimanente parte z della stringa $x = wz$, quale delle produzioni aventi A a sinistra applicare.
- Se la produzione selezionata è $A \longrightarrow yB\beta$, con $y \in V_T^*$, $B \in V_N$, $\beta \in (V_T \cup V_N)^+$, la nuova forma di frase è $wyB\beta\alpha$
- Il parser ad ogni istante fa riferimento alla forma di frase attuale e alla parte di stringa di input ancora da leggere: all'inizio evidentemente queste informazioni sono l'assioma S e l'intera stringa x in input

Parsing predittivo

$$
\begin{aligned}
T &\longrightarrow R \mid aTc \\
R &\longrightarrow RbR \mid \varepsilon
\end{aligned}
$$

letta	forma di frase	stringa	possibilità	scelta
ε	$\underline{T}$	aabbcc	$T \longrightarrow aTc \mid T \longrightarrow R$	$T \longrightarrow aTc$
a	a$\underline{T}$c	abbcc	$T \longrightarrow aTc \mid T \longrightarrow R$	$T \longrightarrow aTc$
aa	aa$\underline{T}$cc	bbcc	$T \longrightarrow aTc \mid T \longrightarrow R$	$T \longrightarrow R$
aa	aa$\underline{R}$cc	bbcc	$R \longrightarrow RbR$	$R \longrightarrow RbR$
aa	aa$\underline{R}$bRcc	bbcc	$R \longrightarrow RbR \mid R \longrightarrow \varepsilon$	$R \longrightarrow RbR$
aa	aa$\underline{R}$bRbRcc	bbcc	$R \longrightarrow RbR \mid R \longrightarrow \varepsilon$	$R \longrightarrow \varepsilon$
aa	aab$\underline{R}$bRcc	bbcc		-
aab	aab$\underline{R}$bRcc	bcc	$R \longrightarrow RbR \mid R \longrightarrow \varepsilon$	$R \longrightarrow \varepsilon$
aab	aabb$\underline{R}$cc	bcc		-
aabb	aabbRcc	cc	$R \longrightarrow RbR \mid R \longrightarrow \varepsilon$	$R \longrightarrow \varepsilon$
aabbc	aabbcc	c		-
aabbcc	aabbcc	ε		-

Parsing a discesa ricorsiva

Implementazione di parser top down: una funzione `A()` per ogni $A \in V_N$, con la struttura seguente. Il programma inizia da S()

A():
  for each $A \longrightarrow X_1 X_2 \cdots X_k \in P$:
    for i in range$(1, k+1)$:
      if $X_i \in V_N$:
        if not $\mathtt{X}_i()$:
          break
        else :
          if X_i uguale al prossimo simbolo a della stringa:
            avanza al simbolo successivo
          else :
            break
    return True
  return False

Parsing a discesa ricorsiva

- Utilizzo del backtracking:esplorazione ricorsiva di tutte le possibilità
- Backtracking: trial and error
- La grammatica non può essere ricorsiva sinistra

$$A \longrightarrow Aw$$

- Può essere molto inefficiente
- Può essere reso efficiente se la scelta della produzione da considerare può essere guidata dall'esame dei caratteri successivi

Parsing predittivo

Grammatica

$$
\begin{aligned}
E &\longrightarrow TE' \\
E' &\longrightarrow +TE' \mid \varepsilon \\
T &\longrightarrow FT' \\
T' &\longrightarrow *FT' \mid \varepsilon \\
F &\longrightarrow (E) \mid \texttt{id}
\end{aligned}
$$

Stringa

`id+id*id`

Parsing predittivo

ε	E	id+id*id	$E \longrightarrow TE'$
ε	TE′	id+id*id	$T \longrightarrow FT'$
ε	FT′E′	id+id*id	$F \longrightarrow$ id
id	idT′E′	+id*id	$T' \longrightarrow \varepsilon$
id	idE′	+id*id	$E' \longrightarrow +TE'$
id+	id+TE′	id*id	$T \longrightarrow FT'$
id+	id+FT′E′	id*id	$F \longrightarrow$ id
id+id	id+idT′E′	*id	$T' \longrightarrow *FT'$
id+id*	id+id*FT′E′	id	$F \longrightarrow$ id
id+id*id	id+id*idT′E′	ε	$T' \longrightarrow \varepsilon$
id+id*id	id+id*idE′	ε	$E' \longrightarrow \varepsilon$
id+id*id	id+id*id	ε	-

Ad ogni passo è possibile selezionare una sola produzione, guardando un solo terminale (token)

Parsing predittivo efficiente

Se per ogni simbolo non terminale da espandere i prossimi k caratteri della stringa consente di individuare la produzione da applicare, il parser è $LL(k)$

- Left-to-right: la derivazione è calcolata da sinistra a destra (dalla prima produzione applicata all'ultima)
- Leftmost derivation: la derivazione calcolata è sinistra
- k simboli (di look-ahead) da considerare

Un linguaggio CF è $LL(k)$ se esiste un parser $LL(k)$ che può effettuarne l'analisi sintattica

Costruzione di un parser LL: la funzione FIRST

Consideriamo il caso $LL(1)$, per semplicità.

- Per ogni sequenza $\alpha \in (V_T \cup V_N)^+$, FIRST($\alpha$) è l'insieme dei terminali che possono comparire all'inizio di una forma di frase derivata da α
- quindi, $c \in$ FIRST(α) se e solo se esiste $\beta \in (V_T \cup V_N)^*$ e $\alpha \stackrel{*}{\Longrightarrow} c\beta$

Costruzione di un parser LL: la funzione FIRST

- Siamo in particolare interessati a $FIRST(\alpha)$ se α è la parte destra di una produzione $A \longrightarrow \alpha$
- Questo perché se per un qualunque c, se $c \in \text{FIRST}(\alpha)$ e $A \longrightarrow \alpha$, allora una stringa che inizia per c potrebbe essere derivata a partire da A, in quanto $A \Longrightarrow \alpha \overset{*}{\Longrightarrow} c\beta$

Costruzione di un parser LL: la funzione FIRST

Date le A-produzioni in P

$$A \longrightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$$

- se ogni terminale appartiene a non più di un insieme FIRST(α_i), allora possiamo sempre individuare quale produzione applicare per riscrivere A, esaminando il solo prossimo carattere c
- infatti, va applicata $A \longrightarrow \alpha_i$ se e solo se $c \in \text{FIRST}(\alpha_i)$
- se non esiste α_i tale che $c \in \text{FIRST}(\alpha_i)$, c'è un errore e la parte di stringa da leggere non è derivabile a partire da A

Costruzione della funzione FIRST

Per la costruzione di FIRST(α) va utilizzato il predicato Nullable(β) definito come Nullable(β) = `TRUE` se e solo se β è annullabile, cioè se e solo se esiste una derivazione $\beta \stackrel{*}{\Longrightarrow} \varepsilon$. Una produzione $B \longrightarrow \beta$ è annullabile se e solo se β è annullabile.

La costruzione di Nullable(β) è basata sulle seguenti proprietà

$$\begin{aligned}
\text{Nullable}(\varepsilon) &= \texttt{TRUE} \\
\text{Nullable}(a) &= \texttt{False} \qquad \forall a \in V_T \\
\text{Nullable}(\alpha\beta) &= \text{Nullable}(\alpha) \wedge \text{Nullable}(\beta) \\
\text{Nullable}(A) &= \bigvee_i \text{Nullable}(\alpha_i) \qquad \forall A \in V_N, \forall A \longrightarrow \alpha_i \in P
\end{aligned}$$

Costruzione della funzione FIRST

La costruzione di FIRST(α) avviene in modo simile.

Consideriamo in primo luogo la costruzione di FIRST(X), dove X è un simbolo della grammatica, $X \in V_T \cup V_N$

1. Se $X \in V_T$, allora FIRST(X) = $\{X\}$
2. Se $X \in V_N$, per ogni $X \longrightarrow Y_1 Y_2 \cdots Y_k \in P$ ($k \geq 1$):
   - 2.1 FIRST(Y_1) $\subseteq$ FIRST(X)
   - 2.2 Per $i = 2, \ldots k$, se Nullable($Y_1 \cdots Y_{i-1}$) allora FIRST(Y_i) $\subseteq$ FIRST(X)

Costruzione della funzione FIRST

Costruzione di FIRST($X_1 \cdots X_n$) da FIRST(X) per ogni X:

- FIRST(X_1) $\subseteq$ FIRST($X_1 \cdots X_n$)
- Per $i = 2, \ldots n$, se Nullable($X_1 \cdots X_{i-1}$) allora FIRST(X_i) $\subseteq$ FIRST($X_1 \cdots X_n$)

La funzione FIRST

Da quanto detto, se

- $x = cy$ è la stringa da leggere
- A è il terminale da riscrivere

allora le possibili produzioni da applicare sono tutte le $A \longrightarrow \alpha_i$ tali che $c \in \text{FIRST}(\alpha_i)$.

Se in tutti i casi possibili c'è al più una di tali produzioni, abbiamo un parser $LL(1)$.

La funzione FIRST

Errore! In realtà, se A è annullabile, cy potrebbe essere prodotta in modo diverso:

- Supponiamo che la forma di frase attuale sia ABw, con $w \in (V_T \cup V_N)^*$
- dato che A è annullabile, esiste una derivazione $A \overset{*}{\Longrightarrow} \varepsilon$

allora, $x = cy$ potrebbe essere ancora derivabile se $c \in \text{FIRST}(B)$ (e quindi se $B \overset{*}{\Longrightarrow} c\beta$) in quanto

$$ABw \overset{*}{\Longrightarrow} Bw \overset{*}{\Longrightarrow} c\beta w$$

La funzione FOLLOW

Definiamo la funzione FOLLOW nel modo seguente:

- Per ogni non terminale $A \in V_N$, FOLLOW(A) è l'insieme dei terminali che possono comparire subito dopo A in una forma di frase derivata da S
- Quindi, dati $A \in V_N$ e $c \in V_T$, $c \in$ FOLLOW(A) se e solo se esistono $\alpha, \beta \in (V_T \cup V_N)^*$ tali che $S \stackrel{*}{\Longrightarrow} \alpha A c \beta$
- In realtà, la funzione FOLLOW(A) riveste interesse soltanto se Nullable(A) = `TRUE`, quindi se esiste una derivazione $A \stackrel{*}{\Longrightarrow} \varepsilon$

La funzione FOLLOW

Durante il parsing:

- Siano A il non terminale da riscrivere e c il simbolo attualmente letto
- Se $c \in \text{FOLLOW}(A)$ e $\text{Nullable}(A) = \texttt{TRUE}$

allora la derivazione $A \overset{*}{\Longrightarrow} \varepsilon$ può portare all'annullamento di A

Costruzione della funzione FOLLOW

Per tener conto del caso in cui A potrebbe essere l'ultimo simbolo di una forma di frase, cioè in cui $S \stackrel{*}{\Longrightarrow} \alpha A$, estendiamo la grammatica con:

- un non terminale \$ di fine stringa
- un nuovo assioma S'
- una produzione $S' \longrightarrow S\$$

Evidentemente, A può comparire a fine stringa nella prima grammatica se e solo se $\$ \in \text{FOLLOW}(A)$ nella nuova grammatica.

Costruzione della funzione FOLLOW

FOLLOW viene costruita a partire da un insieme di vincoli derivati dalle produzioni.

- $\$ \in \text{FOLLOW}(S)$
- Se $A \longrightarrow \alpha B \beta \in P$, allora $\text{FIRST}(\beta) \subseteq \text{FOLLOW}(B)$
- se $A \longrightarrow \alpha B \beta \in P$ e $\text{Nullable}(\beta)$, allora $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$
- se $A \longrightarrow \alpha B \in P$ allora $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$

Esempio

Grammatica

$$
\begin{aligned}
E &\longrightarrow TE' \\
E' &\longrightarrow +TE' \mid \varepsilon \\
T &\longrightarrow FT' \\
T' &\longrightarrow *FT' \mid \varepsilon \\
F &\longrightarrow (E) \mid \texttt{id}
\end{aligned}
$$

Esempio

$\text{Nullable}(E') = \text{Nullable}(T') = \texttt{TRUE}$

- $\text{FIRST}(F) = \{(, \texttt{id}\}$
- $\text{FIRST}(T') = \{*\}$
- $\text{FIRST}(E') = \{+\}$
- $\text{FIRST}(T) = \text{FIRST}(F) = \{(, \texttt{id}\}$
- $\text{FIRST}(E) = \text{FIRST}(T) = \{(, \texttt{id}\}$

Esempio

- $\$ \in \text{FOLLOW}(E)$
- $\text{FIRST}(E') = \{+\} \subseteq \text{FOLLOW}(T)$
- $\text{FIRST}(T') = \{*\} \subseteq \text{FOLLOW}(F)$
- $\text{FIRST}(')') = \{)\} \subseteq \text{FOLLOW}(E)$
- $\text{FOLLOW}(E) \subseteq \text{FOLLOW}(E')$
- $\text{FOLLOW}(E) \subseteq \text{FOLLOW}(T)$
- $\text{FOLLOW}(T) \subseteq \text{FOLLOW}(T')$
- $\text{FOLLOW}(E') \subseteq \text{FOLLOW}(T)$
- $\text{FOLLOW}(T') \subseteq \text{FOLLOW}(F)$

Esempio

Da cui deriva

- $\text{FOLLOW}(E) = \{\$,)\}$
- $\text{FOLLOW}(E') = \text{FOLLOW}(E) = \{\$,)\}$
- $\text{FOLLOW}(T) = \text{FOLLOW}(E) \cup \{+\} = \{\$,), +\}$
- $\text{FOLLOW}(T') = \text{FOLLOW}(T) = \{\$,), +\}$
- $\text{FOLLOW}(F) = \text{FOLLOW}(T') \cup \{*\} = \{\$,), +, *\}$

Tabella di parsing predittivo

Associa ad ogni coppia (a, X), $a \in V_T$, $X \in V_N$, un insieme di produzioni (1 se $LL(1)$) da applicare nel caso in cui X sia il non terminale da riscrivere e a sia il simbolo letto in input.

Costruzione della tabella M:

Per ogni produzione $A \longrightarrow \alpha \in P$:

- se $\alpha \neq \varepsilon$, per ogni $a \in \text{FIRST}(A)$ aggiungi $A \longrightarrow \alpha$ a $M[A, a]$
- se Nullable(α), per ogni $b \in \text{FOLLOW}(A)$ aggiungi $A \longrightarrow \alpha$ a $M[A, b]$

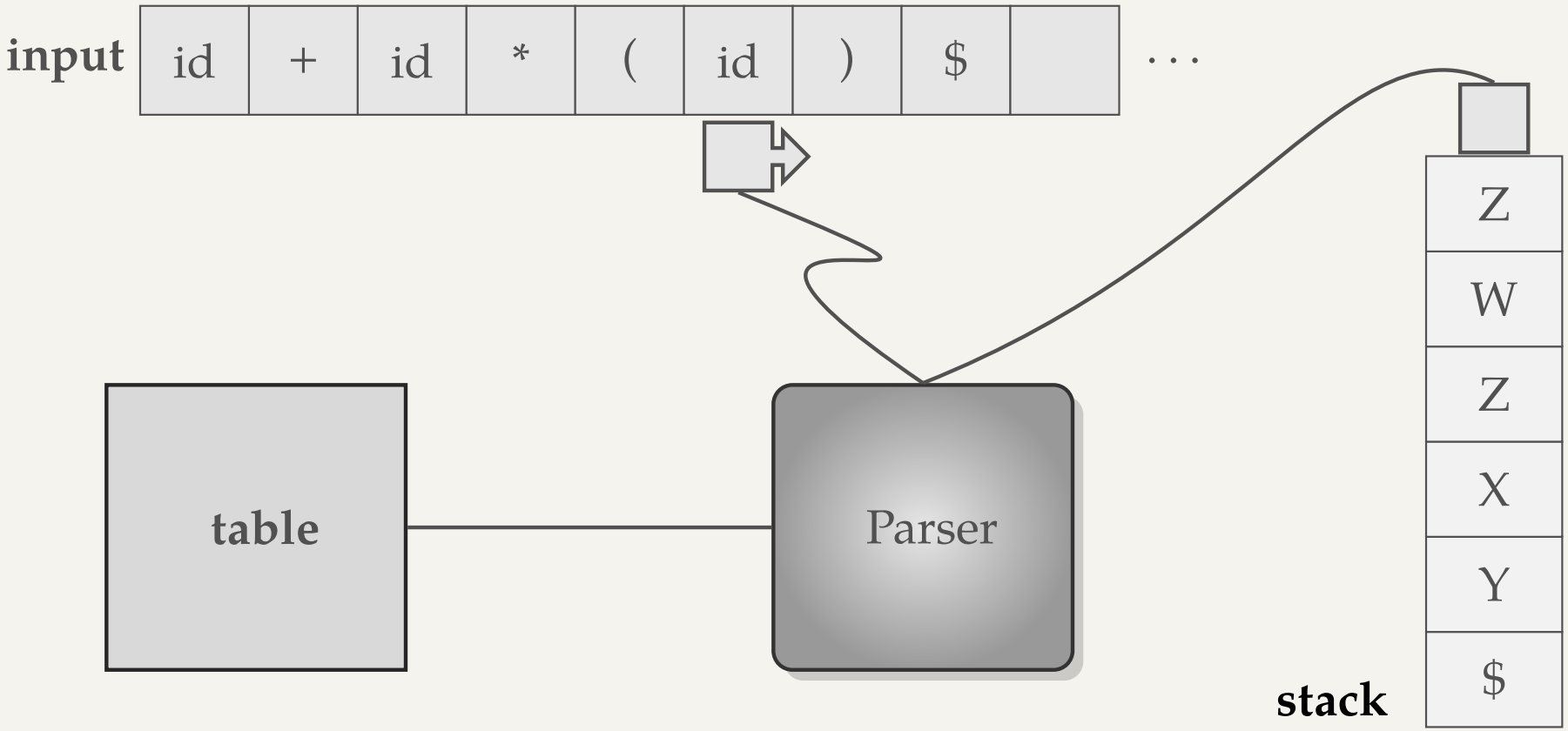
Esempio

Per la grammatica precedente

	`id`	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to$ `id`			$F \to (E)$		

Parsing predittivo non ricorsivo

Utilizza uno stack (pila) in modo esplicito, invece che implicitamente , simulando una derivazione sinistra della stringa.



Parsing predittivo non ricorsivo

input.first()
stack.push(S$)
while stack.top()! = $:
  if stack.top() == input.current():
    stack.pop()
    input.next()
  elif table[stack.top(), input.current()]! = Null:
    Let table[stack.top(),input.current()] be $X \longrightarrow Y_1 \cdots Y_k$
    output stack.top() $\longrightarrow Y_1 \cdots Y_k$
    stack.pop()
    stack.push($Y_1 \cdots Y_k$)
  else:
    error

Parsing predittivo non ricorsivo

Esempio di parsing di id + id $*$ id

Matched	Stack	Input	Action
	E$	id+id*id$	
	TE′$	id+id*id$	output $E \rightarrow TE'$
	FT′E′$	id+id*id$	output $T \rightarrow FT'$
	idT′E′$	id+id*id$	output $F \rightarrow$ id
id	T′E′$	+id*id$	match id
id	E′$	+id*id$	output $T' \rightarrow \varepsilon$
id	+TE′$	+id*id$	output $E' \rightarrow +TE'$
id+	TE′$	id*id$	match +
id+	FT′E′$	id*id$	output $T \rightarrow FT'$
id+	idT′E′$	id*id$	output $F \rightarrow$ id
id+id	T′E′$	*id$	match id

Parsing predittivo non ricorsivo

Matched	Stack	Input	Action
id+id	*FT′E′$	*id$	output $T' \to *FT'$
id+id*	FT′E′$	id$	match $*$
id+id*	idT′E′$	id$	output $F \to$ id
id+id*id	T′E′$	$	match id
id+id*id	E′$	$	output $T' \to \varepsilon$
id+id*id	$	$	output $E' \to \varepsilon$

Parsing predittivo non ricorsivo

Ne risulta la derivazione sinistra

$$
\begin{aligned}
E &\Longrightarrow TE' && \Longrightarrow FT'E' && \Longrightarrow \mathrm{id}T'E' \\
&\Longrightarrow \mathrm{id}E' && \Longrightarrow \mathrm{id} + TE' && \Longrightarrow \mathrm{id} + FT'E' \\
&\Longrightarrow \mathrm{id} + \mathrm{id}T'E' && \Longrightarrow \mathrm{id} + \mathrm{id} * FT'E' && \Longrightarrow \mathrm{id} + \mathrm{id} * \mathrm{id}T'E' \\
&\Longrightarrow \mathrm{id} + \mathrm{id} * \mathrm{id}E' && \Longrightarrow \mathrm{id} + \mathrm{id} * \mathrm{id}
\end{aligned}
$$

Parsing predittivo non ricorsivo

E l'albero sintattico