4 自顶向下解析

徐辉，xuh@fudan.edu.cn

本章学习目标：

- 了解 Earley 解析算法；
- 掌握 LL(1) 文法；
- 掌握 LL(1) 解析算法；

4.1 自顶向下解析

给定 CFG 文法 G 和句子 s，找到由文法推导出该句子的过程称为解析。本章介绍一种自顶向下的解析思路，即从 G 的初始符号或语法解析树的根结点开始，根据语法规则递归向下展开每个非终结符，直至最终生成的语法解释树叶子节点顺序与目标句子完全匹配。本章采用最左推导的方法，即每次选择当前状态最左侧的非终结符展开。对于无二义性的 CFG 文法 G，每个句子 $s \in L(G)$ 至多存在一种解析方式；如不存在则说明此句子不属于该语言，即 $s \notin L(G)$。

该解析问题的难点是每一步应如何选取合适的规则。一个基本思路是根据当前非终结符和目标终结符选取产生式。下面分别介绍两种解析方法，Earley 算法和 LL(1) 文法。

4.2 Earley 解析算法

Earley 解析算法 [1] 是一种通用的 CFG 解析算法，即可解析任意 CFG 文法。Earley 算法涉及以下三种基本操作（非正式定义）：

- **预测**：对于状态 $X \to \alpha \circ Y\beta$，根据语法规则展开 $Y \to \circ\gamma$；符号 $\circ$ 表示当前解析位置；
- **扫描**：如果下一个终结符是 a，且存在状态 $X \to \alpha \circ a\beta$，则将状态变更为 $X \to \alpha a \circ \beta$；
- **完成（或更新）**：$Y \to \circ\gamma$ 即完成了对 Y 的分析，将所有关联状态 $X \to \alpha \circ Y\beta$ 更新为 $X \to \alpha Y \circ \beta$。

为避免左递归等导致的展开方式选择爆炸问题，Earley 有很多独特的设计，详细算法可参考 1。

算法 1 Earley 解析算法

Input: G: context-free grammar; ts: token stream; **Output**: a parse tree;

```
1: procedure EARLEY PARSE(ts, G)
2:     S[0].add((G → •γ, 0))
3:     for each i in 0..ts.len() do
4:         for each item in S[i] do
5:             match item :
6:                 case FIN ⇒  // a complete state
7:                     Complete(item, i)
8:                 case others ⇒  // not a complete state
9:                     if NextSymbol(item) == ts[i] then
10:                        Scan(item, i, ts)
11:                    else// not a terminal symbol
12:                        Predict(item, i, G)
13:                    end if
14:            end match
15:        end for
16:    end for
17: end procedure
18: procedure COMPLETE((A → β•, j), i)
19:    for each (B → α•Aδ, k)∈ S[j] do
20:        S[i].add((B → αA•δ, k))
21:        if δ == ϵ then
22:            Complete((B → αA•, k), i)
23:        end if
24:    end for
25: end procedure
26: procedure PREDICT((A → α•Bβ, j), i)
27:    for each B → γ in G do
28:        S[i].add((B → γ, j))
29:    end for
30: end procedure
31: procedure SCAN((A → α•aβ, j), i)
32:    if a == ts[i] then
33:        S[i + 1].add((A → αa•β, j))
34:    end if
35: end procedure
```

下面以解析算式 `1+2*3` 为例演示 Earley 算法的解析步骤。

表 4.1: 状态 $S[0]$: `E` → ∘ `<UNUM> '+' <UNUM> '*' <UNUM>`

序号	操作	条目	
		规范项	起源
1	初始化	`E` → ∘ `E OP1 E1`	S[0]
2	初始化	`E` → ∘ `E1`	S[0]
3	预测 2	`E1` → ∘ `E1 OP2 E2`	S[0]
4	预测 2	`E1` → ∘ `E2`	S[0]
5	预测 4	`E2` → ∘ `E3 OP3 E2`	S[0]
6	预测 5	`E2` → ∘ `E3`	S[0]
7	预测 5	`E3` → ∘ `NUM`	S[0]
8	预测 5	`E3` → ∘ `<LPAR> E <RPAR>`	S[0]
9	预测 7	`NUM` → ∘ `<UNUM>`	S[0]
10	预测 7	`NUM` → ∘ `<SUB> <UNUM>`	S[0]
11	扫描 9	-	-

表 4.2: 状态 $S[1]$: E → <UNUM> ∘ '+' <UNUM> '*' <UNUM>

序号	操作	条目	
		规范项	起源
1	扫描 s[0][9]	NUM → <UNUM> ∘	S[0]
2	完成: 基于 1 更新 s[0][7]	E3 → NUM ∘	S[0]
3	完成: 基于 2 更新 s[0][5]	E2 → E3 ∘ OP3 E2	S[0]
4	完成: 基于 2 更新 s[0][6]	E2 → E3 ∘	S[0]
5	完成: 基于 4 更新 s[0][4]	E1 → E2 ∘	S[0]
6	完成: 基于 5 更新 s[0][2]	E → E1 ∘	S[0]
7	完成: 基于 5 更新 s[0][3]	E1 → E1 ∘ OP2 E2	S[0]
8	完成: 基于 6 更新 s[0][1]	E → E ∘ OP1 E1	S[0]
9	预测 3	OP3 → ∘ '^'	S[1]
10	预测 7	OP2 → ∘ '*'	S[1]
11	预测 7	OP2 → ∘ '/'	S[1]
12	预测 8	OP1 → ∘ '+'	S[1]
13	预测 8	OP1 → ∘ '-'	S[1]
14	扫描 12	-	-

表 4.3: 状态 $S[2]$: E → <UNUM> '+' ∘ <UNUM> '*' <UNUM>

序号	操作	条目	
		规范项	起源
1	扫描 s[1][12]	OP1 → '+' ∘	S[1]
2	完成: 基于 1 更新 s[1][8]	E → E OP1 ∘ E1	S[0]
3	预测 2	E1 → ∘ E1 OP2 E2	S[2]
4	预测 2	E1 → ∘ E2	S[2]
5	预测 4	E2 → ∘ E3 OP3 E2	S[2]
6	预测 5	E2 → ∘ E3	S[2]
7	预测 5	E3 → ∘ NUM	S[2]
8	预测 5	E3 → ∘ <LPAR> E <RPAR>	S[2]
9	预测 7	NUM → ∘ <UNUM>	S[2]
10	预测 7	NUM → ∘ <SUB> <UNUM>	S[2]
11	扫描 9	-	-

表 4.4: 状态 $S[3]$: E → <UNUM> '+' <UNUM> ∘ '*' <UNUM>

序号	操作	条目	
		规范项	起源
1	扫描 s[11][9]	NUM → <UNUM> ∘	S[2]
2	完成: 基于 1 更新 s[2][7]	E3 → NUM ∘	S[2]
3	完成: 基于 2 更新 s[2][5]	E2 → E3 ∘ OP3 E2	S[2]
4	完成: 基于 2 更新 s[2][6]	E2 → E3 ∘	S[2]
5	完成: 基于 4 更新 s[2][4]	E1 → E2 ∘	S[2]
6	完成: 基于 5 更新 s[2][2]	E → E OP1 E1 ∘	S[0]
7	完成: 基于 5 更新 s[2][3]	E1 → E1 ∘ OP2 E2	S[2]
8	预测 3	OP3 → ∘ '^'	S[3]
9	预测 7	OP2 → ∘ '*'	S[3]
10	预测 7	OP2 → ∘ '/'	S[3]
11	扫描 9	-	-

表 4.5: 状态 $S[4]$: E → <UNUM> '+' <UNUM> '*' ∘ <UNUM>

序号	操作	条目	
		规范项	起源
1	扫描 s[1][9]	OP2 → '*' ∘	S[3]
2	完成：基于 1 更新 s[3][7]	E1 → E1 OP2 ∘ E2	S[2]
3	预测 2	E2 → ∘ E3 OP3 E2	S[4]
4	预测 2	E2 → ∘ E3	S[4]
5	预测 3	E3 → ∘ NUM	S[4]
6	预测 3	E3 → ∘ <LPAR> E <RPAR>	S[4]
7	预测 5	NUM → ∘ <UNUM>	S[4]
8	预测 5	NUM → ∘ <SUB> <UNUM>	S[4]
11	扫描 7	-	-

表 4.6: 状态 $S[5]$: E → <UNUM> '+' <UNUM> '*' <UNUM> ∘

序号	操作	条目	
		规范项	起源
1	扫描 s[4][7]	NUM → <UNUM> ∘	S[4]
2	完成：基于 1 更新 s[4][5]	E3 → NUM ∘	S[4]
3	完成：基于 2 更新 s[4][3]	E2 → E3 ∘ OP3 E2	S[4]
4	完成：基于 2 更新 s[4][4]	E2 → E3 ∘	S[4]
5	完成：基于 4 更新 s[4][2]	E1 → E1 OP2 E2 ∘	S[2]
6	完成：基于 5 更新 s[2][2]	E → E OP1 E1 ∘	S[0]

4.3 LL(1) 文法和解析

4.3.1 LL(1) 文法

为了降低解析算法的复杂度，我们可以强制要求 CFG 文法具备某些特性，如 LL(1)（Left-to-right, Left most, lookahead 1 symbol）有两个基本要求，一是不含左递归，二是无回溯特性。下面对这两个特性进行探讨。

4.3.1.1 左递归和消除

对一条文法规则来说，如果其右侧推导出的第一个符号与左侧符号相同，则存在左递归问题，如（E ↦ E OP1 E1）。左递归可能会使搜索过程无限递归下去，无法终止。一般可以采用下列方式对左递归文法进行修改。

$$\begin{aligned} &\texttt{X} \mapsto \texttt{X 'a'} \mid \texttt{X 'b'} \mid \texttt{'c'} \mid \texttt{'d'} \\ &\qquad\Downarrow \\ &\texttt{X} \mapsto \texttt{'c' Y} \mid \texttt{'d' Y} \\ &\texttt{Y} \mapsto \texttt{'a' Y} \mid \texttt{'b' Y} \mid \epsilon \end{aligned} \tag{4.1}$$

将该方法应用于上一章的计算器文法，可消除其左递归问题。结果如语法规则 4.5所示。

$$
\begin{aligned}
\texttt{E} &\mapsto \texttt{E1 E'}\\
\texttt{E'} &\mapsto \texttt{OP1 E1 E'}\\
\texttt{E'} &\mapsto \epsilon\\
\texttt{E1} &\mapsto \texttt{E2 E1'}\\
\texttt{E1'} &\mapsto \texttt{OP2 E2 E1'}\\
\texttt{E1'} &\mapsto \epsilon\\
\texttt{E2} &\mapsto \texttt{E3 OP3 E2}\\
\texttt{E2} &\mapsto \texttt{E3}\\
\texttt{E3} &\mapsto \texttt{NUM}\\
\texttt{E3} &\mapsto \texttt{'(' E ')'}\\
\texttt{NUM} &\mapsto \texttt{<UNUM>}\\
\texttt{NUM} &\mapsto \texttt{'-' <UNUM>}\\
\texttt{OP1} &\mapsto \texttt{'+'}\\
\texttt{OP1} &\mapsto \texttt{'-'}\\
\texttt{OP2} &\mapsto \texttt{'*'}\\
\texttt{OP2} &\mapsto \texttt{'/'}\\
\texttt{OP3} &\mapsto \texttt{'\^{}'}
\end{aligned}
\tag{4.2}
$$

4.3.1.2 无回溯语法

对于每个非终结符的任意两条规则，如果其产生的首个终结符均不同，则前瞻一个单词总能够选择正确的规则。当规则的首个字符是非终结符时，应对该非终结符递归展开直至遇到终结符为止。

$$
\begin{aligned}
&X \xrightarrow{[i]}{}' a'...\\
&X \xrightarrow{[j]}{}' b'...\\
&X \xrightarrow{[k]} Y... \xrightarrow{[l]}{}' c'...\\
&...
\end{aligned}
\tag{4.3}
$$

当文法规则存在回溯问题时，可以通过提取左公因子消除回溯。

$$
\begin{aligned}
&X \rightarrow' a'A|'a'B|'b'\\
&\qquad\Downarrow\\
&X \rightarrow' a'Y|'b'\\
&Y \rightarrow A|B
\end{aligned}
\tag{4.4}
$$

将该方法应用于语法规则 4.5，可消除其回溯问题。结果如语法规则 [1]所示。

$$
\begin{aligned}
\texttt{E} &\mapsto \texttt{E1 E'} && ([1])\\
\texttt{E'} &\mapsto \texttt{OP1 E1 E'} && ([2])\\
\texttt{E'} &\mapsto \epsilon && ([3])\\
\texttt{E1} &\mapsto \texttt{E2 E1'} && ([4])\\
\texttt{E1'} &\mapsto \texttt{OP2 E2 E1'} && ([5])\\
\texttt{E1'} &\mapsto \epsilon && ([6])\\
\texttt{E2} &\mapsto \texttt{E3 E2'} && ([7])\\
\texttt{E2'} &\mapsto \texttt{OP3 E2} && ([8])\\
\texttt{E2} &\mapsto \epsilon && ([9])\\
\texttt{E3} &\mapsto \texttt{NUM} && ([10])\\
\texttt{E3} &\mapsto \texttt{'(' E ')'} && ([11])\\
\texttt{NUM} &\mapsto \texttt{<UNUM>} && ([12])\\
\texttt{NUM} &\mapsto \texttt{'-' <UNUM>} && ([13])\\
\texttt{OP1} &\mapsto \texttt{'+'} && ([14])\\
\texttt{OP1} &\mapsto \texttt{'-'} && ([15])\\
\texttt{OP2} &\mapsto \texttt{'*'} && ([16])\\
\texttt{OP2} &\mapsto \texttt{'/'} && ([17])\\
\texttt{OP3} &\mapsto \texttt{'\^{}'} && ([18])
\end{aligned}
$$

4.3.2 构造 LL(1) 解析表

我们定义 $First(X \xrightarrow{[i]} \beta_1\beta_2...\beta_n)$ 表示 X 的第 i 条规则产生的首字符集合。如果 $\epsilon \notin \beta_1$, 则 $First(X) = First(\beta_1)$; 如果 $\epsilon \in \beta_1 \& ... \& \epsilon \in \beta_i$, 则 $First(X) = First(\beta_{i+1})$。如果 $\epsilon \in \beta_1 \& ... \& \epsilon \in \beta_n$, 我们还需考虑 X 之后可能出现的字符 $Follow(X \xrightarrow{[i]} ...)$, 并据此决定是否采用规则 $X \mapsto \epsilon$。因此我们使用 $First^+(X \xrightarrow{[i]} \beta)$X 的第 i 条规则可产生的首字符集合（不含 ϵ）。

$$
First^+(X \mapsto \beta) = \begin{cases} First(\beta), & \text{if } \epsilon \in \beta \\ First(\beta) \cup Follow(X), & \text{otherwise} \end{cases}
$$

基于上述定义，我们可以准确描述出无回溯语法的必要性质。

$$
\forall 1 \le i, j \le n, First^+(X \to \beta_i) \cap \text{ First }^+(X \to \beta_j) = \emptyset
$$

表 **??**展示了语法 [1]中每条规则对应的 $First$ 集合；其每一行表示一个非终结符，每一列表示一个终结符，单元格内容表示对应的规则编号。

表 4.7: 记录每条生成式的 $First$ 集合。

	<UNUM>	'+'	'-'	'*'	'/'	'^'	'('	')'	ϵ
E	[1]		[1]				[1]		
E'		[2]	[2]						[3]
E1	[4]		[4]				[4]		
E1'				[5]	[5]				[6]
E2	[7]		[7]				[7]		
E2'						[8]			[9]
E3	[10]		[10]				[11]		
NUM	[12]		[13]						
OP1		[14]	[15]						
OP2				[16]	[17]				
OP3						[18]			

进一步消除表 4.7中的 ϵ 字符便可以得到 $First^+$ 或 LL(1) 解析表 4.8。基于无回溯文法的特性，该表的所有单元格至多存在一条规则。通过查表便可以实现精准快速解析。

表 4.8: LL(1) 解析表：生成式的 $First^+$ 集合。

	<UNUM>	'+'	'-'	'*'	'/'	'^'	'('	')'
E	[1]		[1]				[1]	
E'		[2]	[2]					[3]
E1	[4]		[4]				[4]	
E1'		[6]	[6]	[5]	[5]			[6]
E2	[7]		[7]				[7]	
E2'		[9]	[9]	[9]	[9]	[8]		[9]
E3	[10]		[10]				[11]	
NUM	[12]		[13]					
OP1		[14]	[15]					
OP2				[16]	[17]			
OP3						[18]		

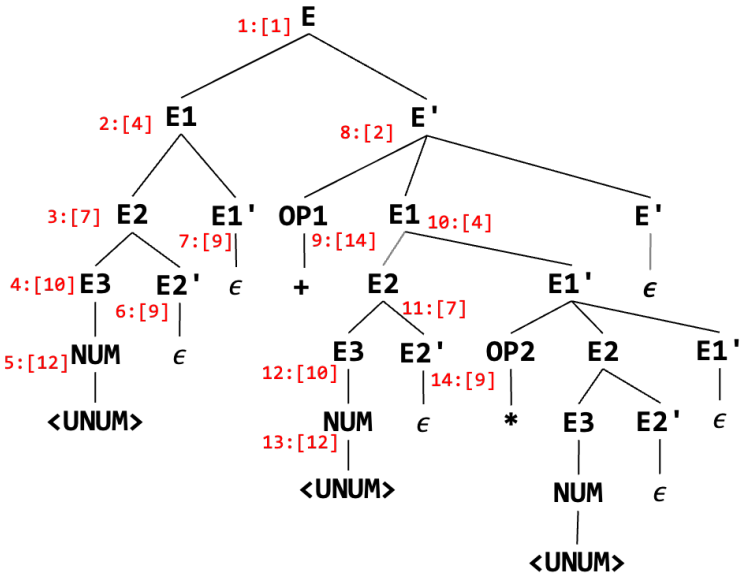

图 4.1: 应用表 4.8解析 1+2*3 的过程和最终的语法解析树

图 4.1展示了使用表 4.8解析算式 1+2*3 的过程和最终结果。

4.4 练习

1. 已知下列正则表达式 CFG 文法，应用 Earley 算法解析正则表达式 `ab*|c`。

$$
\begin{aligned}
\texttt{Regex} &\mapsto \texttt{Regex '|' Concat} \\
\texttt{Regex} &\mapsto \texttt{Concat} \\
\texttt{Concat} &\mapsto \texttt{Concat Closure} \\
\texttt{Concat} &\mapsto \texttt{Closure} \\
\texttt{Closure} &\mapsto \texttt{Closure '*'} \\
\texttt{Closure} &\mapsto \texttt{Item} \\
\texttt{Item} &\mapsto \texttt{'(' Regex ')'} \\
\texttt{Item} &\mapsto \texttt{<Char>}
\end{aligned}
\tag{4.5}
$$

2. 上述文法是否是 LL(1)，如果不是将其改为 LL(1) 并构造解析表。

3. 分析比较 Earley 算法和 LL(1) 算法的复杂度。

Bibliography

[1] Jay Earley. "An efficient context-free parsing algorithm." Communications of the ACM 13, no. 2 (1970): 94-102.