

3 上下文无关文法

徐辉, xuh@fudan.edu.cn

本章学习目标:

- 掌握上下文无关文法。
- 掌握 LL(1) 文法。
- 掌握自顶向下解析方法。

3.1 上下文无关文法

3.1.1 使用上下文无关文法声明算式语言

定义 1 (上下文无关文法 (Context-free Grammar)). 由一系列形如 $X \mapsto \gamma$ 的生成式 (productions) 组成, 其中 X 是非终结符, γ 可以是终结符 (terminal symbol) 或非终结符。

注意, CFG 文法的生成式左侧为一个非终结符, 不含其它限制条件, 否则不是 CFG 文法。例如, $aX \mapsto ab$, $bX \mapsto bc$ 的左侧对于如何展开 X 有其它前置条件限制。我们一般使用 BNF 范式 (Backus-Naur Form) 来表示 CFG 文法。

```
[1] E → E <ADD> E
[2]   | E <SUB> E
[3]   | E <MUL> E
[4]   | E <DIV> E
[5]   | E <EXP> E
[6]   | <LPAR> E <RPAR>
[7]   | NUM
[8] NUM → <UNUM>
[9]   | <SUB> <UNUM>
```

图 3.1: 计算器的 CFG 语法

图 3.1 尝试用 CFG 文法定义计算器算式, 即通过对算式 E 可能的形式进行逐步展开和枚举得到。

3.1.2 二义性问题和消除

图 3.1 定义的文法对于特定算式可能会存在两种以上的推导方式, 带来二义性问题。以算式 $1 + 2 * 3$ 为例, 图 ?? 展示了两种可能存在的语法解析树, 但是只有解释树 1 是正确的。该二义性的原因是没有考虑运算符优先级。

图 3.3 将优先级和结合性信息加入 CFG 文法中, 从而消除了二义性问题。主要思路是先将运算符按照优先级分类并使用不同的符号表示, 如 $OP1$ 表示优先级最低的加减运算, $OP2$ 表示乘除运算, $OP3$ 表示优先级最高的指数运算。同时, 我们将算式按照优先级由低到高的方式展开, 如使用 E 表示顶层算式是加减运算 ($E \mapsto E \text{ } OP1 \text{ } E1$), 它的子算式即可以是优先级更高的乘除运算 (使用另外一个符号 $E1$ 表示), 也可以是同等优先级的加减运算; 为了保证文法等价性, E 也可以直接是乘除运算 $E1$ 。在结合性方面, ($E \mapsto E \text{ } OP1 \text{ } E1$) 可以实现运算符 $OP1$ 左结合的特性; ($E2 \mapsto E3 \text{ } OP3 \text{ } E2$) 可以实现运算符 $OP2$ 右结合的特性。

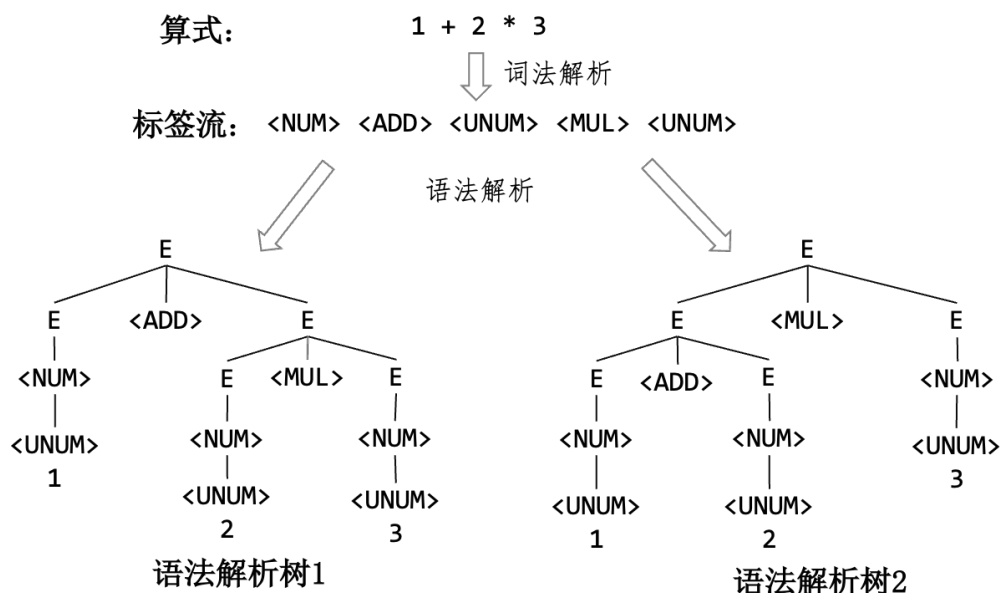


图 3.2: 语法解析: 算式 $1+2*3$

```

[1] E → E OP1 E1
[2]   | E1
[3] E1 → E1 OP2 E2
[4]   | E2
[5] E2 → E3 OP3 E2
[6]   | E3
[7] E3 → NUM
[8]   | <LPAR> E <RPAR>
[9] NUM → <UNUM>
[10]   | <SUB> <UNUM>
[11] OP1 → <ADD>
[12]   | <SUB>
[13] OP2 → <MUL>
[14]   | <DIV>
[15] OP3 → <POW>

```

图 3.3: 消除二义性后的 CFG 语法

3.2 自顶向下解析

给定 CFG 文法和句子，找到由文法推导出该句子的过程称为解析。如何找到该推导方式呢？一种方式就是从根节点 E 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与目标算式等价；如果出现不匹配的情况则立即回退。因此，该问题的难点是如何精准判断当前应采用哪个展开式，避免回退。一个基本思路是根据目标终结符决定当前应采用哪个生成式。为此，我们可以强制要求文法具备某些特性，如 LL (1) (Left-to-right, Left most, lookahead 1 symbol)。

3.2.1 LL(1) 文法

LL (1) 文法有两个基本要求，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。

3.2.1.1 左递归和消除

对一条语法规则来说，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 ($E \mapsto E \text{ OP1 } E1$)。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。

$$\begin{aligned}
 X &\mapsto X \langle a \rangle \mid X \langle b \rangle \mid \langle c \rangle \mid \langle d \rangle \\
 &\Downarrow \\
 X &\mapsto \langle c \rangle Y \mid \langle d \rangle Y \\
 Y &\mapsto \langle a \rangle Y \mid \langle b \rangle Y \mid \epsilon
 \end{aligned} \tag{3.1}$$

将该方法应用于图 3.3 中的文法，可消除其左递归问题。结果如图 3.4 所示。


<pre> [1] E → E OP1 E1 [2] E1 [3] E1 → E1 OP2 E2 [4] E2 [5] E2 → E3 OP3 E2 [6] E3 [7] E3 → NUM [8] <LPAR> E <RPAR> [9] NUM → <UNUM> [10] <SUB> <UNUM> [11] OP1 → <ADD> [12] <SUB> [13] OP2 → <MUL> [14] <DIV> [15] OP3 → <POW> </pre>	<p>消除左递归</p> 	<pre> [1] E → E1 E' [2] E' → OP1 E1 E' [3] ε [4] E1 → E2 E1' [5] E1' → OP2 E2 E1' [6] ε [7] E2 → E3 OP3 E2 [8] E3 [9] E3 → NUM [10] <LPAR> E <RPAR> [11] NUM → <UNUM> [12] <SUB> <UNUM> [13] OP1 → <ADD> [14] <SUB> [15] OP2 → <MUL> [16] <DIV> [17] OP3 → <POW> </pre>
---	--	---

图 3.4: 消除左递归后的 CFG 语法

3.2.1.2 无回溯语法

对于每个非终结符的任意两个生成式，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。当生成式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。

$$\begin{aligned}
 X &\xrightarrow{[i]} \langle a \rangle \dots \\
 X &\xrightarrow{[j]} \langle b \rangle \dots \\
 X &\xrightarrow{[k]} Y \dots \xrightarrow{[l]} \langle c \rangle \dots \\
 &\dots
 \end{aligned} \tag{3.2}$$

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。

$$\begin{aligned}
 X &\rightarrow \langle a \rangle A \mid \langle a \rangle B \mid \langle b \rangle \\
 &\Downarrow \\
 X &\rightarrow \langle a \rangle Y \mid \langle b \rangle \\
 Y &\rightarrow A \mid B
 \end{aligned} \tag{3.3}$$

3.2.2 解析算法实现

我们定义 $First(X \xrightarrow{[i]} \beta_1 \beta_2 \dots \beta_n)$ 表示 X 的第 i 条产生式的首字符集合。如果 $\epsilon \notin \beta_1$ ，则 $First(X) = First(\beta_1)$ ；如果 $\epsilon \in \beta_1 \& \dots \& \epsilon \in \beta_i$ ，则 $First(X) = First(\beta_{i+1})$ 。如果 $\epsilon \in \beta_1 \& \dots \& \epsilon \in \beta_n$ ，我们还需考

考虑 X 之后可能出现的字符 $Follow(X \xrightarrow{[i]} \dots)$ ，并据此决定是否采用该 $X \mapsto \epsilon$ 的生成式。因此我们使用 $First^+(X \xrightarrow{[i]} \beta)X$ 的第 i 条产生式的首字符集合（不含 ϵ ）。

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

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。

$$\forall 1 \leq i, j \leq n, First^+(A \rightarrow \beta_i) \cap First^+(A \rightarrow \beta_j) = \emptyset$$

表 3.1 展示了图 3.4 中文法规则的 $First$ 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 3.1: 记录每条生成式的 $First$ 集合。

	<UNUM>	<ADD>	<SUB>	<MUL>	<DIV>	<POW>	<LPAR>	<RPAR>	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 3.1 中的 ϵ 字符便可以得到 $First^+$ 或 LL(1) 解析表 3.2。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 3.2: LL(1) 解析表：生成式的 $First^+$ 集合。

	<UNUM>	<ADD>	<SUB>	<MUL>	<DIV>	<POW>	<LPAR>	<RPAR>
E	[1]		[1]				[1]	
E'		[2]	[2]					[3]
E1	[4]		[4]				[4]	
E1'		[6]	[6]	[5]	[5]			
E2	[7]		[7]				[7]	
E2'		[9]	[9]	[9]	[9]	[8]		
E3	[10]		[10]				[11]	
NUM	[12]		[13]					
OP1		[14]	[15]					
OP2				[16]	[17]			
OP3						[18]		

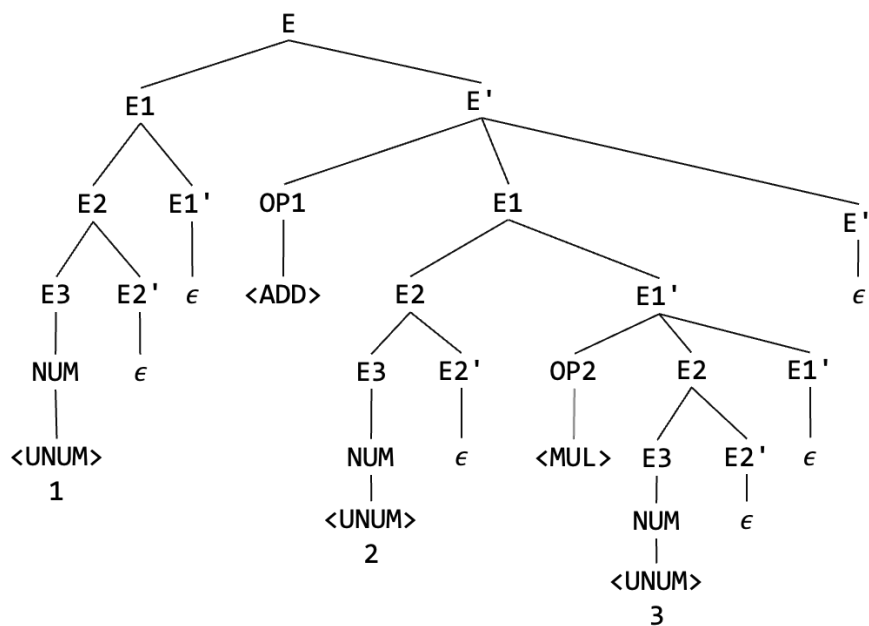


图 3.5: 使用 LL(1) 语法解析 $1+2*3$ 得到的语法解析树

基于 LL(1) 文法解析算式 $1+2*3$ 得到的语法解析树（图 3.5）。

3.3 练习

以开发正则表达式工具（输入任意的正则表达式，生成对应的正则匹配器）为目标：1) 设计用于解析正则表达式的 CFG 文法；2) 将其改写为 LL(1) 文法；3) 计算 LL(1) 解析表。