5 自底向上解析

徐辉，xuh@fudan.edu.cn

本章学习目标：

- ⋆⋆ 理解自底向上的语法分析思想，掌握移进-归约分析的基本过程
- ⋆⋆ 掌握 SLR 文法的解析方法，包括 SLR 分析表的构造与使用
- ⋆ 了解 LALR 和 LR(1) 文法的基本思想及其与 SLR 文法的区别

5.1 自底向上解析思想

自底向上解析是从输入串开始，通过不断进行规约操作，最终将其归约为文法开始符号的过程。本章主要介绍 LR（Left-to-right, Rightmost）自底向上解析方法，即从左到右扫描输入，并构造最右推导的逆过程。

LR 解析的基本操作包括：

- 移进（shift）：将输入串中的下一个词元读入解析栈；
- 规约（reduce）：根据产生式 $X \mapsto \beta$，将解析栈顶的 β 规约为 X。

该问题的难点在于：在解析过程中，某些状态下可能存在多种可选操作，例如既可以移进又可以规约，或存在多个规约选择，必须通过合适的分析方法进行选择。本章将以 SLR（Simple LR）文法为基础，详细介绍 LR 解析方法，并在此基础上进一步讨论 LALR 和 LR(1) 等扩展方法。

5.2 SLR 文法和解析

SLR 文法是指其对应的上下文无关文法在构造 SLR 分析表时不存在冲突，即解析表的每个单元格中至多包含一个分析动作。

对于一套 SLR 文法，其解析表的构建通常包括两个步骤：

- 构造 LR(0) 项目集规范族（canonical collection of LR(0) items）及其对应的有穷自动机；
  - LR(0) 项目（item）：在产生式右部插入一个点（$\circ$）得到的形式，如 $\texttt{A} \mapsto \alpha \circ \beta$，用于表示产生式右部的识别进度；
  - LR(0) 项目集（item set）：若干 LR(0) 项目构成的集合；在 LR 分析中，通常从某一项目出发，对待识别非终结符进行闭包展开，从而得到项目集。具体而言，若项目中存在 $\texttt{A} \mapsto \alpha \circ \texttt{B}\beta$，则需将所有形如 $\texttt{B} \mapsto \circ\gamma$ 的项目加入该项目集。
  - LR(0) 项目集规范族：从初始 LR(0) 项目进行闭包展开，并结合转移操作（Goto）逐步构造得到的全部项目集的集合。其中每一个 LR(0) 项目集对应一个 LR(0) 有穷自动机状态，每一个 Goto 操作表示状态转移关系。
- 根据 LR(0) 有穷自动机构造 SLR 解析表，即 Action-Goto 表。

下面以计算器文法为例，详细讲解 SLR 上述过程。

5.2.1 构造 LR(0) 有穷自动机

由于计算器语法中开始符号对应的产生式不唯一，为便于后续分析，我们在原文法基础上增加一条新的产生式 $\texttt{S} \mapsto \texttt{E}$。更新后的规则如语法 5.1 所示。

```
[0] S  ↦  E
[1] E  ↦  E OP1 E1
[2] E  ↦  E1
[3] E1  ↦  E1 OP2 E2
[4] E1  ↦  E2
[5] E2  ↦  E3 OP3 E2
[6] E2  ↦  E3
[7] E3  ↦  NUM
[8] E3  ↦  '(' E ')'
[9] NUM  ↦  <UNUM>
[10] NUM  ↦  '-' <UNUM>
[11] OP1  ↦  '+'
[12] OP1  ↦  '-'
[13] OP2  ↦  '*'
[14] OP2  ↦  '/'
[15] OP3  ↦  '^'
```
(5.1)

从初始 LR(0) 项目 $\texttt{S} \mapsto \circ\ \texttt{E}$ 开始，我们对其产生的符号进行闭包展开，可以得到一个初始的项目集，如表达式 5.2 所示，即 LR(0) 有穷自动机的初始状态。

$$
\begin{aligned}
\texttt{S} &\mapsto \circ\ \texttt{E} \\
\texttt{E} &\mapsto \circ\ \texttt{E OP1 E1} \\
\texttt{E} &\mapsto \circ\ \texttt{E1} \\
\texttt{E1} &\mapsto \circ\ \texttt{E1 OP2 E2} \\
\texttt{E1} &\mapsto \circ\ \texttt{E2} \\
\texttt{E2} &\mapsto \circ\ \texttt{E3 OP3 E2} \\
\texttt{E2} &\mapsto \circ\ \texttt{E3} \\
\texttt{E3} &\mapsto \circ\ \texttt{NUM} \\
\texttt{E3} &\mapsto \circ\ \texttt{'(' E ')'} \\
\texttt{NUM} &\mapsto \circ\ \texttt{<UNUM>} \\
\texttt{NUM} &\mapsto \circ\ \texttt{'-' <UNUM>}
\end{aligned}
\tag{5.2}
$$

算法 1 给出了上述 LR(0) 项目集的闭包构造过程。其中，输入集合 Q 表示一组核心项目（kernel items），通过不断展开 $\circ$ 后为非终结符的项目，最终得到完整的项目集。对于上述初始项目集来说，其核心项目是 $\texttt{S} \mapsto \circ\ \texttt{E}$。

算法 1 LR(0) 规范项集生成算法

```
procedure RegularSet(Q)
    hasChanged ← TRUE
    while hasChanged do
        hasChanged ← FALSE
        for each A ↦ β ∘ Cδ ∈ Q do
            for each C ↦ λ ∈ G do
                if C ↦ ∘λ ∉ Q then
                    Q ← Q ∪ {C ↦ ∘λ}
                    hasChanged ← TRUE
                end if
            end for
        end for
    end while
end procedure
```

LR(0) 有穷自动机中的每个状态对应一个项目集，边表示在文法符号上的状态转移关系。该自动机刻画了项目集之间在读入一个符号后的转移，即 Goto 操作。其构造过程从初始状态 S_0 开始，对每个项目集和可能的文法符号应用 Goto 操作，生成新的项目集；不断迭代该过程，直到不再产生新的项目集和状态转移关系为止。图 5.1 展示了语法规则 5.1 对应的 LR(0) 有穷自动机。

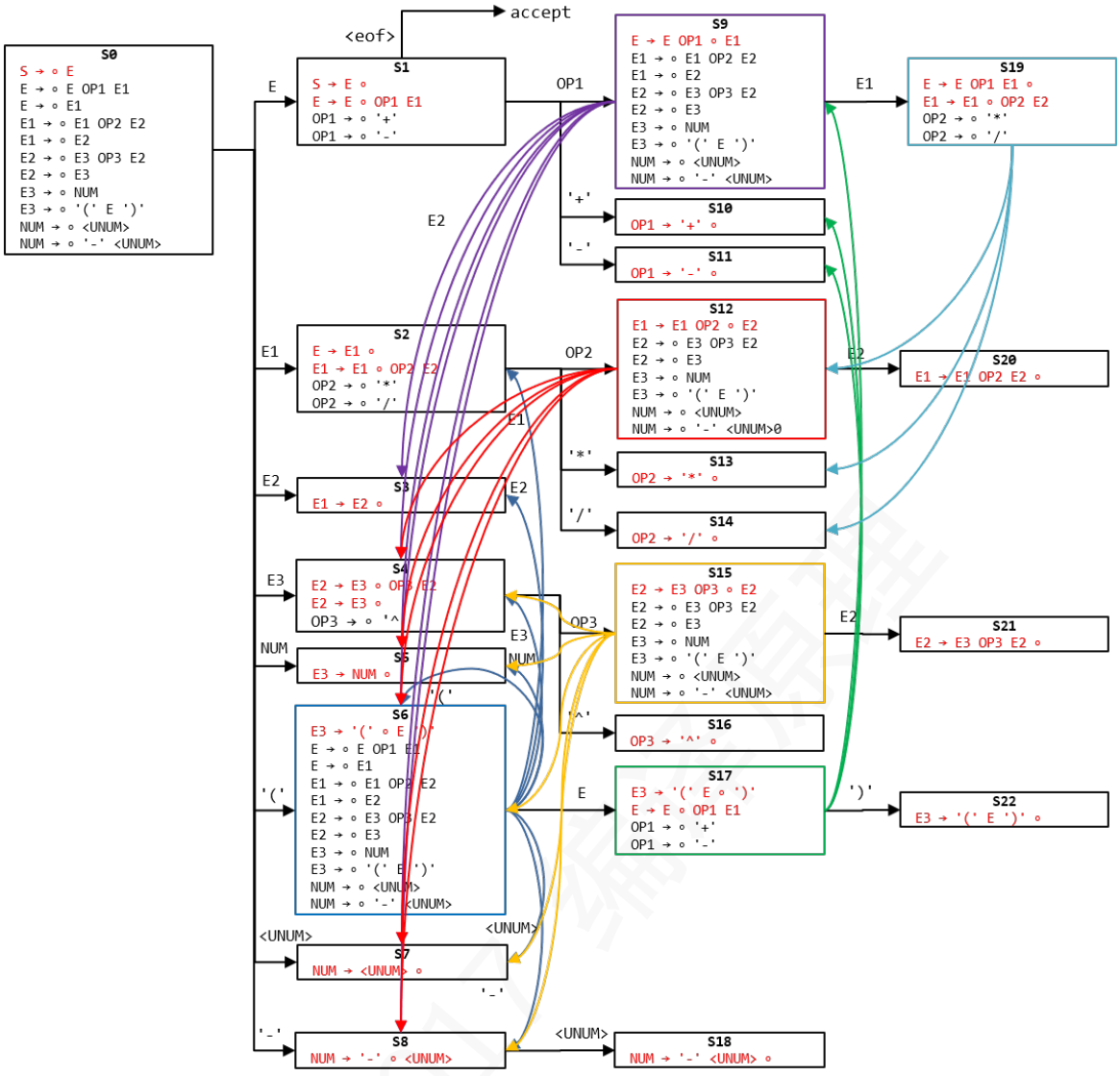


图 5.1: 语法规则 5.1 对应的 LR(0) 有穷自动机

5.2.2 创建 SLR 解析表

将 LR(0) 有穷自动机的状态转移关系转化为表格形式，即可得到 SLR 解析表。如表 5.1 所示，每一行对应一个 LR(0) 有穷自动机状态（即一个项目集），每一列对应一个文法符号，每个单元格表示在读入相应符号后转移到的目标状态。此外，对于包含形如 $\mathtt{A} \mapsto \alpha\circ$ 的项目的状态，还需要在表中填入规约动作。例如，R[2] 表示根据语法规则 [2] 进行规约，即将 $\mathtt{E} \mapsto \mathtt{E1}\circ$ 规约为 E。需要注意的是，规约操作并非在所有输入符号下均可执行，而仅在下一个输入符号属于 Follow(A) 时才允许。因此，每个状态的规约动作仅填入 SLR 解析表中对应的 Follow 集合的列中。

通常，根据文法符号是否为终结符，SLR 解析表分为两部分：Action 表（对应终结符）和 Goto 表（对应非终结符），其中规约操作仅出现在 Action 表中。

表 5.1: 语法规则 5.1 对应的 SLR 解析表

LR(0) 项目集	Goto								Action (Shift-Reduce)								
	E	E1	E2	E3	OP1	OP2	OP3	NUM	<UNUM>	'+'	'-'	'*'	'/'	'^'	'('	')'	eof
S0	S1	S2	S3	S4				S5	S7		S8				S6		
S1					S9					S10	S11						accept
S2						S12				R[2]	R[2]	S13	S14		R[2]	R[2]	
S3										R[4]	R[4]	R[4]	R[4]			R[4]	R[4]
S4							S15			R[6]	R[6]	R[6]	R[6]	S16		R[6]	R[6]
S5										R[7]	R[7]	R[7]	R[7]	R[7]		R[7]	R[7]
S6	S17	S2	S3	S4				S5	S7		S8				S6		
S7										R[9]	R[9]	R[9]	R[9]	R[9]		R[9]	R[9]
S8									S18								
S9		S19	S3	S4				S5	S7		S8				S6		
S10									R[11]		R[11]				R[11]		
S11									R[12]		R[12]				R[12]		
S12			S20	S4				S5	S7		S8				S6		
S13									R[13]		R[13]				R[13]		
S14									R[14]		R[14]				R[14]		
S15			S21	S4				S5	S7		S8				S6		
S16									R[15]		R[15]				R[15]		
S17					S9					S10	S11					S22	
S18										R[10]	R[10]			R[10]		R[10]	R[10]
S19						S12				R[1]	R[1]	S13	S14			R[1]	R[1]
S20										R[3]	R[3]	R[3]	R[3]			R[3]	R[3]
S21										R[5]	R[5]					R[5]	R[5]
S22										R[8]	R[8]	R[8]	R[8]	R[8]		R[8]	R[8]

5.2.3 应用 SLR 解析表

本节以算式 <UNUM(1)> '*' <UNUM(2)> 为例，演示 SLR 解析方法。解析过程中使用两个栈：状态栈和符号栈，分别用于记录当前分析状态和已识别的符号。在每一步中，根据状态栈栈顶状态以及当前输入符号，查 SLR 解析表以确定下一步操作。具体的解析过程如表 5.2 所示。

表 5.2: 应用 SLR 解析表 5.1 解析乘法算式 <UNUM(1)> '*' <UNUM(2)>。

状态栈	符号栈	待读入词元	操作
S0		<UNUM(1)> '*' <UNUM(2)> <eof>	shift <UNUM(1)>, Goto S7
S0,S7	<UNUM(1)>	'*' <UNUM(2)> <eof>	Reduce [9], back to S0, Goto S5
S0,S5	NUM	'*' <UNUM(2)> <eof>	Reduce [7], back to S0, Goto S4
S0,S4	E3	'*' <UNUM(2)> <eof>	Reduce [6], back to S0, Goto S3
S0,S3	E2	'*' <UNUM(2)> <eof>	Reduce [4], back to S0, Goto S2
S0,S2	E1	'*' <UNUM(2)> <eof>	Shift '*', Goto S13
S0,S2,S13	E1 '*'	<UNUM(2)> <eof>	Reduce [13], back to S2, Goto S12
S0,S2,S12	E1 OP2	<UNUM(2)> <eof>	Shift <UNUM(2)>, Goto S7
S0,S2,S12,S7	E1 OP2 <UNUM(2)>	<eof>	Reduce [9], back to S12, Goto S5
S0,S2,S12,S5	E1 OP2 NUM	<eof>	Reduce [7], back to S12, Goto S4
S0,S2,S12,S4	E1 OP2 E3	<eof>	Reduce [6], back to S12, Goto S20
S0,S2,S12,S20	E1 OP2 E2	<eof>	Reduce [3], back to S0, Goto S2
S0,S2	E1	<eof>	Reduce [2], back to S0, Goto S1
S0,S1	E	<eof>	accept

5.3 更多文法

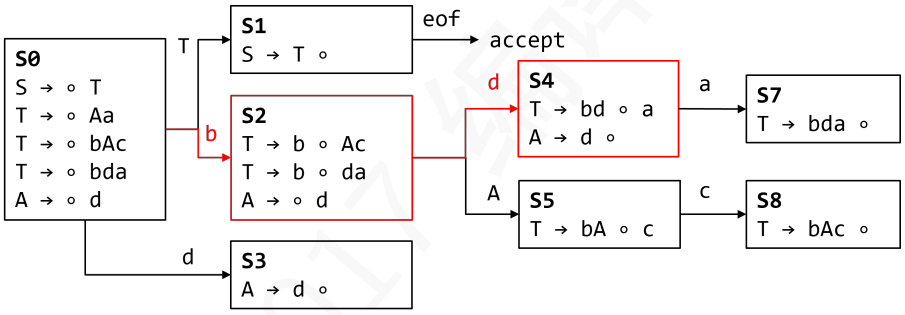
5.3.1 LR(1) 文法

SLR 方法的表达能力较为有限。当构造的 SLR 解析表中某些单元格存在多个分析动作（即出现移进-规约冲突或规约-规约冲突）时，对应的文法就不是 SLR 文法。例如，下列语法规则（文法 5.3）不属于 SLR 文法。

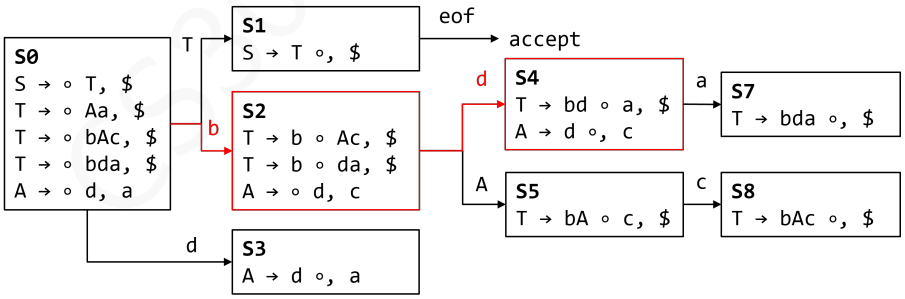
$$\begin{aligned} \texttt{T} &\mapsto \circ\ \texttt{A}\ \texttt{a} \\ \texttt{T} &\mapsto \circ\ \texttt{b}\ \texttt{A}\ \texttt{c} \\ \texttt{T} &\mapsto \circ\ \texttt{b}\ \texttt{d}\ \texttt{a} \\ \texttt{A} &\mapsto \circ\ \texttt{d} \end{aligned} \tag{5.3}$$

图 5.2a 构造了文法 5.3 对应的 LR(0) 有穷自动机。可以看到，在状态 S_4 中（已移进符号 b 和 d），当下一个输入符号为 a 时，同时存在移进和规约两种可能的分析动作，从而产生移进-规约冲突。

对该情况进行分析可以发现，应选择移进操作作为正确的分析动作。这是因为，若按照项目 $\texttt{A} \mapsto \texttt{d}\circ$ 进行规约，则该产生式在原文法中的使用场景要求其后继符号为 c，而不是 a。然而，SLR 方法在确定规约操作时仅依赖于 Follow(A) 集合，其粒度较粗，无法区分当前项目的具体上下文来源，从而导致冲突无法消除。



(a) LR(0) 有穷自动机：解析输入串 “bda” 时存在移进规约冲突



(b) LR(1) 有穷自动机：可以有效解析输入串 “bda”

图 5.2: LR(1)，但非 SLR 文法举例

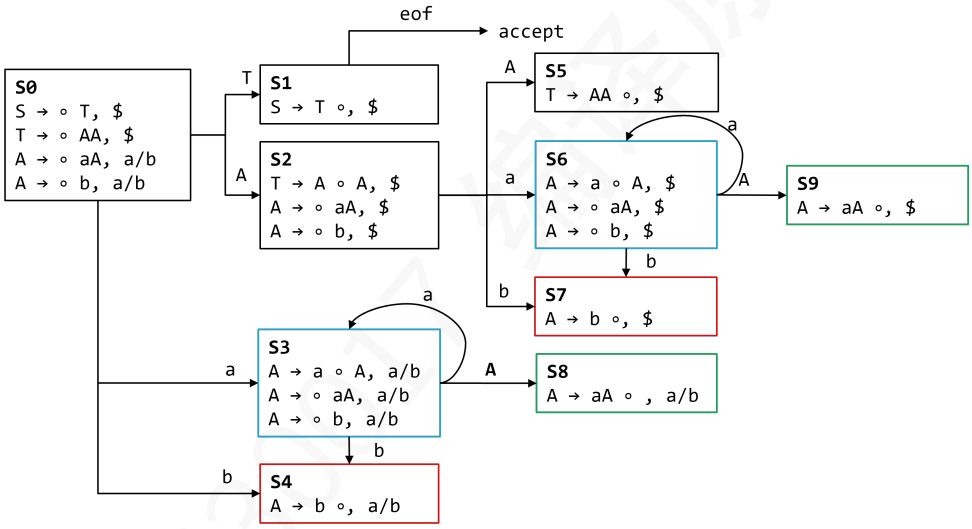
LR(1) 分析方法能够解决上述问题。相比于 SLR 方法，其改进之处在于在构造 LR(1) 有穷自动机时，为每个项目引入一个向前看符号，从而携带更精确的上下文信息。如图 5.2b所示，LR(1) 项目形如 $[\texttt{A} \mapsto \alpha \circ \beta, \texttt{a}]$，其中 a 表示当前项目在后续规约时所允许的输入符号。通过这种方式，可以更精确地限定规约操作的适用范围，从而避免 SLR 方法中由于 Follow 集合过于粗糙而引发的冲突。在构造有穷自动机时，若两个项目集中的项目在向前看符号上存在差异，则将其视为不同的状态，从而避免潜在的分析冲突。

5.3.2 LALR 文法

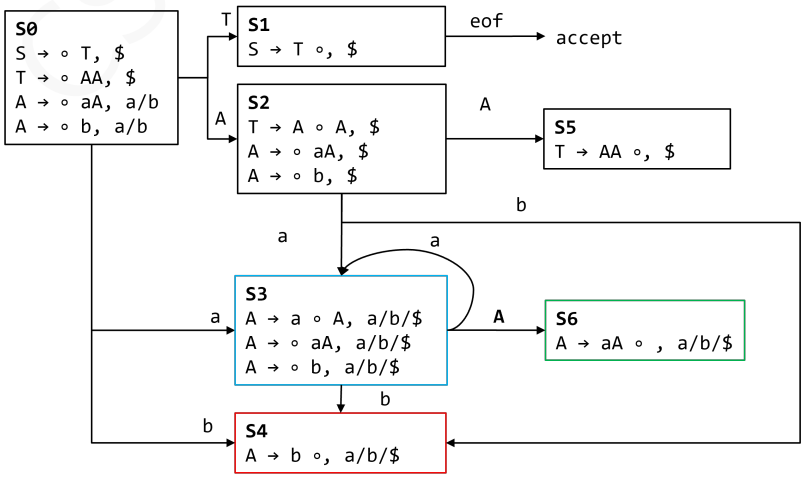
LR(1) 分析方法 [1] 通过在项目中引入前瞻符号，从而避免解析表中的冲突。然而，其缺点是构造得到的项目集族规模较大，相比 SLR 可能会显著增加，尤其在文法较复杂时更为明显。为减小 LR(1) 项目集族规模增长带来的开销，LALR 方法将具有相同核心项的 LR(1) 项目集合并，即在保持项目集内容相同的前提下，合并其前瞻符号集合。因此，LALR 在分析能力上接近 LR(1)，但状态数量大大减少，同时相比 SLR 在上下文信息的利用上更加精细。

$$
\begin{aligned}
\texttt{S} &\mapsto \circ\ \texttt{T} \\
\texttt{T} &\mapsto \circ\ \texttt{A}\ \texttt{A} \\
\texttt{A} &\mapsto \circ\ \texttt{a}\ \texttt{A} \\
\texttt{A} &\mapsto \circ\ \texttt{b}
\end{aligned}
\tag{5.4}
$$

文法 5.4 展示了一套语法规则。按照 LR(1) 解析表的构造过程，可以得到图 5.3a 所示的 LR(1) 项目集族及其对应的有穷自动机。可以看到，其中存在三对具有相同核心项的项目集：S_3 和 S_6，S_4 和 S_7，以及 S_8 和 S_9。将这些核心项目相同的项目集进行合并，即可得到图 5.3b 所示的 LALR 项目集族。



(a) LR(1) 项目集族对应的有穷自动机



(b) LALR 项目集族对应的自动机

图 5.3: LALR 文法举例

什么情况下 LR(1) 项目集可以合并，即合并后构造的语法分析表仍不存在冲突？通过对该案例的分析可以发现，其关键在于避免合并后引入新的分析冲突。具体而言，首先，合并后的项目集中不存在两个不同的规约项目，从而避免规约-规约冲突；其次，对于包含规约项目的状态，要么不存在移进操作，要么其规约项目的前瞻符号集合与移进符号集合互不相交，从而避免移进-规约冲突。

5.3.3 应用探讨

在实际应用中，LR(1) 文法已经具有较强的表达能力，许多语法分析工具基于 LR(1) 或其变种进行实现。例如，Rust 中的 lalrpop 库[1]主要采用 LR(1) 分析方法。尽管 LR(1) 分析方法能力较强，但仍不能处理所有上下文无关文法。在需要更高通用性的场景中，可以采用 GLR（Generalized LR）方法 [2]。该方法在出现分析冲突时同时保留多个可能的分析路径，并进行并行推进，从而能够处理任意上下文无关文法。GLR 方法可以视为对 LR 分析方法的推广，并可与 LALR、LR(1) 等分析框架结合使用。此外，还存在其它提升分析能力的思路。例如，通过增加向前看符号的数量可以得到 LR(k) 文法 [3]，但其实现复杂度也随之显著增加；或者采用基于动态规划思想的 CYK 算法 [4]。

参考文献

[1] Frank DeRemer, and Thomas Pennello. ”Efficient computation of LALR (1) look-ahead sets.” ACM Transactions on Programming Languages and Systems (TOPLAS), 1982.

[2] Masaru Tomita. “An efficient context-free parsing algorithm for natural languages.” *In International Joint Conference on Artificial Intelligence (IJCAI)*, 1985.

[3] Donald E. Knuth. “On the translation of languages from left to right.” *Information and Control*, 1965.

[4] Daniel H. Younger. “Recognition and parsing of context-free languages in time n3.” *Information and Control*, 1967.

[1]lalrpop: https://lalrpop.github.io/lalrpop/

练习

1) (多选题) 下列文法属于:

(a) LL(1)

(b) SLR

$$
\begin{aligned}
\texttt{S} &\mapsto \texttt{S A} \\
\texttt{S} &\mapsto \texttt{A} \\
\texttt{A} &\mapsto \texttt{a}
\end{aligned}
\tag{5.5}
$$

2) 已知以下上下文无关文法规则，分析该文法是否为 SLR 文法。如果是，请为其构造 SLR 解析表。

$$
\begin{aligned}
\texttt{Regex} &\mapsto \texttt{Regex '|' Concat} \\
\texttt{Regex} &\mapsto \texttt{Concat} \\
\texttt{Concat} &\mapsto \texttt{Concat Closure} \\
\texttt{Concat} &\mapsto \texttt{Closure} \\
\texttt{Closure} &\mapsto \texttt{Closure '*'} \\
\texttt{Closure} &\mapsto \texttt{Item} \\
\texttt{Item} &\mapsto \texttt{'(' Regex ')'} \\
\texttt{Item} &\mapsto \texttt{<Char>}
\end{aligned}
\tag{5.6}
$$

3) 修改文法 5.3，分别达到以下几个目标:

1) 在 SLR 解析表中引入规约-规约冲突;

2) 在 LR(1) 解析表中引入移进-规约冲突;

3) 在 LR(1) 解析表中引入规约-规约冲突。

4) LL(1) 文法一定符合 SLR 文法吗?