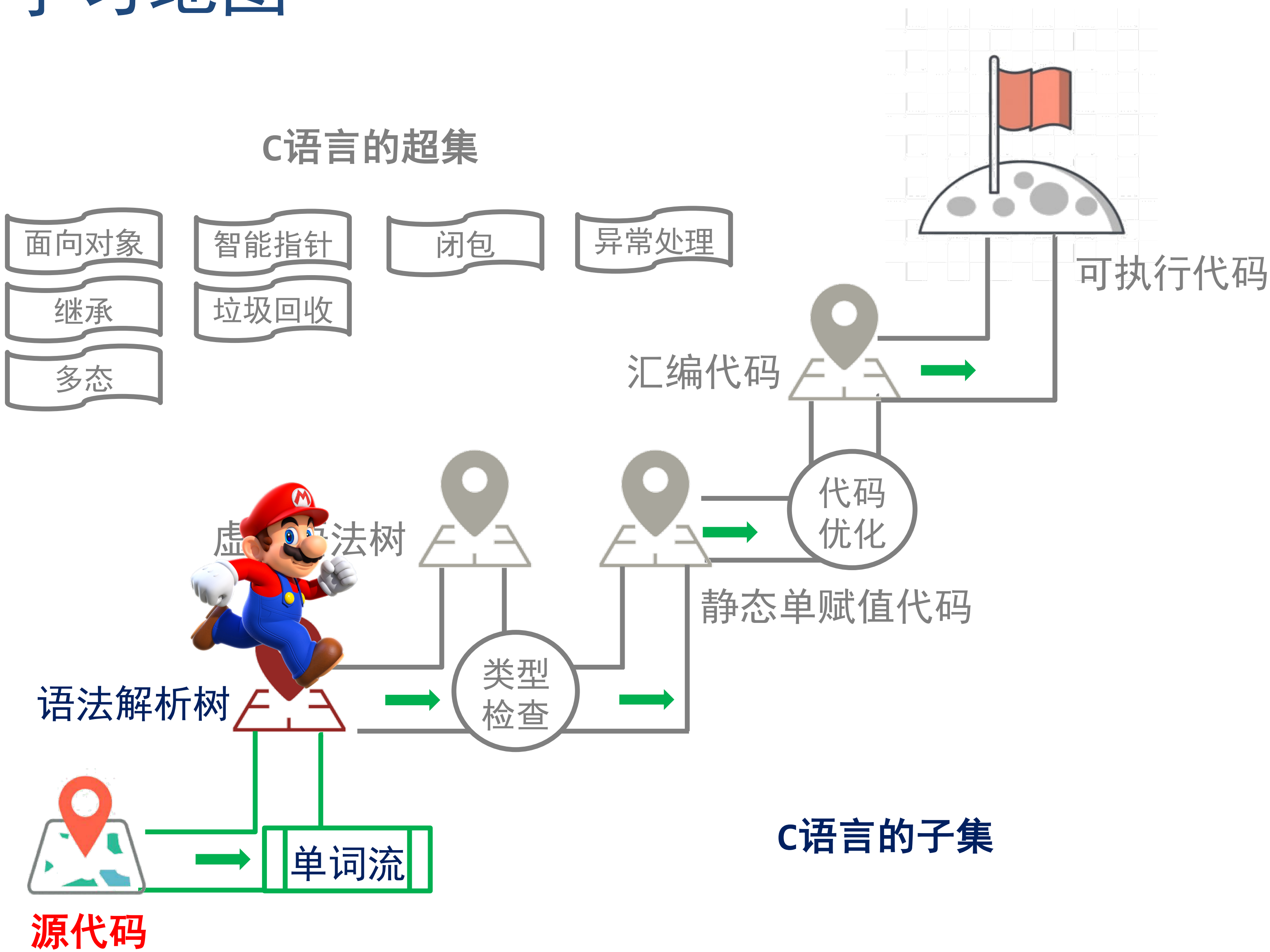
Lecture 3

句式分析

徐 辉

xuh@fudan.edu.cn

学习地图



C语言的超集
面向对象
智能指针
闭包
异常处理
继承
垃圾回收
多态
可执行代码
汇编代码
代码优化
虚
法树
静态单赋值代码
类型检查
语法解析树
单词流
源代码
C语言的子集

大纲

一、句式分析的基本概念

二、自顶向下分析

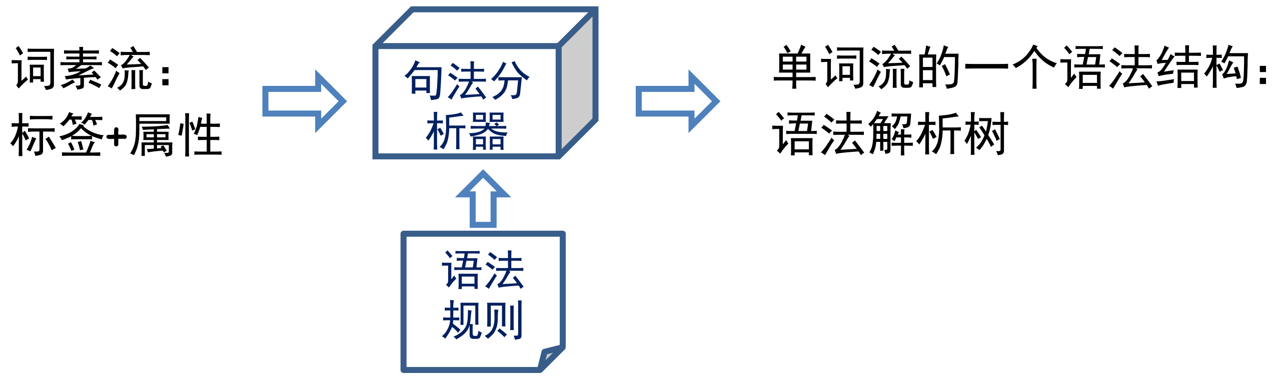
三、自底向上分析

四、语法分析工具

一、句式分析的基本概念

问题定义

- 给定一个句子和语法规则，找到可生成该句子的一个语法推导。
- 通过词法分析已经将句子转换为了标签流。
- 语法规则（Grammar）定义了：
  - 什么是语法分析器（parser）可接受的标签组成，
  - 及其语法推导方式。



基本概念

- 一门语言（language）是多个句子（sentences）的集合。
- 句子（sentence）是由终结符（terminal symbols）组成的序列（sequence）。
- 字符串（string）是包含终结符和非终结符的序列。
  - 字符串符号：α, β, γ
  - 非终结符：X, Y, Z
  - 终结符（标签）：a, b, c
- 一条语法（grammar）包括一个开始符号S和多条推导规则（productions）
  - $\alpha \rightarrow \beta$。

语法推导

- 语法G的语言$L(G)$是该语法可推导的所有句子的集合。
- 问题：下列语法是否可推导出句子$aaabbbccc$？

语法规则

[1] $S \rightarrow aBSc$
[2] $S \rightarrow abc$
[3] $Ba \rightarrow aB$
[4] $Bb \rightarrow bb$

推导

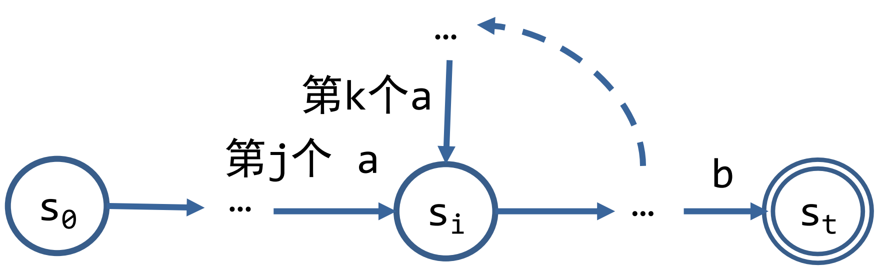
[1] $S \rightarrow aBSc$
[1] $S \rightarrow aBaBScc$
[3] $S \rightarrow aaBBScc$
[2] $S \rightarrow aaBBabccc$
[3] $S \rightarrow aaBaBbccc$
[3] $S \rightarrow aaaBBbccc$
[4] $S \rightarrow aaaBbbccc$
[4] $S \rightarrow aaabbbccc$

语法表示：使用正则表达式？

- 正则表达式是否可识别四则运算？
  - $y = a \times x + b$
    - $(var|num)\big((+|-|\times|\div)\ (var|num)\big)^*$
  - $y = a \times (x + b)$
    - $('('|var|num)\big((+|-|\times|\div)\ ('('|var|num|')')')\big)^*$
    - 可导致单词流被错误接收：
      - $y = (a \times (x + b)$
      - $y = (a \times (x + (b)$
- 正则表达式不能处理括号匹配问题：$(^*)^*$

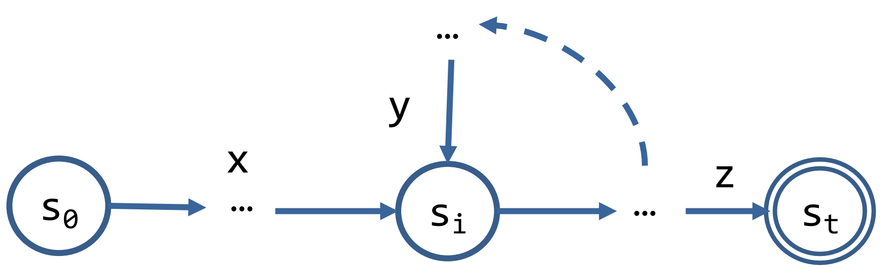
非正则语言

- 不能用正则表达式或有穷自动机表示的语言。
  - 正则语言不能计数，如$L = \{a^n b^n, n > 0\}$
  - 证明：
    - 假设DFA可识别该语言，其包含p个状态；
    - 假设某词素为$a^q b^q, q > p$。
    - 识别该词素需要经过某状态s_i至少两次，分别对应第j和第k个a；
    - 该DFA可同时接受$a^q b^q$和$a^{q+k-j} b^q$，推出矛盾。



正则语言的泵引理（Pumping Lemma）

- 词素数量有限的语言一定是正则语言。
- 词素数量无穷多的语言是否为正则语言？
- 某语言$L(r)$是正则语言的必要条件：
  - 任意长度超过p（泵长）的句子都可以被分解为xyz的形式
  - 其中x和z可为空，
  - 子句y被重复任意次（如$xyyz$）后得到的句子仍属于该语言。

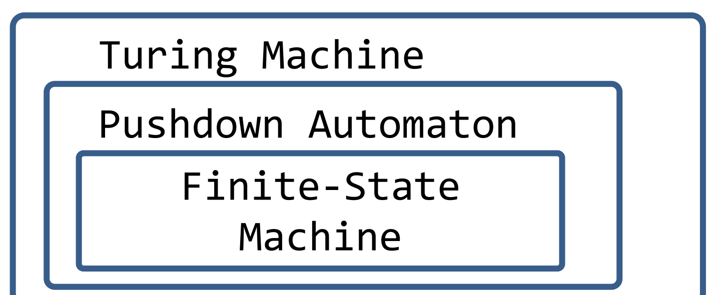


语言分析问题难度

- 通常来说，判断一个句子是否属于某个语言$w \in L(G)$是不能计算的。

Chomsky Hierarchy

Class	Languages	Automaton	Rules	Word Problem	Example
type-0	recursively enumerable	Turing machine	no restriction	undecidable	Post's corresp. problem
type-1	context sensitive	linear-bounded TM	$\alpha \to \gamma$ $\lvert\alpha\rvert \le \lvert\gamma\rvert$	PSPACE-complete	$a^n b^n c^n$
type-2	context free	pushdown automaton	$A \to \gamma$	cubic	$a^n b^n$
type-3	regular	NFA / DFA	$A \to a$ or $A \to aB$	linear time	$a^* b^*$



上线文无关语法和BNF范式

- 上下文无关语法（CFG/context-free grammar）是一个四元组(T, NT, S, P)
  - T：终结符
  - NT：非终结符
  - S：起始符号
  - P：产生式规则集合$X \rightarrow \gamma$，
    - X 是非终结符
    - γ 是可能包含终结符和非终结符的字符串
- BNF范式（Backus-Naur form）是传统的上下文无关语法表示方法。

$$\begin{aligned} \langle SheepNoise \rangle ::= &\ baa\ \langle SheepNoise \rangle \\ &\mid baa \end{aligned}$$

上线文无关语法举例

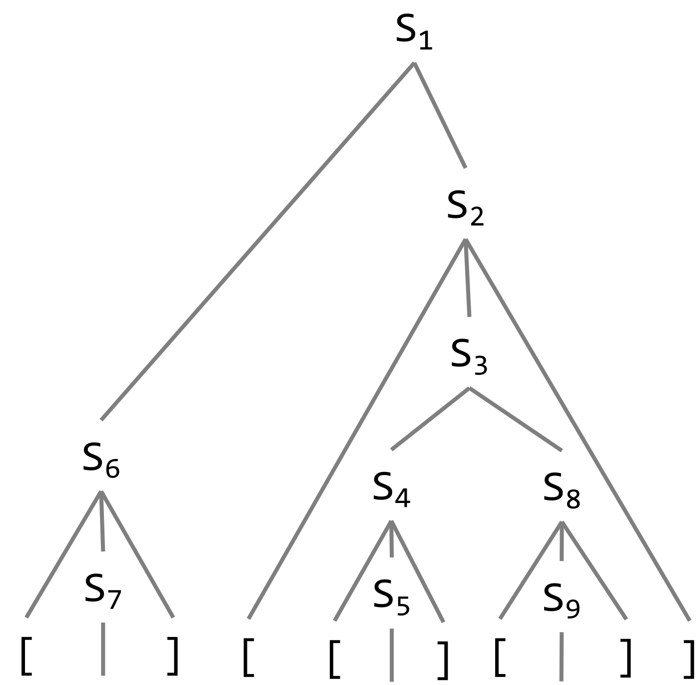
- 给定可生成所有匹配括号对的语法，[ ][[ ][ ]]是该语法的一个推导吗?

语法规则

$$
\begin{array}{lll}
[1] & S \rightarrow \epsilon \\
[2] & \quad\mid [S] \\
[3] & \quad\mid SS
\end{array}
$$

推导

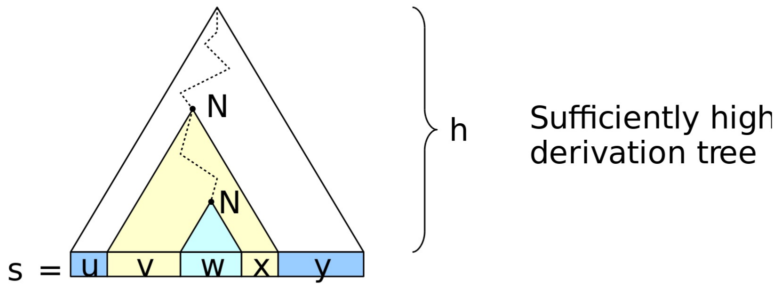
$$
\begin{array}{ll}
[3] & S \rightarrow SS \\
[2] & S \rightarrow S[S] \\
[3] & S \rightarrow S[SS] \\
[2] & S \rightarrow S[S[S]] \\
[1] & S \rightarrow S[S[\ ]] \\
[2] & S \rightarrow S[[S][\ ]\ ] \\
[1] & S \rightarrow S[[\ ][\ ]] \\
[2] & S \rightarrow [S][[\ ][\ ]] \\
[1] & S \rightarrow [\ ][[\ ][\ ]]
\end{array}
$$

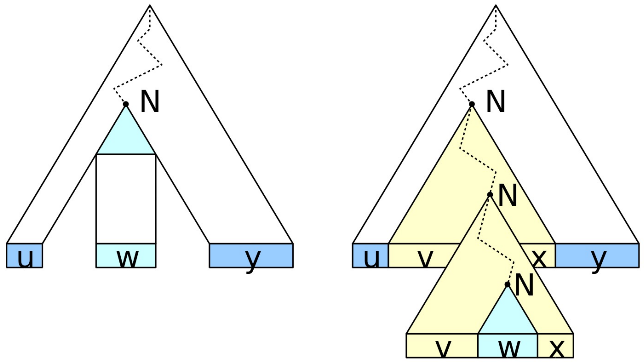


语法解析树

非CFG语言

- $L = \{a^n b^n c^n, n > 0\}$不是CFG语言
- CFG语言的泵引理：
  - 任意长度超过p（泵长）的句子可以被拆分为$uvwxy$，
  - 子句v和x被重复任意次后得到的新句子（如$uvvwxxy$）仍属于该语言。
  - 正则属于CFG：$uv^n w \epsilon^n \epsilon$





Generating uv^{0}wx^{0}y Generating uv^{2}wx^{2}y

推导（Derivation）的优先级

[1] $Expr \rightarrow (Expr)$
[2] $\quad\quad\quad\ \mid Expr\ Op\ name$
[3] $\quad\quad\quad\ \mid num$
[4] $Op \rightarrow +$
[5] $\quad\quad \mid -$
[6] $\quad\quad \mid \times$
[7] $\quad\quad \mid \div$

(a + b) × 2

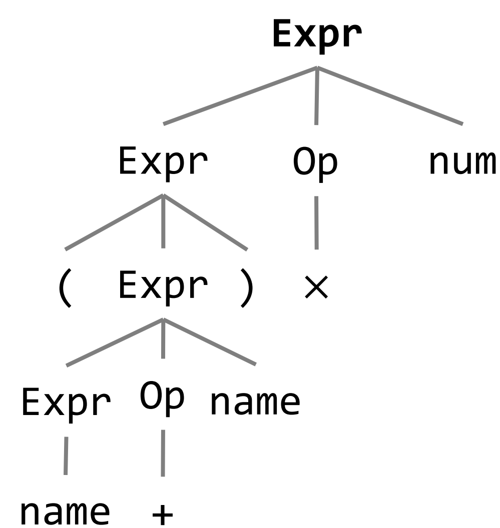
右侧优先推导
（Rightmost Derivation）

$Expr$
[2] $\underset{rm}{\Rightarrow} Expr\ Op\ num$
[6] $\underset{rm}{\Rightarrow} Expr \times num$
[1] $\underset{rm}{\Rightarrow} (Expr) \times num$
[2] $\underset{rm}{\Rightarrow} (Expr\ Op\ num) \times num$
[4] $\underset{rm}{\Rightarrow} (Expr + num) \times num$
[3] $\underset{rm}{\Rightarrow} (num + num) \times num$

(a + b) × 2

$Expr$
[2] $\underset{lm}{\Rightarrow} Expr\ Op\ num$
[1] $\underset{lm}{\Rightarrow} (Expr)\ Op\ num$
[2] $\underset{lm}{\Rightarrow} (Expr\ Op\ num) Op\ num$
[3] $\underset{lm}{\Rightarrow} (num\ Op\ num)\ Op\ num$
[4] $\underset{lm}{\Rightarrow} (num + num)\ Op\ num$
[6] $\underset{lm}{\Rightarrow} (num + int) \times num$

左侧优先推导
（Leftmost Derivation）



语法解析树完全相同

练习：语法推导

给定语法 $S \rightarrow SS + \mid SS * \mid a$，和字符串$aa + a *$

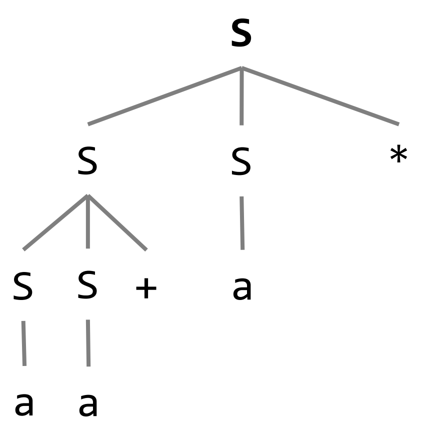
1） 写出左推导

2） 写出右推导

3） 写出语法推导树

$$S \underset{lm}{\Rightarrow} SS * \underset{lm}{\Rightarrow} SS + S * \underset{lm}{\Rightarrow} aS + S * \underset{lm}{\Rightarrow} aa + S * \underset{lm}{\Rightarrow} aa + a *$$

$$S \underset{rm}{\Rightarrow} SS * \underset{rm}{\Rightarrow} Sa * \underset{rm}{\Rightarrow} SS + a * \underset{rm}{\Rightarrow} Sa + a * \underset{rm}{\Rightarrow} aa + a *$$



练习：语法设计

- 分析下列语言是否为正则或CFG语言并设计语法。
  1) 所有0和1组成的字符串，每一个0后面紧跟着若干个1
  2) 所有0和1组成的字符串，0和1的个数相同
  3) 所有0和1组成的字符串，0和1的个数不相同

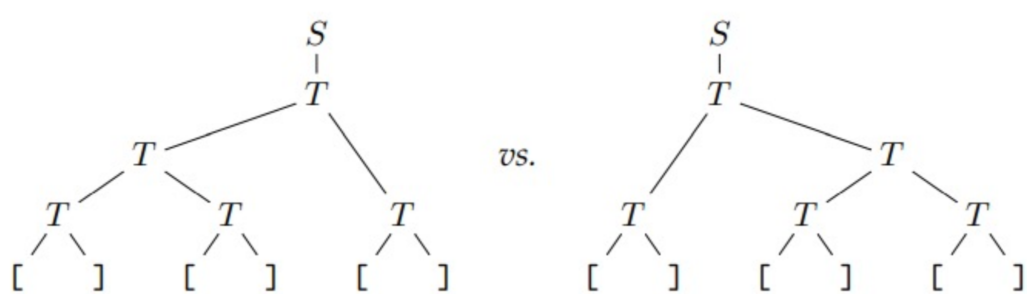
二义性（ambiguity）

- 如果L(G)中的某个句子有一个以上的最左（或最右）推导，那么语法G就有二义性。
  - 语法解析树不同
- 根据下列语法规则如何推导出[ ][ ][ ]?

[1]	$S \to \epsilon$
[2]	$S \to T$
[3]	$T \to []$
[4]	$T \to [T]$
[5]	$T \to TT$

$$S \to T \to TT \to TTT \to [\,]TT \to T[\,][\,] \to [\,][\,][\,]$$

$$S \to T \to TT \to [\,]T \to [\,]TT \to [\,][\,]T \to [\,][\,][\,]$$



极端情况

- 存在无数棵语法解析树
  - 考虑循环的情况
- 根据下列语法规则如何推导出[ ][[ ][ ]]?

[1] $S \rightarrow \epsilon$
[2] $S \rightarrow [S]$
[3] $S \rightarrow SS$

$$S \rightarrow SS \rightarrow [S]S \rightarrow [\ ]S \rightarrow [\ ][S] \rightarrow [\ ][S\ S] \rightarrow [\ ]\big[[S]\ [S]\big] \rightarrow [\ ][[\ ][\ ]]$$

$$S \rightarrow SS \rightarrow S \rightarrow SS \rightarrow \cdots$$

消除二义性

[1] $S \to \epsilon$
[2] $S \to [S]$
[3] $S \to SS$

⇒

[1] $S \to \epsilon$
[2] $S \to T$
[3] $T \to []$
[4] $T \to [T]$
[5] $T \to TT$

⇒

[1] $S \to \epsilon$
[2] $S \to T$
[3] $T \to UT$
[4] $T \to U$
[5] $U \to [\,]$
[6] $U \to [T]$

消除循环引起的二义性

$$S \to SS \to S$$

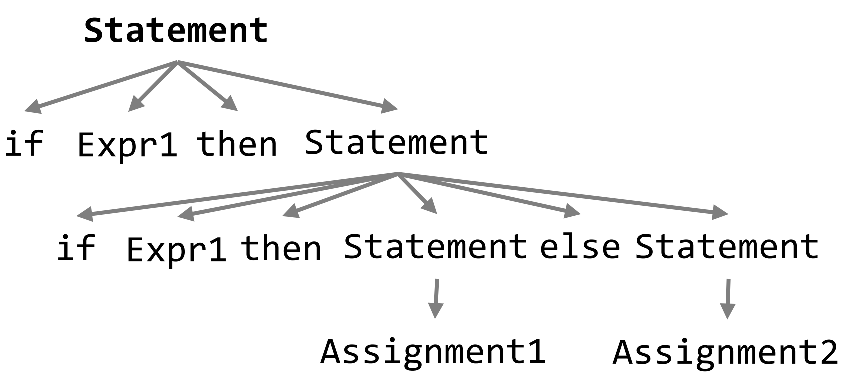
$T \to TT$左递归会引起二义性

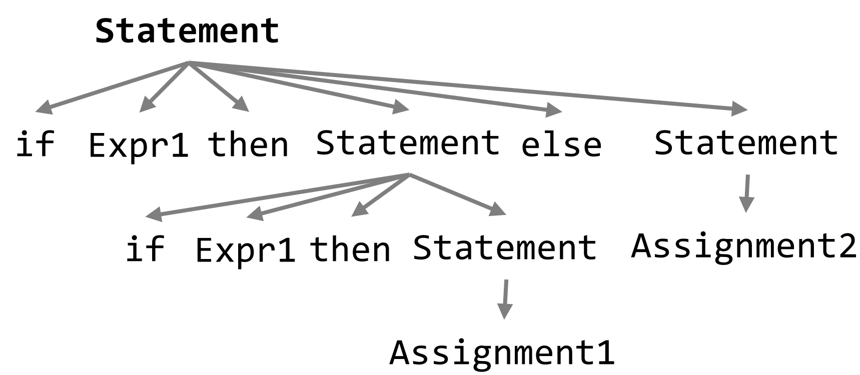
推导[ ][ ][ ]的例子

If-Else嵌套的二义性语法

[1] *Statment* → if *Expr* then *Statement* else *Statement*
[2] | if *Expr* then *Statement*
[3] | *Assignment*
[4] | …

```
if Expr1 then if Expr2 then Assignment1 else Assignment2
```






```
if Expr1 then
  if Expr2 then
    Assignment1
  else
    Assignment2
```

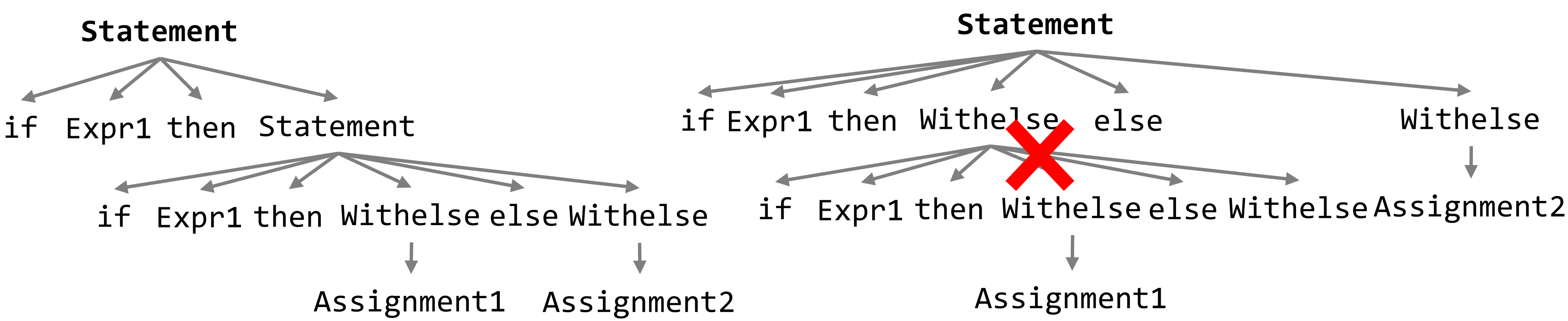
```
if Expr1 then
  if Expr2 then
    Assignment1
else
    Assignment2
```

消除If-Else语法的二义性

- 将语义编码加入到结构中

[1] $Statment \rightarrow$ if $Expr$ then $Withelse$ else $Statement$
[2] $\quad\quad\quad\quad\quad\quad$ | if $Expr$ then $Statement$
[3] $\quad\quad\quad\quad\quad\quad$ | $Assignment$
[4] $Withelse \rightarrow$ if $Expr$ then $Withelse$ else $Withelse$
[5] $\quad\quad\quad\quad\quad\quad$ | $Assignment$

`if Expr1 then if Expr2 then Assignment1 else Assignment 2`




```
if Expr1 then
  if Expr2 then
    Assignment1
  else
    Assignment2
```

不存在其它推导方式

四则运算的例子

$$
\begin{array}{ll}
[1] & Expr \rightarrow Expr + Expr \\
[2] & \quad\quad\quad |\ Expr\ - Expr \\
[3] & \quad\quad\quad |\ Expr \times Expr \\
[4] & \quad\quad\quad |\ Expr \div Expr \\
[5] & \quad\quad\quad |\ num \\
[6] & \quad\quad\quad |\ (Expr)
\end{array}
$$

$3 + 4 \times 5$的语义？

$Expr$
$\rightarrow Expr + Expr$
$\rightarrow Expr + Expr \times Expr$

$3 + (4 \times 5)$

$Expr$
$\rightarrow Expr \times Expr$
$\rightarrow Expr + Expr \times Expr$

$(3 + 4) \times 5$

优先级：（ ）>×/÷>+/−

$$
\begin{array}{ll}
[1] & Expr \rightarrow Expr + Term \\
[2] & \quad\quad\quad\quad |\ Expr\ - Term \\
[3] & \quad\quad\quad\quad |\ Term \\
[4] & Term \rightarrow Term \times Factor \\
[5] & \quad\quad\quad\quad |\ Term \div Factor \\
[6] & \quad\quad\quad\quad |\ Factor \\
[7] & Facor \rightarrow (Expr) \\
[8] & \quad\quad\quad\quad |\ num \\
[9] & \quad\quad\quad\quad |\ name
\end{array}
$$

$Expr$
$\rightarrow Expr + Term$
$\rightarrow Expr + Term \times Factor$

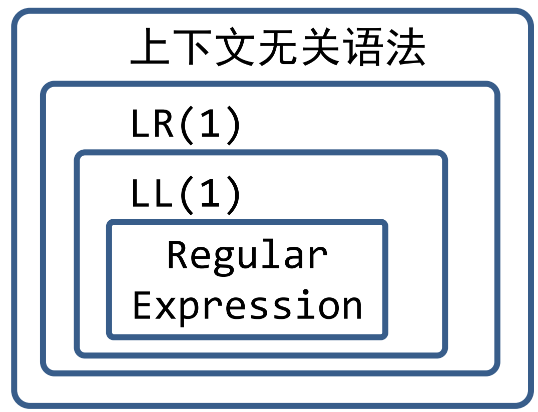
练习：二义性分析

- 下列If-Else语法是否存在二义性？

[1] $Statment \rightarrow \text{if } Expr \text{ then } Statement$
[2] $\quad\quad\quad\quad | \ matchedStatement$
[3] $matchedStatement \rightarrow \text{if } Expr \text{ then } matchedStatement \text{ else } Statement$
[4] $\quad\quad\quad\quad | \ Assignment$

编译器的任务：找到语法树推导

- 方法：
  - 自顶向下（top-down parser）
  - 自底向上（bottom-up parser）
- 语法难度：CFG>LR(1)>LL(1)>RE
  - 任意CFG需要花费更多时间进行语法分析
    - Earley/CYK算法复杂度$O(n^3)$
  - LL(1)是LR(1)的一个子集
    - Left-to-Right，Leftmost
    - 前瞻单词1个
    - 适合自顶向下分析
  - LR(1)是无歧义CFG的一个子集
    - Left-to-Right，Rightmost
    - 前瞻单词1个
    - 适合自底向上分析



二、自顶向下分析

自顶向下搜索

```
输入: 程序单词流 seq;
      CFG语法 rules;
Output: accept: 语法解析树ptree,
        reject;
初始化:
let ptree = start symbol;
let ptr = root;
let st = stack();
st.push(null);

开始:
let word = seq.NextWord();
While (true) do:
  if (!ptr.nodetype().isTerminal())
    For each rule in {A → β_1, …, β_n; A → ⋯}
      ptr.children = (β_1, …, β_n);
      For 1 < i < n+1
        st.push(β_{n+2−i});
      ptr = β_1;
  //ptr.nodetype()=terminal
  else if (word == ptr) //单词匹配成功
    word = seq.NextWord(); //下一个单词
    ptr = st.pop()
  else if (word == eof && cur == null)
    accept and return ptree;
  else
    backtrack(); //回溯
```

不考虑递归的情况：

$$A \rightarrow \cdots \rightarrow A$$

复杂度高：$l \times m \times n$

- l：单词个数
- m：规则个数
- n：每个规则生产的符号数

应用举例

(3+4)×5

(num+num)×num

$$
\begin{aligned}
[1]\ & Expr \rightarrow Expr + Term \\
[2]\ & \quad\quad\quad\ \mid Expr\ - Term \\
[3]\ & \quad\quad\quad\ \mid Term
\end{aligned}
$$

$$
\begin{aligned}
[4]\ & Term \rightarrow Term \times Factor \\
[5]\ & \quad\quad\quad\ \mid Term \div Factor \\
[6]\ & \quad\quad\quad\ \mid Factor
\end{aligned}
$$

$$
\begin{aligned}
[7]\ & Facor \rightarrow (Expr) \\
[8]\ & \quad\quad\quad\ \mid num \\
[9]\ & \quad\quad\quad\ \mid name
\end{aligned}
$$

假设每次都能选对规则

word	cur	Rule	Stack
(	Expr	[3]	Term
(	Term	[4]	Term, ×, Factor
(	Term	[6]	Factor, ×, Factor
(	Factor	[7]	(, Expr,), ×, Factor
(	(	-	Expr,), ×, Factor
num	Expr	[1]	Expr,+, Term,), ×, Factor
num	Expr	[3]	Term,+, Term,), ×, Factor
num	Term	[6]	Factor,+, Term,), ×, Factor
num	Term	[8]	num,+, Term,), ×, Factor
num	num	-	+, Term,), ×, Factor
+	+	-	Term,), ×, Factor
num	Term	[6]	Factor,), ×, Factor
num	Factor	[8]	num,), ×, Factor
num	num	-	), ×, Factor
)	)	-	×, Factor
×	×	-	Factor
num	Factor	[8]	num
num	num	-	null
eof			

左递归问题

- 对CFG的一个规则来说，其右侧的第一个符号与左侧符号相同或者能够推导出左侧符号。
- 主要问题：可使搜索算法无限递归下去，不终止。

[1] $Expr \rightarrow Expr + Term$
[2] $\quad\quad\quad | \; Expr - Term$
[3] $\quad\quad\quad | \; Term$

word	cur	Rule	Stack
(	Expr	[1]	$Expr$, +, $Term$
(	Expr	[1]	$Expr$, +, $Term$, +, $Term$
(	Expr	[1]	$Expr$, +, $Term$, +, $Term$, +, $Term$
			…

消除左递归

- 引入新的非终结符，基本规则：

$$
\begin{aligned}
E &\to E\ \alpha \\
&\ \ |\ \beta
\end{aligned}
\quad\Longrightarrow\quad
\begin{aligned}
E &\to \beta\ E' \\
E' &\to \alpha\ E' \\
&\ \ |\ \epsilon
\end{aligned}
$$

举例：

$$
\begin{array}{ll}
[1] & Expr \to Expr + Term \\
[2] & \qquad\quad |\ Expr\ - Term \\
[3] & \qquad\quad |\ Term
\end{array}
\quad\Longrightarrow\quad
\begin{array}{ll}
[1] & Expr \to Term\ Expr' \\
[2] & Expr' \to +\ Term\ Expr' \\
[3] & \qquad\quad |\ -\ Term\ Expr' \\
[4] & \qquad\quad |\ \epsilon
\end{array}
$$

$$
\begin{array}{ll}
[4] & Term \to Term \times Factor \\
[5] & \qquad\quad |\ Term \div Factor \\
[6] & \qquad\quad |\ Factor
\end{array}
\quad\Longrightarrow\quad
\begin{array}{ll}
[5] & Term \to Factor\ Term' \\
[6] & Term' \to \times Factor\ Term' \\
[7] & \qquad\quad |\ \div\ Factor\ Term' \\
[8] & \qquad\quad |\ \epsilon
\end{array}
$$

$$
\begin{array}{ll}
[7] & Facor \to (Expr) \\
[8] & \qquad\quad |\ num \\
[9] & \qquad\quad |\ name
\end{array}
\quad\Longrightarrow\quad
\begin{array}{ll}
[9] & Facor \to (Expr) \\
[10] & \qquad\quad |\ num \\
[11] & \qquad\quad |\ name
\end{array}
$$

间接左递归问题

$$
\begin{aligned}
E &\rightarrow \alpha \\
\alpha &\rightarrow \beta + \\
\beta &\rightarrow E
\end{aligned}
\quad \Longrightarrow \quad E \rightarrow E +
$$

展开所有非终结符NT检测和消除间接左递归

输入: Grammar{T,NT}

开始:

for i=1 to n

  for j=1 to i-1

    if ∃ $NT_i \rightarrow NT_j\ \gamma$

      展开 $NT_i \rightarrow NT_j\ \gamma$中的非终结符$NT_j$

  重写会造成NT_i左递归的规则

通用自顶向下语法分析算法：Earley算法

- 三种基本操作：
  - 预测（Prediction）：对于每个状态$X \to \alpha \circ Y\beta$，根据语法规则预测$Y \to \circ\gamma$。
  - 扫描（Scanning）：如果下一个待处理的符号是a，并且存在状态 $X \to \alpha \circ a\beta$，则扫描该字符并且将状态变更为 $X \to \alpha a \circ \beta$ 。
  - 完成（Completion）：$Y \to \gamma \circ$完成了对Y的分析，进而更新 $X \to \alpha \circ Y\beta$为 $X \to \alpha Y \circ \beta$。

Earley算法

如何根据下列语法规则解析(3+4)×5?

$$\begin{aligned}&[1]\ Expr \rightarrow Expr + Term\\&[2]\qquad\quad \mid Expr\ - Term\\&[3]\qquad\quad \mid Term\end{aligned}$$

$$\begin{aligned}&[4]\ Term \rightarrow Term \times Factor\\&[5]\qquad\quad \mid Term \div Factor\\&[6]\qquad\quad \mid Factor\end{aligned}$$

$$\begin{aligned}&[7]\ Facor \rightarrow (Expr)\\&[8]\qquad\quad \mid num\\&[9]\qquad\quad \mid name\end{aligned}$$

(num+num)×num

no	production	origin	comment
s(0) = ◦(3+4)×5			
1	$Expr \rightarrow\circ\ Expr + Term$	0	start rule
2	$Expr \rightarrow\circ\ Expr - Term$	0	start rule
3	$Expr \rightarrow\circ\ Term$	0	start rule
4	$Term \rightarrow\circ\ Term \times Factor$	0	Predict from [0][3]
5	$Term \rightarrow\circ\ Term \div Factor$	0	Predict from [0][3]
6	$Term \rightarrow\circ\ Factor$	0	Predict from [0][3]
7	$Facor \rightarrow\circ\ (Expr)$	0	Predict from [0][6]
8	$Facor \rightarrow\circ\ num$	0	Predict from [0][6]
9	$Facor \rightarrow\circ\ name$	0	Predict from [0][6]
s(1) = (◦3+4)×5			
1	$Facor \rightarrow\ (\circ\ Expr)$	0	Scan from [0][7]
2	$Expr \rightarrow\circ\ Expr + Term$	1	Predict from [1][1]
3	$Expr \rightarrow\circ\ Expr - Term$	1	Predict from [1][1]
4	$Expr \rightarrow\circ\ Term$	1	Predict from [1][1]
5	$Term \rightarrow\circ\ Term \times Factor$	1	Predict from [1][4]
6	$Term \rightarrow\circ\ Term \div Factor$	1	Predict from [1][4]
7	$Term \rightarrow\circ\ Factor$	1	Predict from [1][4]
8	$Facor \rightarrow\circ\ (Expr)$	1	Predict from [1][7]
9	$Facor \rightarrow\circ\ num$	1	Predict from [1][7]
10	$Facor \rightarrow\circ\ name$	1	Predict from [1][7]

Earley算法

如何根据下列语法规则解析(3+4)×5?

[1] $Expr \rightarrow Expr + Term$
[2] $\quad | Expr - Term$
[3] $\quad | Term$

[4] $Term \rightarrow Term \times Factor$
[5] $\quad | Term \div Factor$
[6] $\quad | Factor$

[7] $Facor \rightarrow (Expr)$
[8] $\quad | num$
[9] $\quad | name$

no	production	origin	comment
s(2) = (3◦+4)×5			
1	$Facor \rightarrow num \circ$	1	Scan from [1][9]
2	$Term \rightarrow Factor \circ$	1	Complete [1][7]
3	$Term \rightarrow Term \circ \times Factor$	1	Complete [1][5]
4	$Term \rightarrow Term \circ \div Factor$	1	Complete [1][6]
5	$Expr \rightarrow Term \circ$	1	Complete [1][4]
6	$Expr \rightarrow Expr \circ +Term$	1	Complete [1][2]
7	$Expr \rightarrow Expr \circ -Term$	1	Complete [1][3]
8	$Facor \rightarrow (Expr \circ)$	0	Complete [0][1]
s(3) = (3+◦4)×5			
1	$Expr \rightarrow Expr + \circ Term$	1	Scan from [2][6]
2	$Term \rightarrow \circ Term \times Factor$	3	Predict from [3][1]
3	$Term \rightarrow \circ Term \div Factor$	3	Predict from [3][1]
4	$Term \rightarrow \circ Factor$	3	Predict from [3][1]
5	$Facor \rightarrow \circ (Expr)$	3	Predict from [3][4]
6	$Facor \rightarrow \circ num$	3	Predict from [3][4]
7	$Facor \rightarrow \circ name$	3	Predict from [3][4]

Earley算法

如何根据下列语法规则解析(3+4)×5?

[1] $Expr \rightarrow Expr + Term$
[2] $\quad | \; Expr \; - Term$
[3] $\quad | \; Term$

[4] $Term \rightarrow Term \times Factor$
[5] $\quad | \; Term \div Factor$
[6] $\quad | \; Factor$

[7] $Facor \rightarrow (Expr)$
[8] $\quad | \; num$
[9] $\quad | \; name$

no	production	origin	comment
s(4) = (3+4∘)×5			
1	$Facor \rightarrow num \circ$	3	Scan from [3][6]
2	$Term \rightarrow Factor \circ$	3	Complete [3][7]
3	$Term \rightarrow Term \circ \times Factor$	3	Complete [3][5]
4	$Term \rightarrow Term \circ \div Factor$	3	Complete [3][6]
5	$Expr \rightarrow Term \circ$	3	Complete [3][4]
6	$Expr \rightarrow Expr \circ +Term$	3	Complete [3][2]
7	$Expr \rightarrow Expr \circ -Term$	3	Complete [3][3]
8	$Expr \rightarrow Expr + Term \circ$	1	Complete [3][1]
9	$Facor \rightarrow \; (Expr \circ)$	0	Complete [1][1]
s(5) = (3+4)∘×5			
1	$Facor \rightarrow \; (Expr) \circ$	0	Scan from [4][9]
2	$Term \rightarrow Factor \circ$	0	Complete [0][6]
3	$Term \rightarrow Term \circ \times Factor$	0	Complete [0][4]
4	$Term \rightarrow Term \circ \div Factor$	0	Complete [0][5]
5			
6			
7			

Earley算法

如何根据下列语法规则解析(3+4)×5？

[1] $Expr \rightarrow Expr + Term$
[2] $\quad | \; Expr \; - Term$
[3] $\quad | \; Term$

[4] $Term \rightarrow Term \times Factor$
[5] $\quad | \; Term \div Factor$
[6] $\quad | \; Factor$

[7] $Facor \rightarrow (Expr)$
[8] $\quad | \; num$
[9] $\quad | \; name$

no	production	origin	comment
s(6) = (3+4)×∘5			
1	$Term \rightarrow Term \times \circ \, Factor$	0	Scan from [6][3]
2	$Facor \rightarrow \circ \, (Expr)$	6	Predict from [6][1]
3	$Facor \rightarrow \circ \, num$	6	Predict from [6][1]
4	$Facor \rightarrow \circ \, name$	6	Predict from [6][1]
s(7) = (3+4)×5∘			
1	$Facor \rightarrow num \circ$	6	Scan from [6][4]
2	$Term \rightarrow Term \times Factor \circ$	0	Complete [6][1]
3	$Expr \rightarrow Term \circ$	0	Complete [0][3]
4			
5			
6			
7			

无回溯语法

- **目的**：消除语法生成规则（右递归表达式）选择时的不确定性，避免回溯。
- **思路**：如果对于每个非终结符的任意两个生成式，其产生的首个终结符号不同，则在前瞻一个单词的情况下，总能够选择正确的生成式规则。
  - [1] $NT_1 \rightarrow NT_i \rightarrow \cdots \rightarrow \mathrm{term}_1\ NT_p$
  - [2] $NT_1 \rightarrow NT_j \rightarrow \cdots \rightarrow \mathrm{term}_2\ NT_q$
- 预测解析（Predictive Parsing）：LL(1)语法
  - Left-to-Right，Leftmost，前瞻一个字符

消除回溯：提取左因子

- 对一组产生式提取并隔离共同前缀

$$A \to \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma_1 | \dots | \gamma_j \quad \Rightarrow \quad \begin{aligned} & A \to \alpha B \mid \gamma_1 | \dots | \gamma_j \\ & B \to \beta_1 \mid \beta_2 | \dots | \beta\_n \end{aligned}$$

应用举例：

[11]	$Factor \to name$
[12]	$\quad\quad \mid name\ [ArgList]$
[13]	$\quad\quad \mid name\ (ArgList)$
[14]	$ArgList \to Expr\ MoreArgs$
[15]	$MoreArgs \to , Expr\ MoreArgs$
[16]	$\quad\quad \mid \epsilon$

$\Rightarrow$

[11]	$Factor \to name\ Arguments$
[12]	$Arguments \to [ArgList]$
[13]	$\quad\quad \mid (ArgList)$
[14]	$\quad\quad \mid \epsilon$
[15]	$ArgList \to Expr\ MoreArgs$
[16]	$MoreArgs \to , Expr\ MoreArgs$
[17]	$\quad\quad \mid \epsilon$

无回溯语法的必要性质

$$First^{+}(A \rightarrow \beta) = \begin{cases} First(\beta), & if \epsilon \notin First(\beta) \\ First(\beta) \cup Follow(A), & otherwise \end{cases}$$

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

- 同一非终结符A 的任意两个语法推导$(A \rightarrow \beta_i)$和$(A \rightarrow \beta_j)$ 所产生的的首个终结符不能相通。
- $First(\beta)$是从语法符号β推导出的每个子句的第一个终结符的集合，其值域是$T \cup \{\epsilon, \mathrm{eof}\}$。
- 如果$First(\beta)$是$\{\epsilon\}$，则计算紧随A之后出现的终结符的集合$Follow(A)$。

First集合计算

- 对于生成式$A \rightarrow \beta_1\beta_2 \ldots \beta_n$ 来说：
  - 如果 $\epsilon \notin First(\beta_1)$， 则$First(A) = First(\beta_1)$
  - 如果 $\epsilon \in First(\beta_1) \& \ldots \& \epsilon \in First(\beta_i)$, 则$First(A) = First(\beta_1) \cup \cdots \cup First(\beta_{i+1})$

[1] $Expr \rightarrow Term\ Expr'$
[2] $Expr' \rightarrow +\ Term\ Expr'$
[3] $\quad\quad | -\ Term\ Expr'$
[4] $\quad\quad | \ \epsilon$

[5] $Term \rightarrow Factor\ Term'$
[6] $Term' \rightarrow \times Factor\ Term'$
[7] $\quad\quad | \div\ Factor\ Term'$
[8] $\quad\quad | \ \epsilon$

[9] $Facor \rightarrow (Expr)$
[10] $\quad\quad | \ num$
[11] $\quad\quad | \ name$

	num	name	+	−	×	÷	(	)	ϵ
$Expr$	√	√					√		
$Expr'$			√	√					√
$Term$	√	√					√		
$Term'$					√	√			√
$Facor$	√	√					√		

Follow集合计算

- 紧随非终结符之后出现的所有可能的终结符

[1] $Expr \rightarrow Term\ Expr'$
[2] $Expr' \rightarrow +\ Term\ Expr'$
[3] $\quad\quad\quad \mid -\ Term\ Expr'$
[4] $\quad\quad\quad \mid \epsilon$

[5] $Term \rightarrow Factor\ Term'$
[6] $Term' \rightarrow \times Factor\ Term'$
[7] $\quad\quad\quad \mid \div\ Factor\ Term'$
[8] $\quad\quad\quad \mid \epsilon$

[9] $Facor \rightarrow (Expr)$
[10] $\quad\quad\quad \mid num$
[11] $\quad\quad\quad \mid name$

	num	name	$+$	$-$	$\times$	$\div$	(	)	ϵ	eof
$Expr$	√	√					√	√		√
$Expr'$			√	√				√	√	√
$Term$	√	√	√	√			√	√		√
$Term'$			√	√	√	√		√	√	√
$Facor$	√	√	√	√	√	√	√	√		√

First+集合计算

$$First^{+}(A \to \beta) = \begin{cases} First(\beta), & if \epsilon \notin First(\beta) \\ First(\beta) \cup Follow(A), & otherwise \end{cases}$$

	num	name	$+$	$-$	$\times$	$\div$	(	)	ϵ	eof
$Expr$	√	√					√	~~√~~		~~√~~
$Expr'$			√	√				√	√	√
$Term$	√	√	~~√~~	~~√~~			√	~~√~~		~~√~~
$Term'$			√	√	√	√		√	√	√
$Facor$	√	√	~~√~~	~~√~~	~~√~~	~~√~~	√	~~√~~		~~√~~

解析表构造：应用哪条规则可得到目标终结符？

	num	name	$+$	$-$	$\times$	$\div$	(	)	ϵ	*eof*
$Expr$	√	√					√	√		√
$Expr'$			√	√				√	√	√
$Term$	√	√	√	√			√	√		√
$Term'$			√	√	√	√		√	√	√
$Facor$	√	√	√	√	√	√	√	√		√

[1] $Expr \rightarrow Term\ Expr'$
[2] $Expr' \rightarrow +\ Term\ Expr'$
[3] $\quad\quad\quad |\ -\ Term\ Expr'$
[4] $\quad\quad\quad |\ \epsilon$

[5] $Term \rightarrow Factor\ Term'$
[6] $Term' \rightarrow \times Factor\ Term'$
[7] $\quad\quad\quad |\ \div\ Factor\ Term'$
[8] $\quad\quad\quad |\ \epsilon$

[9] $Facor \rightarrow (Expr)$
[10] $\quad\quad\quad |\ num$
[11] $\quad\quad\quad |\ name$

	num	name	$+$	$-$	$\times$	$\div$	(	)	ϵ	*eof*
$Expr$	1	1					1			
$Expr'$			2	3				4		4
$Term$	5	5					5			
$Term'$			8	8	6	7		8		8
$Facor$	10	11					9			

应用解析表

每次都有且仅有一个匹配规则

(3+4)×5

(num+num)×num

$$
\begin{aligned}
&[1]\ Expr \rightarrow Term\ Expr' \\
&[2]\ Expr' \rightarrow +\,Term\ Expr' \\
&[3]\qquad\quad \mid -\,Term\ Expr' \\
&[4]\qquad\quad \mid \epsilon
\end{aligned}
$$

$$
\begin{aligned}
&[5]\ Term \rightarrow Factor\ Term' \\
&[6]\ Term' \rightarrow \times Factor\ Term' \\
&[7]\qquad\quad \mid \div\ Factor\ Term' \\
&[8]\qquad\quad \mid \epsilon
\end{aligned}
$$

$$
\begin{aligned}
&[9]\ Facor \rightarrow (Expr) \\
&[10]\qquad\quad \mid num \\
&[11]\qquad\quad \mid name
\end{aligned}
$$

word	cur	Rule	Stack
(	Expr	[1]	Term, Expr’
(	Term	[5]	Factor, Term’, Expr’
(	Factor	[9]	(, Expr,), Term’, Expr’
(	(	-	Expr,), Term’, Expr’
num	Expr	[1]	Term, Expr’,), Term’, Expr’
num	Term	[5]	Factor, Term’, Expr’,), Term’, Expr’
num	Factor	[10]	num, Term’, Expr’,), Term’, Expr’
num	num	-	Term’, Expr’,), Term’, Expr’
+	Term’	[8]	Expr’,), Term’, Expr’
+	Expr’	[2]	+, Term, Expr’,), Term’, Expr’
+	+	-	Term, Expr’,), Term’, Expr’
num	Term	[5]	Factor, Term’, Expr’,), Term’, Expr’
num	Factor	[10]	num, Term’, Expr’,), Term’, Expr’
num	num	-	Term’, Expr’,), Term’, Expr’
)	Term’	[8]	Expr’,), Term’, Expr’
)	Expr’	[4]	), Term’, Expr’
)	)	-	Term’, Expr’
+	Term’	[8]	Expr’
…	…	…	…

三、自底向上分析

LR(0)

LR(1): SLR/LALR

基本思路：基于规约的方法

- $A \to \beta$, 如果在语法分析树的上边缘找到β，则将其规约为A

$$
\begin{array}{ll}
[1] & S \to \epsilon \\
[2] & \quad\; \mid [S]S
\end{array}
$$

	当前栈	待输入
		[ ]
第1步:	[	]
第2步:	[S	]
第3步:	[S]	
第4步:	[S]S	
第5步:	S	

移进和规约（Shift-Reduce）

Shift
移进表示：

$$\frac{w : \beta}{wa : \beta a}\,\text{shift}$$

Reduce
规约表示：

$$\frac{[r]\ X \rightarrow \alpha \qquad w : \beta\alpha}{w : \beta X}\,\text{reduce(r)}$$

示例：

$$\frac{\epsilon : \beta_0}{[: \beta_0 [}\,\text{shift}$$

$$\frac{[: \beta_1}{[] : \beta_1]}\,\text{shift}$$

$$\cdots$$

$$\frac{[][[][] : \beta_7}{[][[][]] : \beta_7]}\,\text{shift}$$

$$[1]\ S \rightarrow \epsilon$$

$$\frac{w : \beta}{w : \beta S}\,\text{reduce([1])}$$

$$[2]\ S \rightarrow [S]S$$

$$\frac{w : \beta [S]S}{w : \beta S}\,\text{reduce([2])}$$

移进-规约应用

最右推导

[1]	$S \to \epsilon$
[2]	$\mid [S]S$

	[][[][]]	$\epsilon : \epsilon$	shift [
[	[][[][]]	$[: [$	reduce([1])
[S	][[][]]	$[: [S$	shift
[S]	[[][]]	$[] : [S]$	shift
[S][	[][]]	$[][: [S][$	shift
[S][[	][]]	$[][[: [S][[$	reduce([1])
[S][[S	][]]	$[][[: [S][[S$	shift
[S][[S]	[]]	$[][[] : [S][[S]$	shift
[S][[S][	]]	$[][[][: [S][[S][$	reduce([1])
[S][[S][S	]]	$[][[][: [S][[S][S$	shift
[S][[S][S]	]	$[][[][] : [S][[S][S]$	reduce([1])
[S][[S][S]S	]	$[][[][] : [S][[S][S]S$	reduce([2])
[S][[S]S	]	$[][[][] : [S][[S]S$	reduce([2])
[S][S	]	$[][[][] : [S][S$	shift
[S][S]		$[][[][]] : [S][[S]S]$	reduce([1])
[S][S]S		$[][[][]] : [S][S]S$	reduce([2])
[S]S		$[][[][]] : [S]S$	reduce([2])
S		$[][[][]] : S$	

LR(0)句柄分析

$$
\begin{aligned}
&[1]\ E \to E + T \\
&[2]\ \quad\ \mid T \\
&[3]\ T \to T \times F \\
&[4]\ \quad\ \mid F \\
&[5]\ F \to (E) \\
&[6]\ \quad\ \mid id
\end{aligned}
$$

增强语法 ⇒

$$
\begin{aligned}
&Goal \to E \\
&E \to E + T \\
&\quad\ \mid T \\
&T \to T \times F \\
&\quad\ \mid F \\
&F \to (E) \\
&\quad\ \mid id
\end{aligned}
$$

增强语法有唯一的目标符号Goal，不会出现在产生式的右侧。

$Goal \to E$ $\xRightarrow{\text{句柄状态}}$ $Goal \to \circ E$, $Goal \to E \circ$

可应用$E \to E + T$规约的句柄状态 ⇒ $E \to \circ E + T$, $E \to E \circ + T$, $E \to E + \circ T$, $E \to E + T \circ$

可应用$E \to T$规约的句柄状态 ⇒ $E \to \circ T$, $E \to T \circ$

可应用$T \to T \times F$规约的句柄状态 ⇒ $T \to \circ T \times F$, $T \to T \circ \times F$, $T \to T \times \circ F$, $T \to T \times F \circ$

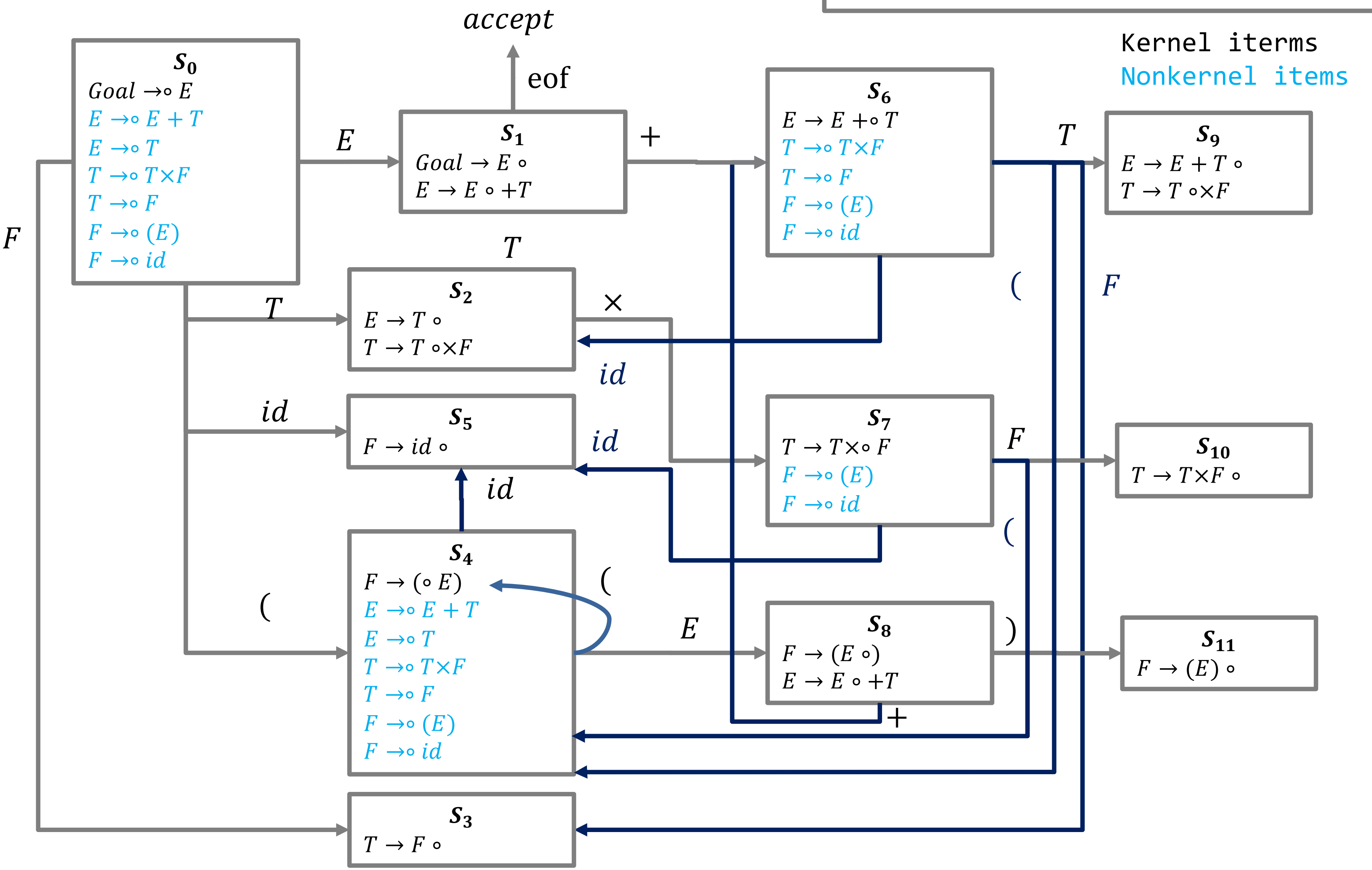
可应用$T \to F$规约的句柄状态 ⇒ $T \to \circ F$, $T \to F \circ$

可应用$F \to (E)$规约的句柄状态 ⇒ $F \to \circ (E)$, $F \to (\circ E)$, $F \to (E \circ)$, $F \to (E) \circ$

可应用$F \to id$规约的句柄状态 ⇒ $F \to \circ id$, $F \to id \circ$

LR(0)自动机构建

While (S has changed)
  for each item $[A \to \beta \circ C\delta, a] \in S$
    for each production $[C \to \lambda] \in G$
      if $[C \to\circ \lambda] \notin S$
        $S \leftarrow S \cup [C \to\circ \lambda]$



LR(0)自动机的状态转移关系表

迭代	规范项	id	+	×	(	)	eof	E	T	F
0	S_0	S_5	$\emptyset$	$\emptyset$	S_4	$\emptyset$	$\emptyset$	S_1	S_2	S_3
1	S_1	$\emptyset$	S_6	$\emptyset$	$\emptyset$	$\emptyset$	$accept$	$\emptyset$	$\emptyset$	$\emptyset$
	S_2	$\emptyset$	$\emptyset$	S_7	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	S_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	S_4	S_5	$\emptyset$	$\emptyset$	S_4	$\emptyset$	$\emptyset$	S_8	S_2	S_3
	S_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2	S_6	S_5	$\emptyset$	$\emptyset$	S_4	$\emptyset$	$\emptyset$	$\emptyset$	S_9	S_3
	S_7	S_5	$\emptyset$	$\emptyset$	S_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	S_{10}
	S_8	$\emptyset$	S_6	$\emptyset$	$\emptyset$	S_{11}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
3	S_9	$\emptyset$	$\emptyset$	S_7	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	S_{10}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	S_{11}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

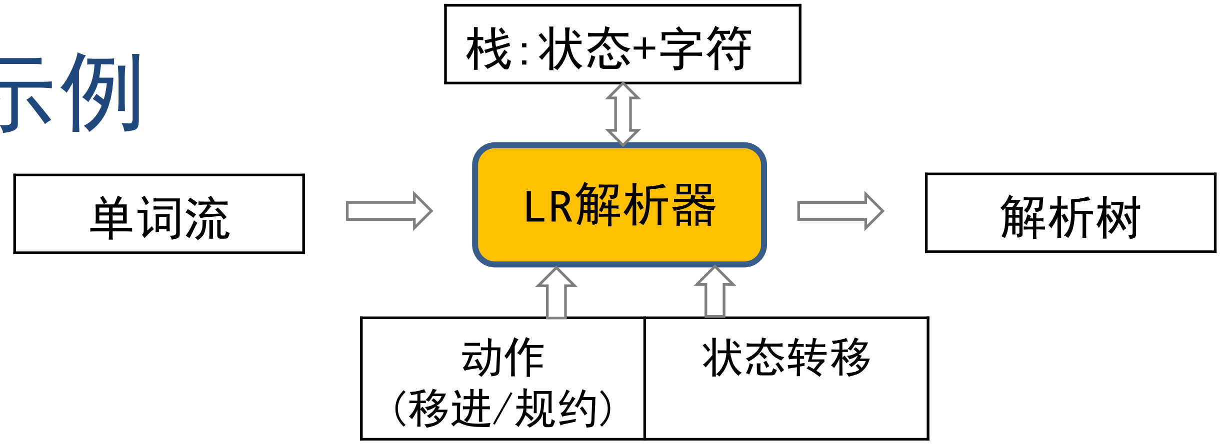
构建SLR解析表

移进条件：如果$A \to \alpha \circ a\beta \in S_i$，并且$Goto(S_i, a) = S_j$，设置$Action(S_i, a) =$ “$shift\ j$”

规约条件：如果$A \to \alpha \circ \in S_i$，$\forall a \in Follow(A)$，设置$Action(S_i, a) =$ “$reduce\ A \to \alpha$”

规范项	Action						Goto		
	id	+	×	(	)	eof	E	T	F
S_0	shift 5			shift 4			1	2	3
S_1		shift 6				acept			
S_2		reduce [2]	shift 7		reduce [2]	reduce [2]			
S_3									
S_4	shift 5			shift 4			8	2	3
S_5		reduce [6]	reduce [6]		reduce [6]	reduce [6]			
S_6	shift 5			shift 4				9	3
S_7	shift 5			shift 4					10
S_8		shift 6			shift 11				
S_9		reduce [1]	shift 7		reduce [1]	reduce [1]			
S_{10}		reduce [3]	reduce [3]		reduce [3]	reduce [3]			
S_{11}		reduce [5]	reduce [5]		reduce [5]	reduce [5]			

SLR应用示例



Stack	Symbols	Input	Action
0		id×id $	shift id, goto S_5
0 5	id	×id $	reduce by $F \to id$, back to S_0, goto S_3
0 3	F	×id $	reduce by $T \to F$, back to S_0, goto S_2
0 2	T	×id $	shift ×, goto S_7
0 2 7	T ×	id $	shift id, goto S_5
0 2 7 5	T × id	$	reduce by $F \to id$, back to S_7, goto S_{10}
0 2 7 10	T × F	$	reduce by $T \to T \times F$, back to $S_7S_2S_0$, goto S_2
0 2	T	$	reduce by $E \to T$, back to S_0, goto S_1
0 1	E	$	

练习

- 下面的语法是否是LL(1)？是否是SLR(1)

[1] $S \rightarrow AaAb$
[2] $\quad\quad |BbBa$
[3] $A \rightarrow \epsilon$
[4] $B \rightarrow \epsilon$

练习

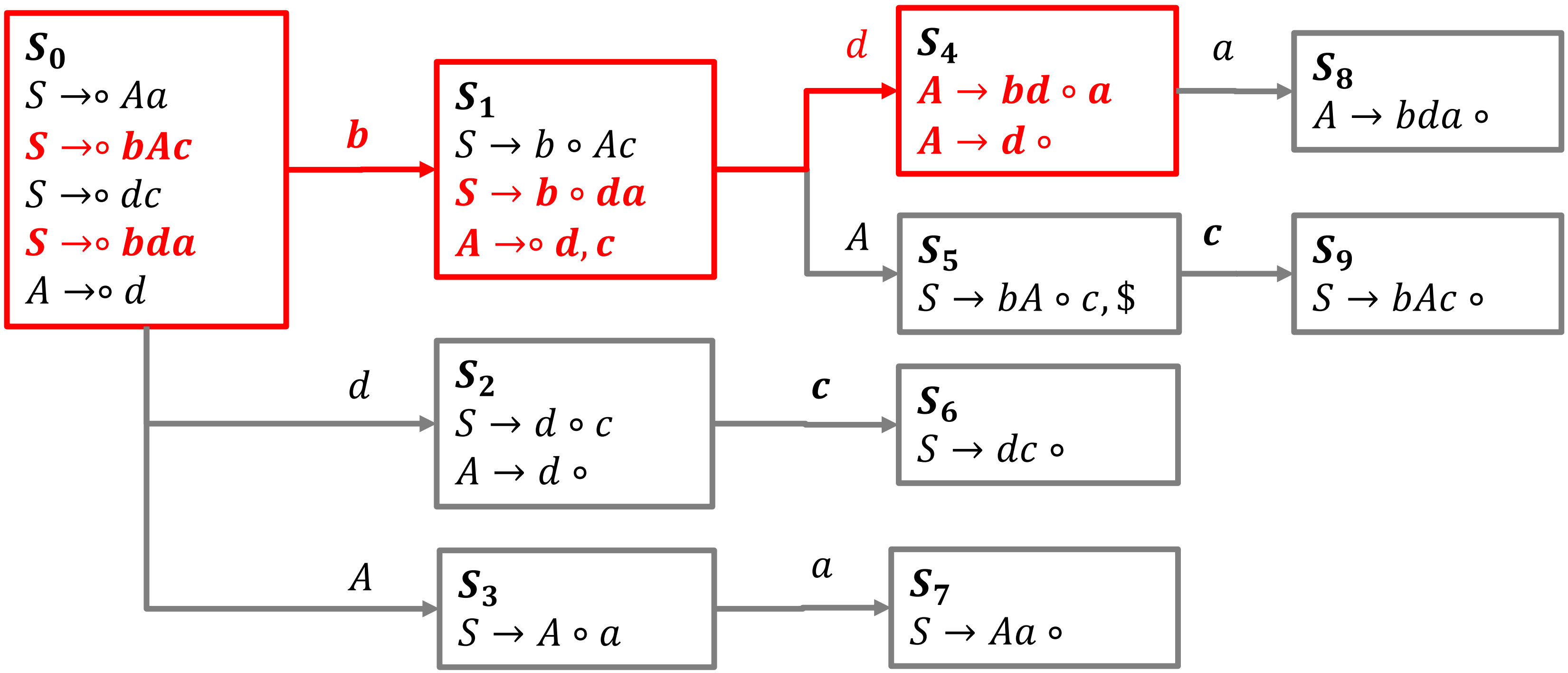
- 下面的语法是否是LL(1)？是否是SLR(1)

[1] $S \rightarrow SA$
[2] $\quad\quad |A$
[3] $A \rightarrow a$

二义性语法：移进-规约冲突

- 构造SLR解析表则解析bda时存在移进-规约冲突
- S_4下一个字符为a，可移进
- $a \in Follow(A)$，可规约

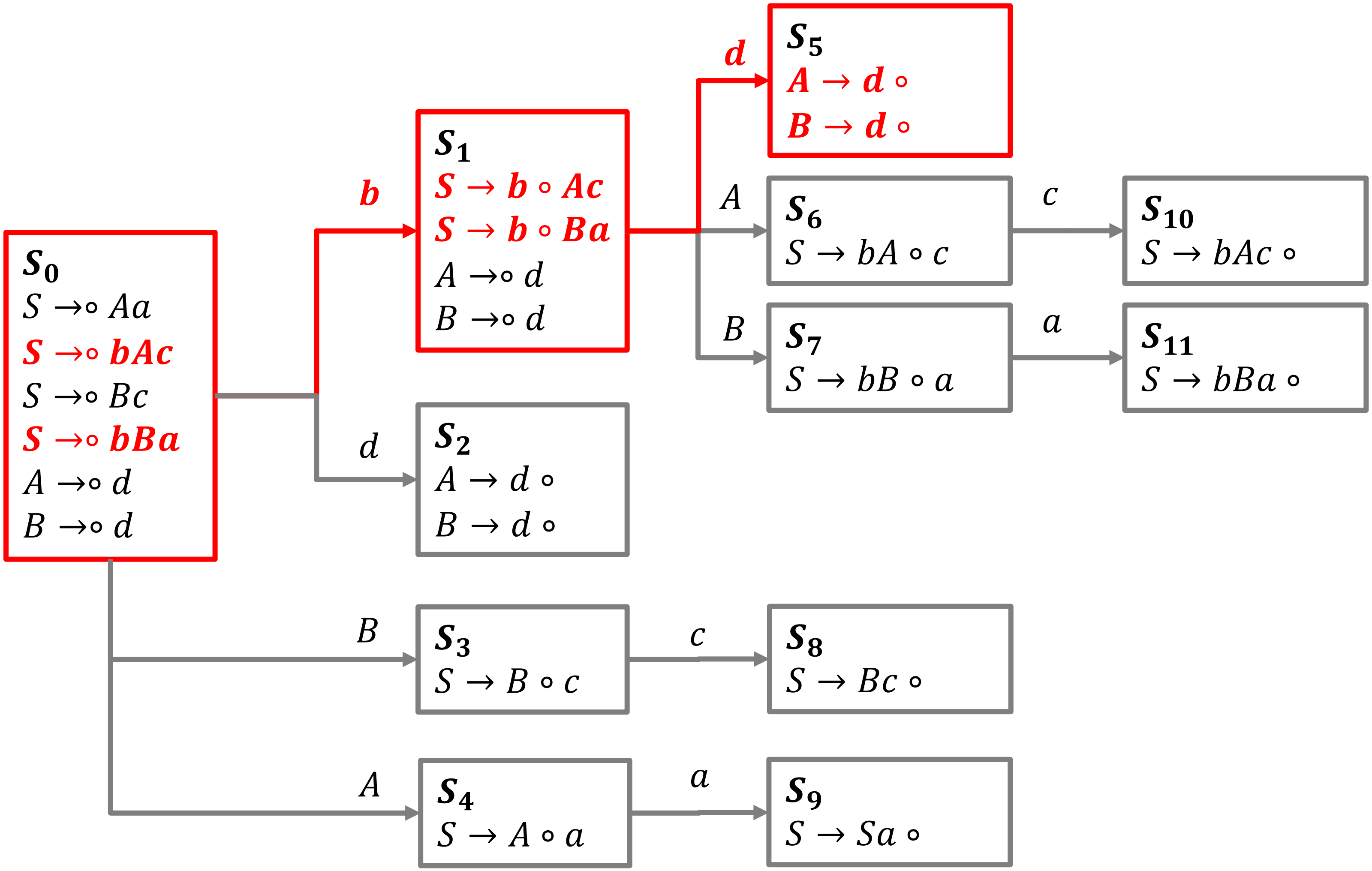
[1] $S \to Aa$
[2] $\quad |bAc$
[3] $\quad |dc$
[4] $\quad |bda$
[5] $A \to d$



二义性语法：规约-规约冲突

- 解析bdc时存在规约($A \rightarrow d$)-规约($B \rightarrow d$)冲突
- 根据Follow选择规约则可以解决冲突问题。

[1] $S \rightarrow Aa$
[2] $\quad |bAc$
[3] $\quad |Bc$
[4] $\quad |bBa$
[5] $A \rightarrow d$
[6] $B \rightarrow d$



如何选取移进-规约操作？

- 根据当前的栈顶句柄信息：SLR（Simple LR）
  - 通过构造LR(0)自动机和下个字符判断是否可以移进
  - 需要规约时根据Follow判断是否可行
- 自动机构造时考虑Follow信息：经典LR(1)：
  - 主要问题：LR(1)自动机状态（规范集）多
- LALR（Lookahead LR）
  - 自动机构造时考虑Follow信息
  - 同时精简规范集

LR(1)自动机构造

[1] $Goal \rightarrow List$
[2] $List \rightarrow List\ Pair$
[3] $\quad\quad | \ Pair$
[4] $Pair \rightarrow (Pair)$
[5] $\quad\quad | \ (\,)$

1) 构造其LR(1)项的全集
2) 迭代过程
   - 通过闭包找到规范族
   - 分析规范族之间的状态转移关系

LR(1)项的全集

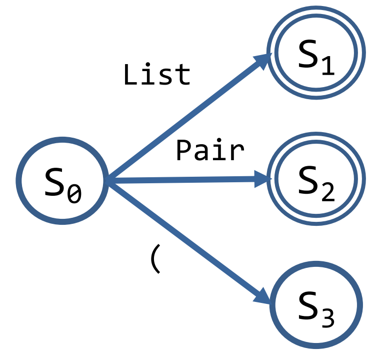
$[Goal \rightarrow \circ\ List, \text{eof}]$		
$[Goal \rightarrow List\ \circ, \text{eof}]$		
$[List \rightarrow \circ\ List\ Pair, \text{eof}]$	$[List \rightarrow \circ\ List\ Pair, (]$	
$[List \rightarrow List \circ Pair, \text{eof}]$	$[List \rightarrow List \circ Pair, (]$	
$[List \rightarrow List\ Pair\ \circ, \text{eof}]$	$[List \rightarrow List\ Pair\ \circ, (]$	
$[List \rightarrow \circ\ Pair, \text{eof}]$	$[List \rightarrow \circ\ Pair, (]$	
$[List \rightarrow Pair\ \circ, \text{eof}]$	$[List \rightarrow Pair\ \circ, (]$	
$[Pair \rightarrow \circ\ (Pair), \text{eof}]$	$[Pair \rightarrow \circ\ (Pair), (]$	$[Pair \rightarrow \circ\ (Pair),)]$
$[Pair \rightarrow (\circ\ Pair), \text{eof}]$	$[Pair \rightarrow (\circ\ Pair), (]$	$[Pair \rightarrow (\circ\ Pair),)]$
$[Pair \rightarrow (Pair\ \circ), \text{eof}]$	$[Pair \rightarrow (Pair\ \circ), (]$	$[Pair \rightarrow (Pair\ \circ),)]$
$[Pair \rightarrow (Pair)\ \circ, \text{eof}]$	$[Pair \rightarrow (Pair)\ \circ, (]$	$[Pair \rightarrow (Pair)\ \circ,)]$
$[Pair \rightarrow \circ\ (\,), \text{eof}]$	$[Pair \rightarrow \circ\ (\,), (]$	$[Pair \rightarrow \circ\ (\,),)]$
$[Pair \rightarrow (\circ), \text{eof}]$	$[Pair \rightarrow (\circ), (]$	$[Pair \rightarrow (\circ),)]$
$[Pair \rightarrow (\,)\ \circ, \text{eof}]$	$[Pair \rightarrow (\,)\ \circ, (]$	$[Pair \rightarrow (\,)\ \circ,)]$

计算LR(1)闭包

```
While (s has changed)
  for each item [A → β ∘ Cδ, a] ∈ s
    for each production [C → λ] ∈ P
      for each b ∈ FIRST(δa)
        s ← s ∪ [C → λ, b]
```

$[Goal \rightarrow \circ\ List, \text{eof}]$		
$[Goal \rightarrow List\ \circ, \text{eof}]$		
$[List \rightarrow \circ\ List\ Pair, \text{eof}]$	$[List \rightarrow \circ\ List\ Pair, (]$	
$[List \rightarrow List \circ Pair, \text{eof}]$	$[List \rightarrow List \circ Pair, (]$	
$[List \rightarrow List\ Pair\ \circ, \text{eof}]$	$[List \rightarrow List\ Pair\ \circ, (]$	
$[List \rightarrow \circ\ Pair, \text{eof}]$	$[List \rightarrow \circ\ Pair, (]$	
$[List \rightarrow Pair\ \circ, \text{eof}]$	$[List \rightarrow Pair\ \circ, (]$	
$[Pair \rightarrow \circ\ (Pair), \text{eof}]$	$[Pair \rightarrow \circ\ (Pair), (]$	$[Pair \rightarrow \circ\ (Pair),)]$
$[Pair \rightarrow (\circ\ Pair), \text{eof}]$	$[Pair \rightarrow (\circ\ Pair), (]$	$[Pair \rightarrow (\circ\ Pair),)]$
$[Pair \rightarrow (Pair\ \circ), \text{eof}]$	$[Pair \rightarrow (Pair\ \circ), (]$	$[Pair \rightarrow (Pair\ \circ),)]$
$[Pair \rightarrow (Pair)\ \circ, \text{eof}]$	$[Pair \rightarrow (Pair)\ \circ, (]$	$[Pair \rightarrow (Pair)\ \circ,)]$
$[Pair \rightarrow \circ\ (\ ), \text{eof}]$	$[Pair \rightarrow \circ\ (\ ), (]$	$[Pair \rightarrow \circ\ (\ ),)]$
$[Pair \rightarrow (\ \circ\ ), \text{eof}]$	$[Pair \rightarrow (\ \circ\ ), (]$	$[Pair \rightarrow (\ \circ\ ),)]$
$[Pair \rightarrow (\ )\ \circ, \text{eof}]$	$[Pair \rightarrow (\ )\ \circ, (]$	$[Pair \rightarrow (\ )\ \circ,)]$

分析状态转移关系



$$S_0 = \boxed{\begin{array}{ll} [Goal \rightarrow \circ\ List, \text{eof}] & \\ [List \rightarrow \circ\ List\ Pair, \text{eof}] & [List \rightarrow \circ\ List\ Pair, (] \\ [List \rightarrow \circ\ Pair, \text{eof}] & [List \rightarrow \circ\ Pair, (] \\ [Pair \rightarrow \circ\ (Pair), \text{eof}] & [Pair \rightarrow \circ\ (Pair), (] \\ [Pair \rightarrow \circ\ (\,), \text{eof}] & [Pair \rightarrow \circ\ (\,), (] \end{array}}$$

$$Goto(S_0, () = \boxed{\begin{array}{lll} [Pair \rightarrow\ (\circ\ Pair), \text{eof}] & [Pair \rightarrow\ (\circ\ Pair), (] & [Pair \rightarrow \circ\ (Pair),)] \\ [Pair \rightarrow\ (\ \circ\ ), \text{eof}] & [Pair \rightarrow\ (\ \circ\ ), (] & [Pair \rightarrow \circ\ (\,),)] \end{array}}$$

$$= \boldsymbol{S_3}$$

$$Goto(S_0,)) = \emptyset$$

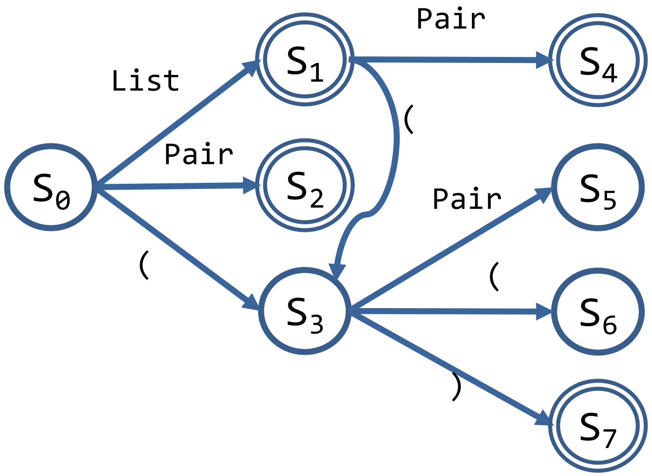
$$Goto(S_0, \text{eof}) = \emptyset$$

$$Goto(S_0, List) = \boxed{\begin{array}{ll} [Goal \rightarrow List\ \circ, \text{eof}] & \\ [List \rightarrow List\ \circ\ Pair, \text{eof}] & [List \rightarrow List\ \circ\ Pair, (] \\ [Pair \rightarrow \circ\ (Pair), \text{eof}] & [Pair \rightarrow \circ\ (Pair), (] \\ [Pair \rightarrow \circ\ (\,), \text{eof}] & [Pair \rightarrow \circ\ (\,), (] \end{array}} = \boldsymbol{S_1}$$

$$Goto(S_0, Pair) = \boxed{\begin{array}{ll} [List \rightarrow Pair\ \circ, \text{eof}] & [List \rightarrow Pair\ \circ, (] \end{array}} = \boldsymbol{S_2}$$

$$Goto(S_0, Goal) = \emptyset$$

分析状态转移关系



$Goto(S_1, () = S_3$

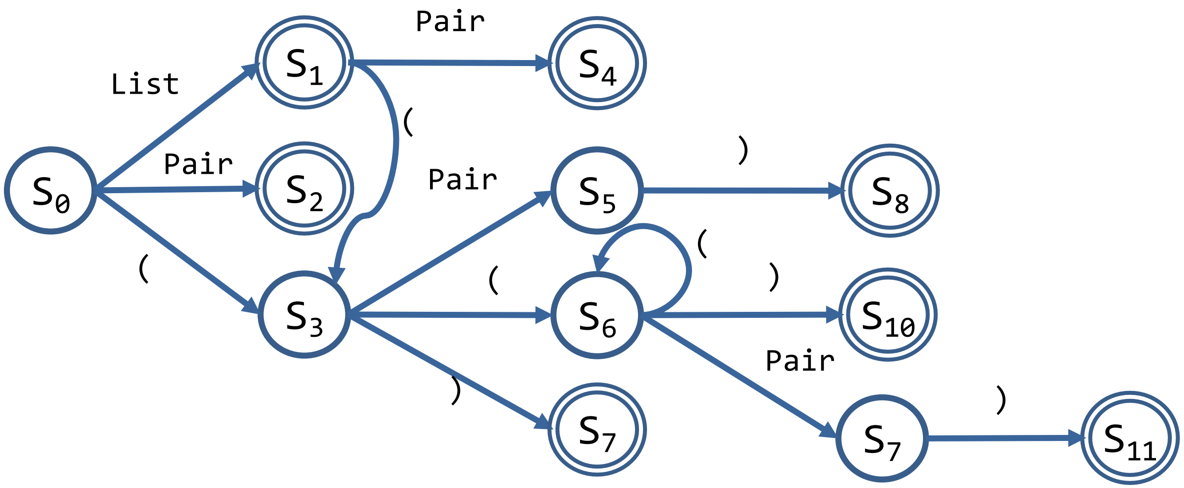
$Goto(S_1, Pair) =$ $\boxed{[List \rightarrow List \circ Pair, \text{eof}] \quad [List \rightarrow List \circ Pair, (]}$ $= \boldsymbol{S_4}$

$Goto(S_3, Pair) =$ $\boxed{[Pair \rightarrow (Pair \circ), \text{eof}] \quad [Pair \rightarrow (Pair \circ), (]}$ $= \boldsymbol{S_5}$

$Goto(S_3, () =$ $\boxed{\begin{array}{ll} [Pair \rightarrow (\circ\, Pair), (] & [Pair \rightarrow \circ\, (Pair),)] \\ [Pair \rightarrow (\circ\,), (] & [Pair \rightarrow \circ\, (\,),)] \end{array}}$ $= \boldsymbol{S_6}$

$Goto(S_3,)) =$ $\boxed{[Pair \rightarrow (\,)\circ, \text{eof}] \quad [Pair \rightarrow (\,)\circ, (]}$ $= \boldsymbol{S_7}$

分析状态转移关系



$Goto(S_6, () = S_6$

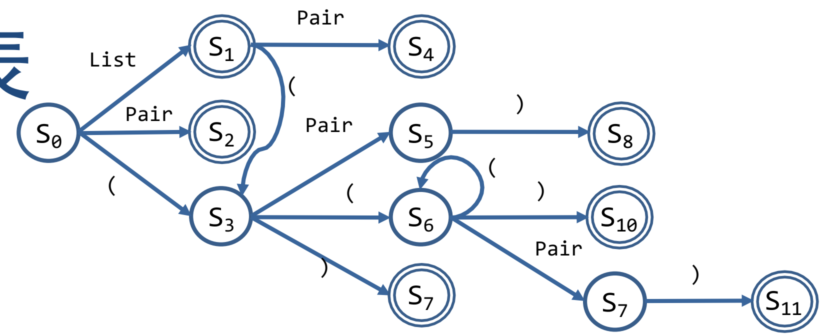
$Goto(S_5,)) =$ $\boxed{[Pair \rightarrow \ (Pair) \circ, \mathrm{eof}] \qquad [Pair \rightarrow \ (Pair) \circ, (]}$ $= \boldsymbol{S_8}$

$Goto(S_6, Pair) =$ $\boxed{[Pair \rightarrow \ (Pair \circ), (]}$ $= \boldsymbol{S_9}$

$Goto(S_6,)) =$ $\boxed{[Pair \rightarrow \ (\,) \circ, (]}$ $= \boldsymbol{S_{10}}$

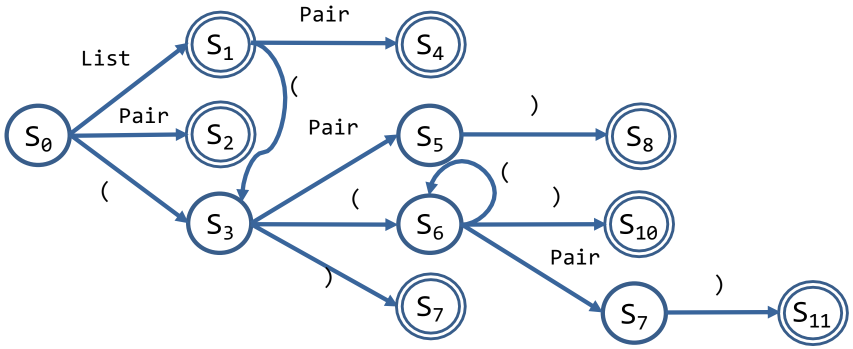
$Goto(S_9,)) =$ $\boxed{[Pair \rightarrow \ (Pair) \circ, (]}$ $= \boldsymbol{S_{11}}$

构造状态转移关系表



迭代	规范项	Goal	List	Pair	(	)	eof
0	S_0	∅	S_1	S_2	S_3	∅	∅
1	S_1	∅	∅	S_4	S_3	∅	∅
	S_2	∅	∅	∅	∅	∅	∅
	S_3	∅	∅	S_5	S_6	S_7	∅
2	S_4	∅	∅	∅	∅	∅	∅
	S_5	∅	∅	∅	∅	S_8	∅
	S_6	∅	∅	S_9	S_6	S_{10}	∅
	S_7	∅	∅	∅	∅	∅	∅
3	S_8	∅	∅	∅	∅	∅	∅
	S_9	∅	∅	∅	∅	S_{11}	∅
	S_{10}	∅	∅	∅	∅	∅	∅
4	S_{11}	∅	∅	∅	∅	∅	∅

得到LR(1)解析表



规范项	Action			Goto	
	(	**)**	**eof**	**List**	**Pair**
S_0	shift 3		*accept*	1	2
S_1	shift 3				4
S_2	reduce [3]		reduce [3]		
S_3	shift 6	shift 7			5
S_4	reduce [2]		reduce [2]		
S_5		shift 8			
S_6	shift 6	shift 10			9
S_7	reduce [5]		reduce [5]		
S_8	reduce [4]		reduce [4]		
S_9		shift 11			
S_{10}		reduce [5]			
S_{11}		reduce [4]			

练习

构造下列语法的LR(1)解析表，并解析bdc

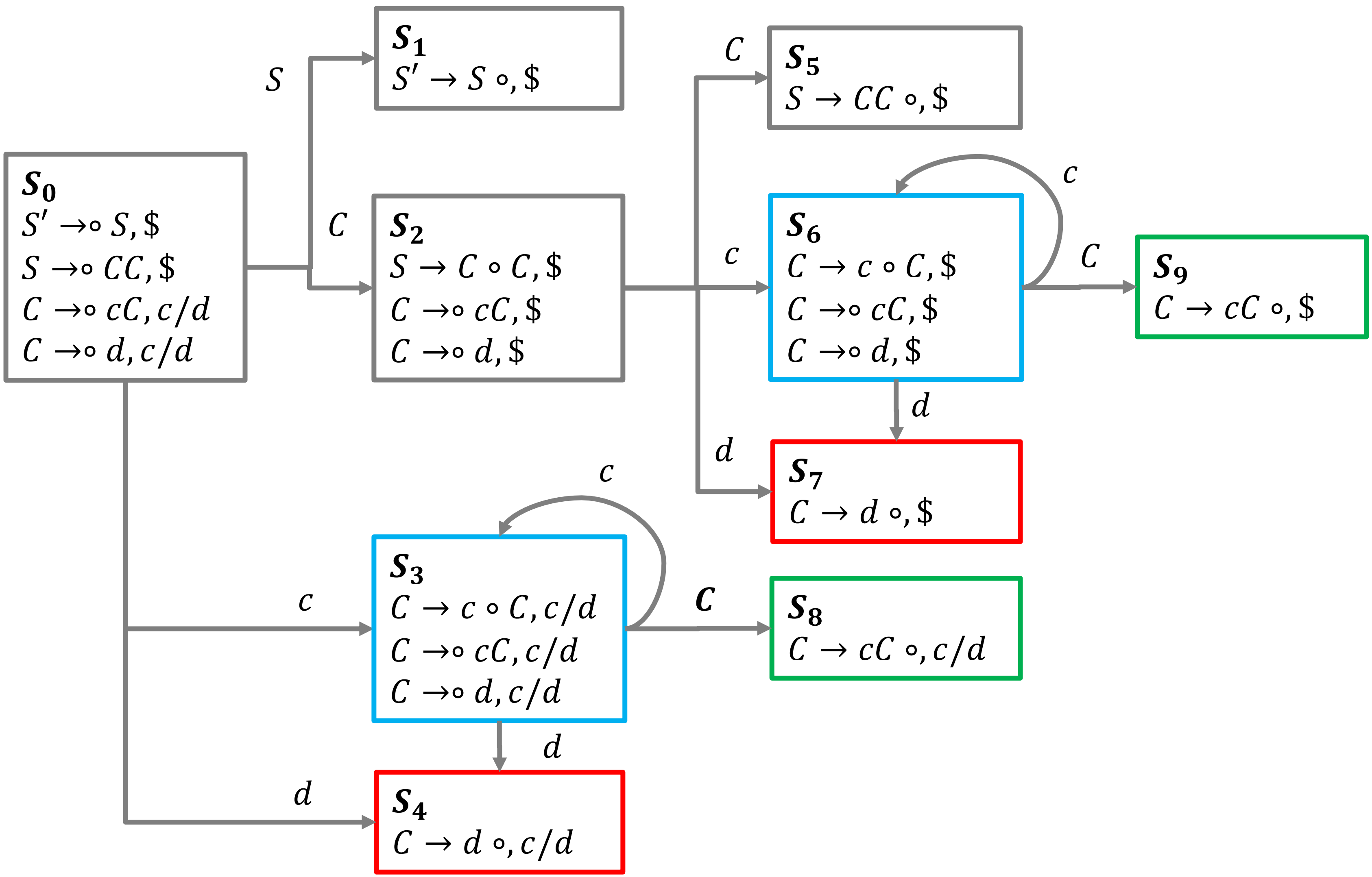
[1] $S \rightarrow Aa$
[2] $\quad\quad |bAc$
[3] $\quad\quad |Bc$
[4] $\quad\quad |bBa$
[5] $A \rightarrow d$
[6] $B \rightarrow d$

LALR

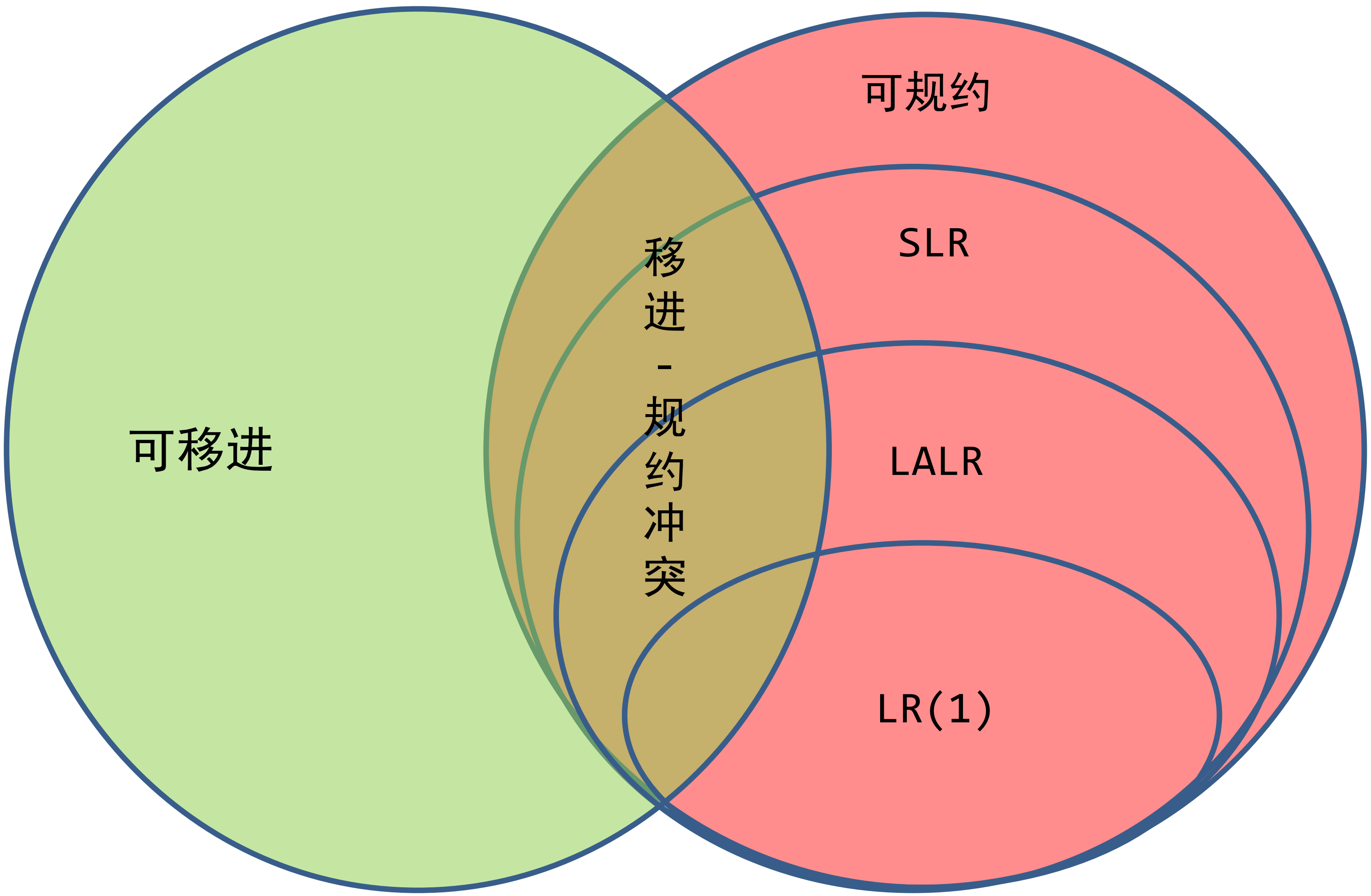
- SLR(1)存在移进-规约、规约-规约冲突问题，支持的语法范围小；
- LR(1)在规范集构造时融合了Follow信息，可以避免很多冲突问题，但解析表可能会比较大；
- LALR是一种折中方法，解析表大小和SLR相同；
- LALR构造思路：合并句柄状态完全相同的状态集

LALR语法举例

[1] $S' \to S$
[2] $S \to CC$
[3] $C \to cC$
[4] $\quad\quad | \; d$



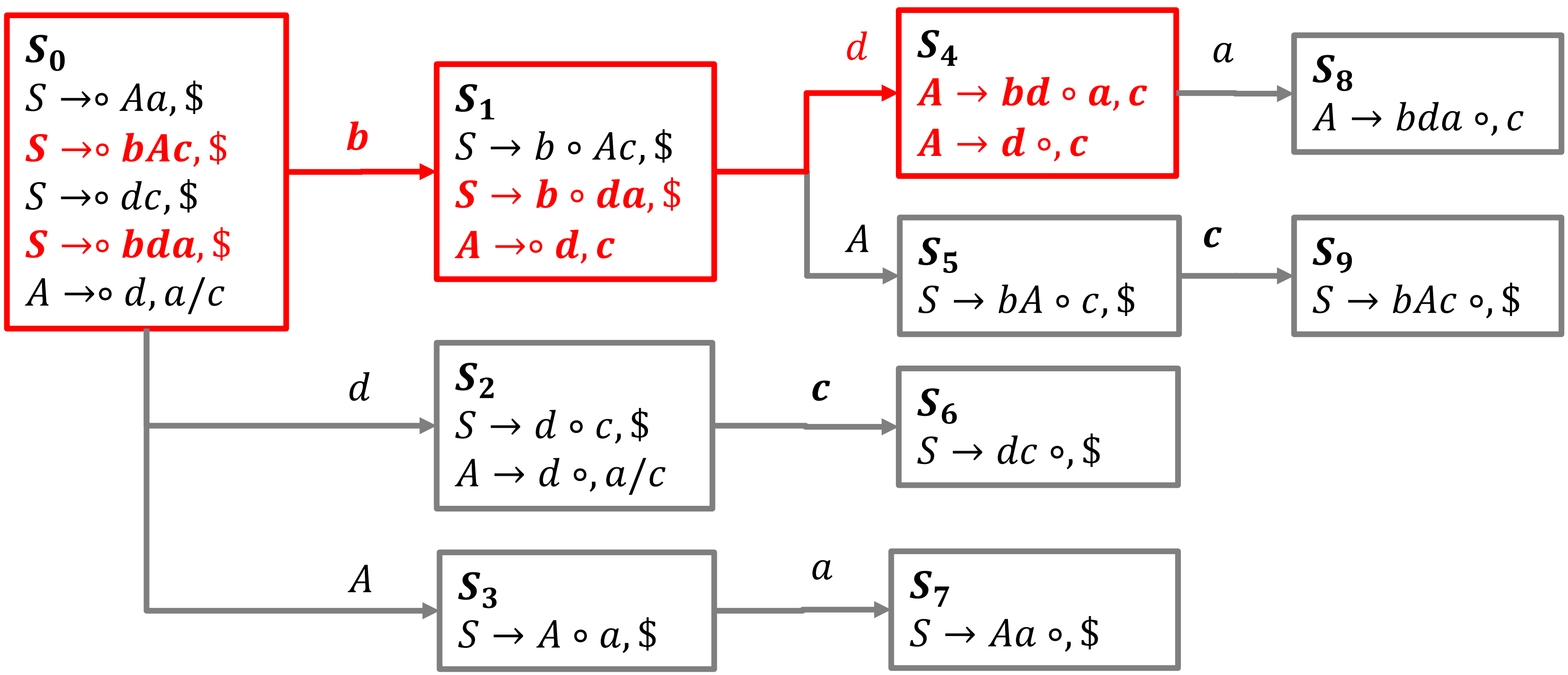
SLR(1)，LALR(1)，LR(1)语法的关系



举例：LALR，非SLR(1)语法

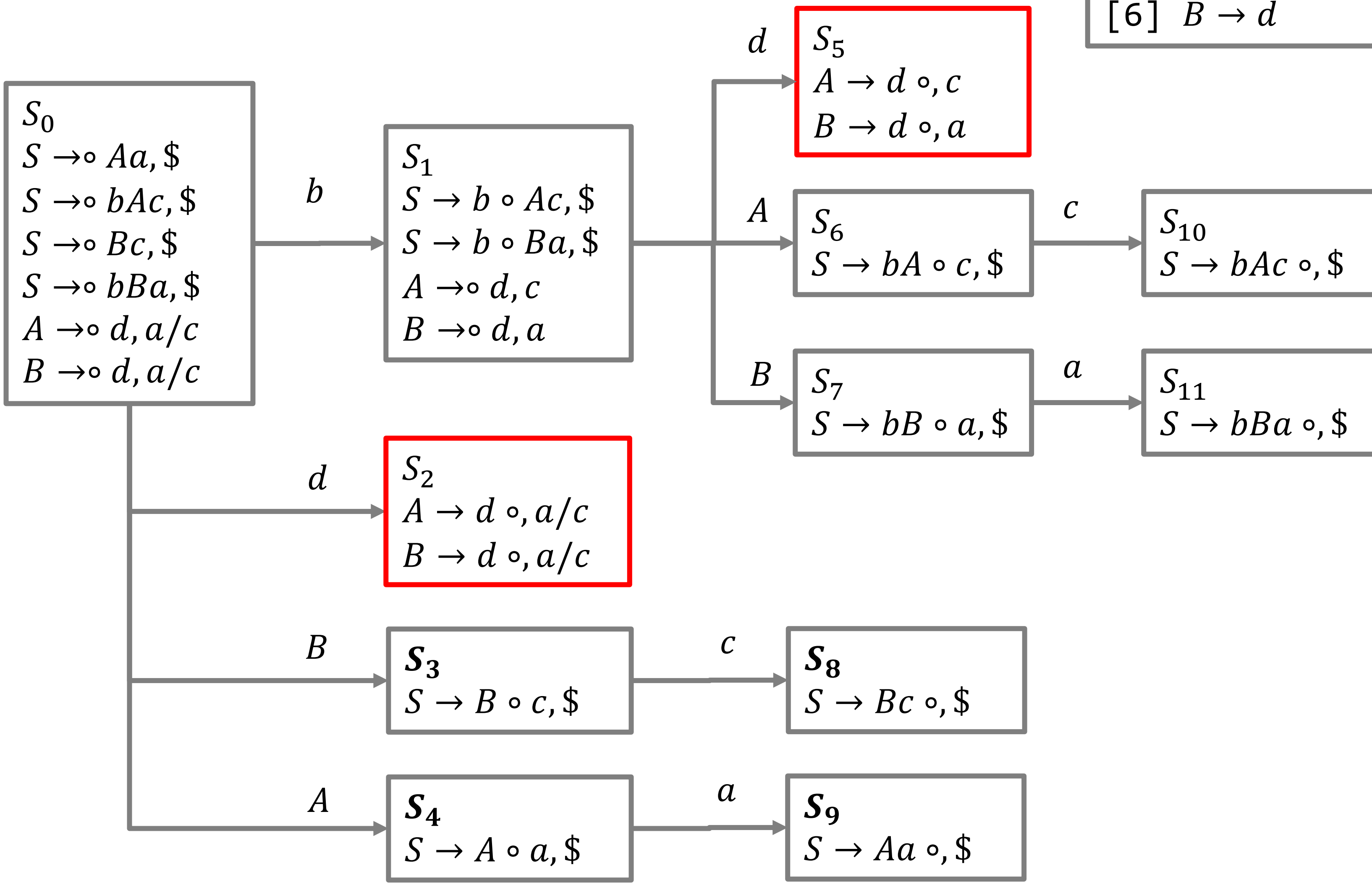
- 构造SLR解析表则解析bda时存在移进-规约冲突
  - S_4下一个字符为a，可移进
  - $a \in Follow(A)$，可规约
- LALR解析方法可以避免冲突

[1] $S \to Aa$
[2] $\quad |bAc$
[3] $\quad |dc$
[4] $\quad |bda$
[5] $A \to d$



举例：LR(1)，非LALR语法

[1] $S \rightarrow Aa$
[2] $\quad |bAc$
[3] $\quad |Bc$
[4] $\quad |bBa$
[5] $A \rightarrow d$
[6] $B \rightarrow d$



二义性语法：非LR(1)语法

$$\begin{array}{ll}[1] & Expr \rightarrow Expr + Expr \\ [2] & \quad\quad\quad\; \mid num\end{array}$$

Stack		Input	Action
	||	num + num + num	shift
num	||	+ num + num	reduce(2)
Expr	||	+ num + num	shift
Expr +	||	num + num	shift
Expr + num	||	+ num	reduce(2)
Expr + Expr	||	+ num	shift/reduce(1)

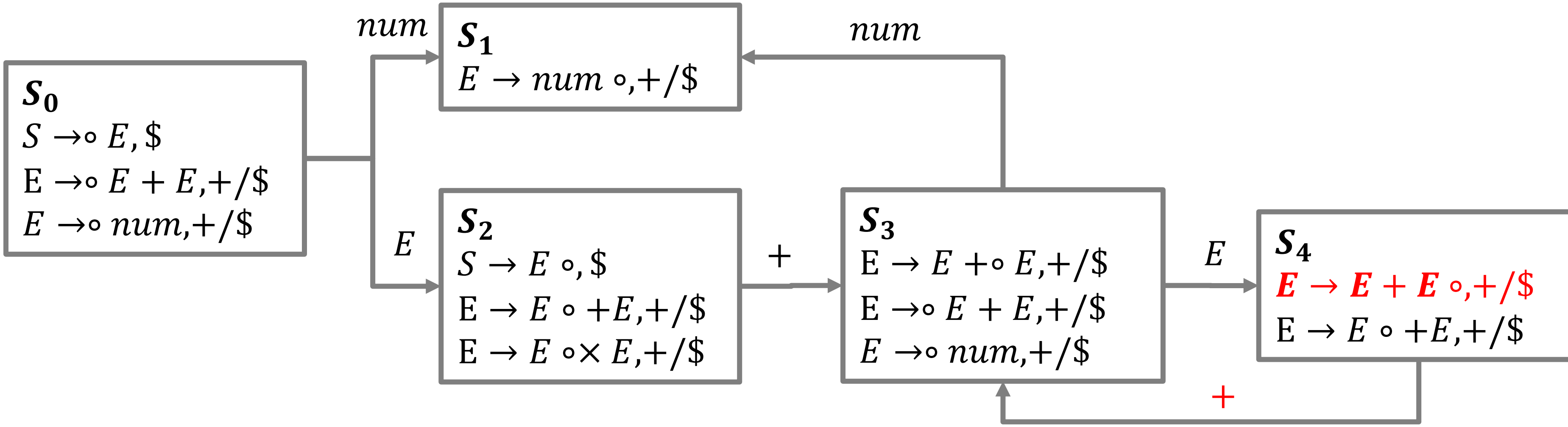
选择一：规约

Expr	||	+ num
Expr +	||	num
Expr + num	||	
Expr + Expr	||	
Expr	||	

选择二：移进

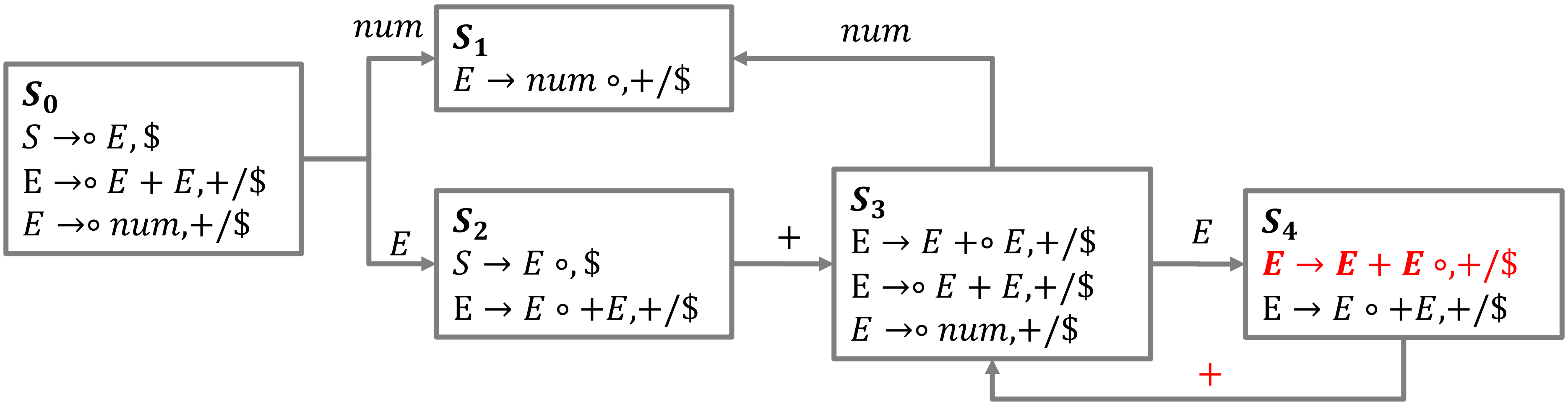
Expr + Expr +	||	num
Expr + Expr + num	||	
Expr + Expr + Expr	||	
Expr + Expr	||	
Expr	||	

举例：非LR(1)语法举例



解析num + num + num时， S_4在处理+时存在移进-规约冲突

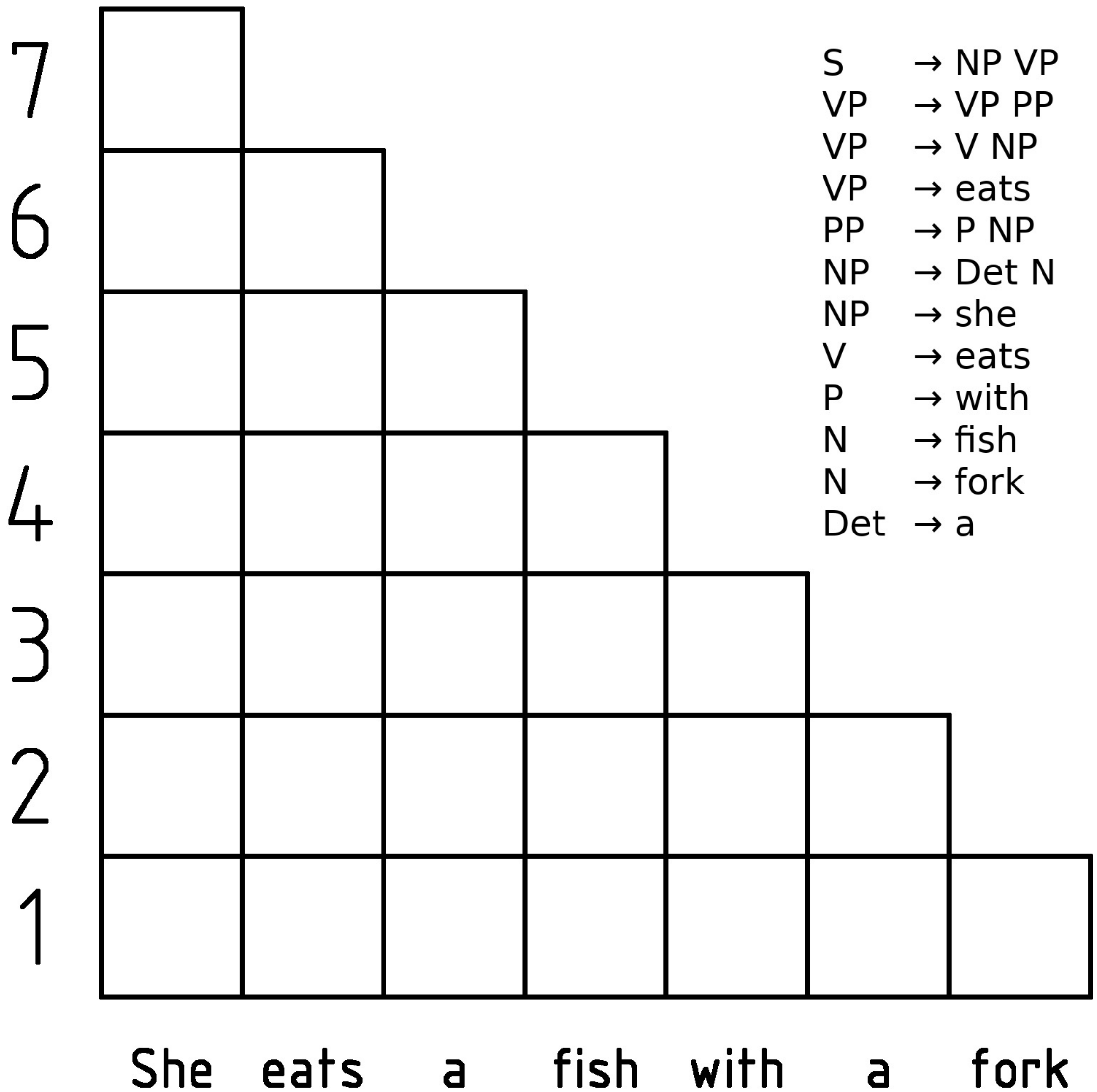
处理二义性语法



在action-goto表格中指定使用移进或是规约规则

规范项	Action			Goto
	num	+	eof	E
S_0	shift 1			2
S_1		reduce [2]	reduce [2]	
S_2		shift 3	accept	
S_3	shift 1			4
S_4		shift 3/~~reduce [2]~~		

通用自底向上CFG分析： CYK算法



S → NP VP
VP → VP PP
VP → V NP
VP → eats
PP → P NP
NP → Det N
NP → she
V → eats
P → with
N → fish
N → fork
Det → a

CYK解析算法伪代码

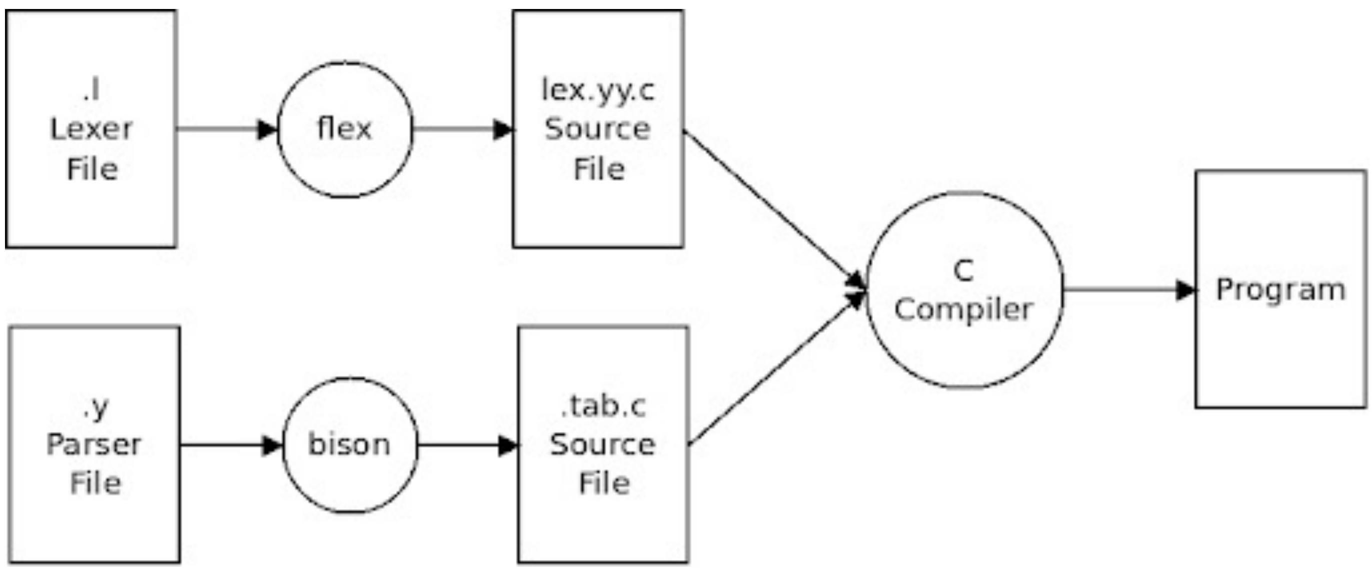
```
INIT:
    Gramma: R1,R2,…Rr
    String to parse: w = w1,w2,…,w1
    P[n,n,r] inited with false

Foreach i = 1 to n:
    Foreach Rr->ai
        P[1,i,r] = True
Foreach l = 2 to n:
    Foreach i = 1 to n-l+1:
        Foreach j = 1 to l-1:
            Foreach Rr->Ra Rb
                If P[j,i,a] and P[l-j,i+j,b]:
                    P[l,i,r]= True
If P[n,1,1]:
    w is a string of the language
Else:
    w is not a string of the language
```

四、语法分析工具

Bison

- 语法分析工具YACC(POSIX)/Bison (GNU)
  - 默认采用LALR(1)解析
  - 支持LR(1)等方法



Lex文件

```
%{ /* Lexer.l file */
#include "Expression.h"
#include "Parser.h"
#include <stdio.h> %}

%option outfile="Lexer.c"
header-file="Lexer.h"
%option warn nodefault
%option reentrant noyywrap never-interactive nounistd
%option bison-bridge

%%
[ \r\n\t]* { continue; /* Skip blanks. */ }
[0-9]+ { sscanf(yytext, "%d", &yylval->value); return TOKEN_NUMBER; }
"*" { return TOKEN_STAR; }
"+" { return TOKEN_PLUS; }
"(" { return TOKEN_LPAREN; }
")" { return TOKEN_RPAREN; }
. { continue; /* Ignore unexpected characters. */}

%%
int yyerror(const char *msg) {
    fprintf(stderr, "Error: %s\n", msg);
    return 0;
}
```

```
%{ /* * Parser.y file * */
#include "Expression.h"
#include "Parser.h"
#include "Lexer.h"
int yyerror(SExpression **expression, yyscan_t scanner, const char *msg) { /* Add error
handling routine as needed */ }
%}

%code requires { typedef void* yyscan_t; }
%output "Parser.c"
%defines "Parser.h"
%define api.pure
%lex-param { yyscan_t scanner }
%parse-param { SExpression **expression }
%parse-param { yyscan_t scanner }
%union { int value; SExpression *expression; }
%token TOKEN_LPAREN "("
%token TOKEN_RPAREN ")"
%token TOKEN_PLUS "+"
%token TOKEN_STAR "*"
%token <value> TOKEN_NUMBER "number"
%type <expression> expr

%left "+”
%left "*" %

%%
input : expr { *expression = $1; } ;
expr : expr[L] "+" expr[R] { $$ = createOperation( eADD, $L, $R ); }
    | expr[L] "*" expr[R] { $$ = createOperation( eMULTIPLY, $L, $R ); }
    | "(" expr[E] ")" { $$ = $E; }
    | "number" { $$ = createNumber($1);
} ; %%
```

计算器程序示例

```
main.c 编译器文件
Lexer.l 词法定义
Parser.y 语法定义
Expression.h 文件
Expression.c 功能函数
```

```c
int yyparse(SExpression **expression, yyscan_t scanner);

SExpression *getAST(const char *expr) {
    SExpression *expression;
    yyscan_t scanner;
    YY_BUFFER_STATE state;
    if (yylex_init(&scanner)) { /* could not initialize */
        return NULL;
    }
    state = yy_scan_string(expr, scanner);
    if (yyparse(&expression, scanner)) { /* error parsing */
        return NULL;
    }
    yy_delete_buffer(state, scanner);
    yylex_destroy(scanner);
    return expression;
}

int evaluate(SExpression *e) {
    switch (e->type) {
        case eVALUE: return e->value;
        case eMULTIPLY: return evaluate(e->left) * evaluate(e->right);
        case eADD: return evaluate(e->left) + evaluate(e->right);
        default: /* should not be here */ return 0;
    }
}

int main(void) {
    char test[] = "4 + 2*10 + 3*( 5 + 1 )";
    SExpression *e = getAST(test);
    int result = evaluate(e);
    printf("Result of '%s' is %d\n", test, result);
    deleteExpression(e);
    return 0;
}
```

Expression.h

```
/*Expression.c*/
#ifndef __EXPRESSION_H__
#define __EXPRESSION_H__

typedef enum tagEOperationType {
    eVALUE,
    eMULTIPLY,
    eADD
} EOperationType;

typedef struct tagSExpression {
    EOperationType type; /* /< type of operation */
    int value; /* /< valid only when type is eVALUE */
    struct tagSExpression *left; /* /< left side of the tree */
    struct tagSExpression *right; /* /< right side of the tree */
} SExpression;

SExpression *createNumber(int value);
SExpression *createOperation(EOperationType type, SExpression *left,
SExpression *right);
void deleteExpression(SExpression *b);

#endif
```

```c
/*Expression.c*/
#include "Expression.h"
#include <stdlib.h>

static SExpression *allocateExpression() {
    SExpression *b = (SExpression *)malloc(sizeof(SExpression));
    if (b == NULL) return NULL;
    b->type = eVALUE;
    b->value = 0;
    b->left = NULL;
    b->right = NULL;
    return b;
}

SExpression *createNumber(int value) {
    SExpression *b = allocateExpression();
    if (b == NULL) return NULL;
    b->type = eVALUE;
    b->value = value;
    return b;
}
SExpression *createOperation(EOperationType type, SExpression *left, SExpression *right) {
    SExpression *b = allocateExpression();
    if (b == NULL) return NULL;
    b->type = type;
    b->left = left;
    b->right = right;
    return b;
}

void deleteExpression(SExpression *b) {
    if (b == NULL) return;
    deleteExpression(b->left);
    deleteExpression(b->right);
    free(b);
}
```

总结

一、句式分析的基本概念

二、自顶向下分析

- LL(1)

三、自底向上分析

- SLR、LALR、LR(1)

四、语法分析工具