Lecture 3

句法分析：CFG和LL(1)文法

徐 辉

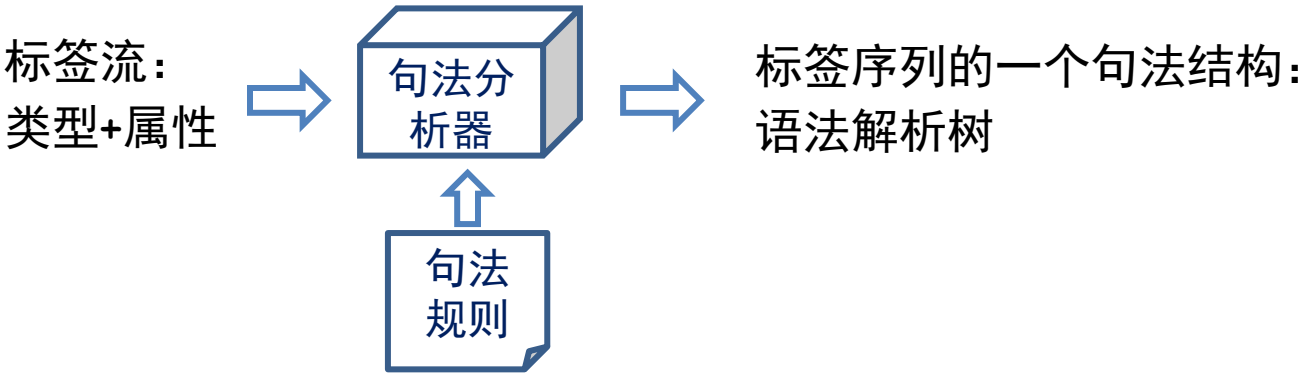
xuh@fudan.edu.cn

主要内容

- 一、上下文无关文法
- 二、语言分析问题
- 三、LL(1)文法和解析

句法解析：问题定义

- 给定一个句子和句法规则，找到可生成该句子的一个句法推导
- 通过词法分析已经将句子转换为了标签流
- 句法规则（句法）定义了：
  - 什么是句法分析器可接受的标签序列
  - 及其推导方式
- 语法 = 词法 + 句法
  - 下文将句法统称为语法，但省略了词法分析步骤



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。
  - 非终结符：X、Y、Z
  - 终结符（标签）：<BINOP>、<NUM>
    - 如考虑词法解析：字符
  - 字符串符号：α、β、γ
- 语法（grammar）包括一个开始符号S和多条推导规则（productions）
  - $S \rightarrow \beta$
  - …

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$1+2\times3$?

语法规则

[1] $E \rightarrow E + E$
[2] $E \rightarrow E - E$
[3] $E \rightarrow E \times E$
[4] $E \rightarrow E / E$
[5] $E \rightarrow$ <NUM>

推导

[1] $E \rightarrow E + E$
[5] $E \rightarrow 1 + E$
[3] $E \rightarrow 1 + E \times E$
[5] $E \rightarrow 1 + 2 \times E$
[5] $E \rightarrow 1 + 2 \times 3$

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$X \to \gamma$,
    - 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）：经典CFG语法表示形式
  - <symbol> ::= __expression__

回顾：括号匹配问题

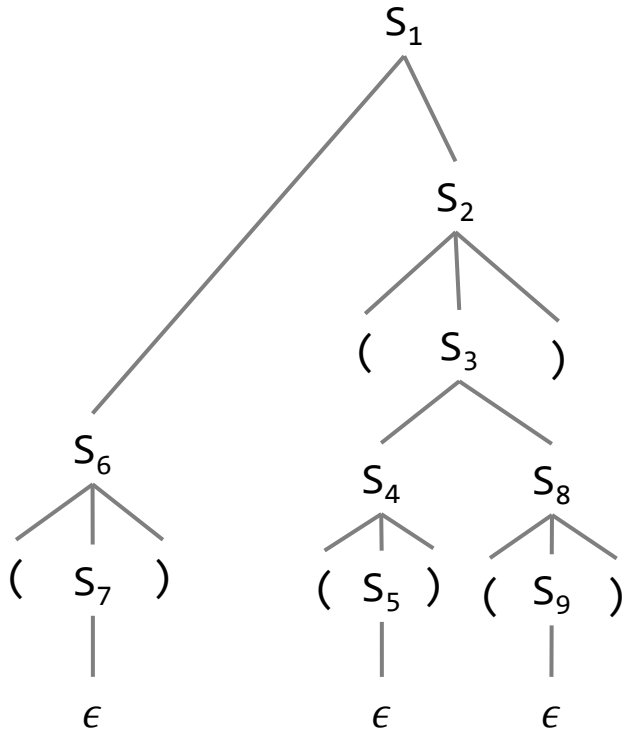
- 用CFG语法设计一套括号匹配规则
- 验证：()(()())是该语法的一个推导吗?

语法规则

[1] $S \rightarrow \epsilon$
[2] $\quad | \ (S)$
[3] $\quad | \ SS$

推导

[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] $S \rightarrow ()(()())$



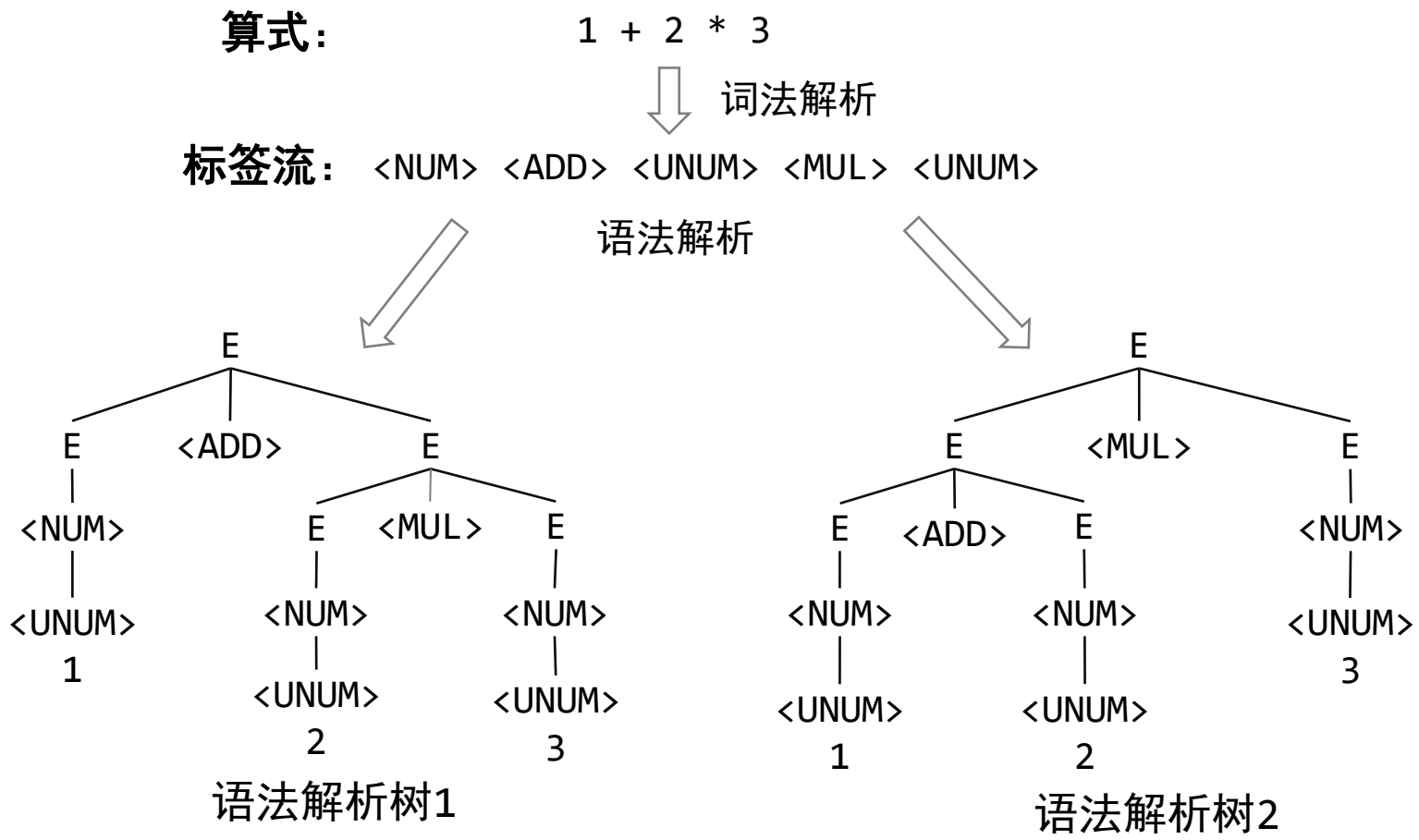
语法解析树

写出计算器的CFG文法

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

二义性问题（ambiguity）

- L(G)中的某个句子存在一个以上的最左（或最右）推导
- 语法解析树不同




If your DOG
does a POO
Please put it
in a litter bin.
Please help keep our
open spaces clean.

A programmer’s wife asks him to go to the grocery. She says "Get a gallon of milk. If they have eggs, get 12."

The programmer returns with 12 gallons of milk.

消除二义性

- 将运算符特性加入到语法规则中：
  - 优先级：^>×/÷>+/−
  - 结合性：×/÷>+/−左结合，^右结合

```
[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>
```

```
[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>
```

练习：语法设计

- 为下列语言设计语法规则：
  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

练习：语法设计

- 为描述正则语言的正则表达式语法设计一种CFG
  - 支持字符 [A-Za-z0-9]
  - 支持连接、或|、闭包*运算
  - 支持()
- 检查语法是否有二义性？

二、语言分析问题

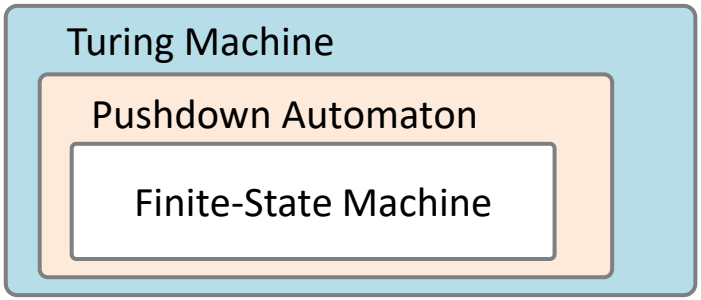
概念回顾

- 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。
  - 非终结符：X、Y、Z
  - 终结符（标签）：<BINOP>、<NUM>
  - 字符串符号：α、β、γ
- 语法（grammar）包括一个开始符号S和多条推导规则（productions）
  - $S \rightarrow \beta$
  - …

语言分析问题分类：按难度

Chomsky Hierarchy

类型	文法名称	自动机模型	生成式形式	语言示例
0 型	递归枚举	图灵机	无限制	
1 型	上下文敏感	Linear bounded TM	左侧可以多个符号 $\alpha S \rightarrow \beta$	$a^n b^n c^n$
2 型	上下文无关	下推自动机	左侧仅一个符号 $S \rightarrow \beta$	$a^n b^n$
3 型	正则	有穷自动机	右侧全部为终结符 $S \rightarrow <a><b>$	a^n



正则语言 VS 上下文无关语言

- 正则语言也可以用CFG规则形式表示：
  - $X \to \gamma$
  - $\gamma \to \gamma_1$
  - …
- 特点：右侧的非终结符均可替换为终结符

[1] $S \to \mathrm{A}|B$
[2] $A \to (0?\,1)^*$
[3] $B \to (1?\,0)^*$

$\Longrightarrow$ $S \to (0?\,1)^*|(1?\,0)^*$

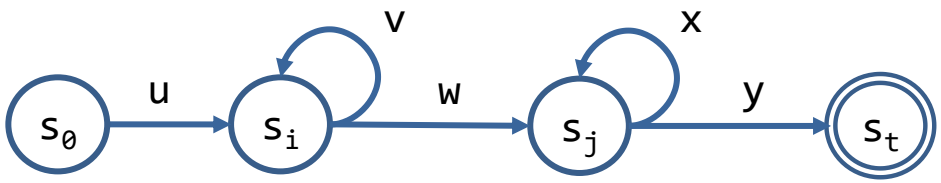
非CFG语言：上下文敏感语法

- $L = \{a^n b^n c^n, n > 0\}$不是CFG语言
- 如何定义？
  - 上下文敏感语法规则形式：$\alpha S \rightarrow \beta$

[1] $S \rightarrow aBC$
[2] $\quad | aSBC$
[3] $CB \rightarrow BC$
[4] $aB \rightarrow ab$
[5] $bB \rightarrow bb$
[6] $bC \rightarrow bc$
[7] $cC \rightarrow cc$

非CFG语言的泵引理

- CFG语言的泵引理（必要条件）：
  - 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：$uv^n w\epsilon^n\epsilon$



练习：下列语言是否为正则语言？

- 集合表示
  1) $L = \{a^n b^n | n \leq 100\}$
  2) $L = \{a^n | n \geq 1\}$
  3) $L = \{a^{2n} | n \geq 1\}$
  4) $L = \{a^p |$ p is prime$\}$
- Regex/CFG语法表示
  1) $S \rightarrow (0?\,1)^*$
  2) $S \rightarrow aT|\epsilon, T \rightarrow Sb$
  3) $S \rightarrow 0S1S|1S0S|\epsilon$

思考

1) 用正则表达式可以定义所有的正则语言吗？
2) 有穷自动机可以解析任意正则表达式吗？
3) 用CFG可以定义任意正则语言吗？
4) 用CFG可以定义任意上下文无关语言吗？
5) 用下推自动机可以解析任意正则表达式吗？
6) 用下推自动机可以解析任意CFG吗？
7) 用通用图灵机可以解析任意CFG吗？
8) 用通用图灵机可以解析任意程序吗？

三、LL(1)文法和解析

如何自动生成语法推导树？

- 应用语法规则（从左至右）逐步展开每个非终结符
- 如无二义性问题，则语法解析树唯一
- 如何精准判断当前应采用哪个展开式？避免盲目搜索
  - 预测解析（Predictive Parsing）
  - LL(1)文法：Left-to-Right，Leftmost，前瞻一个字符
- LL(1)文法的基本要求
  - 无左递归
  - 无回溯

左递归问题

- 一条规则中右侧的第一个符号与左侧符号相同
- 会导致搜索算法无限递归下去，不终止

```
[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>
```

消除左递归

- 引入新的非终结符

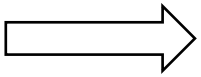
$$E \rightarrow E\ \alpha \mid \beta \quad \Rightarrow \quad E \rightarrow \beta\ E' \qquad E' \rightarrow \alpha\ E' \mid \epsilon$$

$$E \rightarrow E\ \alpha \mid \beta \mid \gamma \quad \Rightarrow \quad E \rightarrow \beta\ E' \mid \gamma E' \qquad E' \rightarrow \alpha\ E' \mid \epsilon$$

应用

```
[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>
```

消除左递归



```
[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>
```

注意间接左递归问题

$$
\begin{array}{l}
E \to \alpha \\
\alpha \to \beta + \\
\beta \to E
\end{array}
\quad \Longrightarrow \quad E \to E +
$$

无回溯语法

- **目的**：消除语法生成规则选择时的不确定性，避免回溯
- **思路**：如每个非终结符的任意两个生成式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
  - [1] $NT_1 \rightarrow NT_i \rightarrow \cdots \rightarrow \text{term}_1\ NT_p$
  - [2] $NT_1 \rightarrow NT_j \rightarrow \cdots \rightarrow \text{term}_2\ NT_q$

消除回溯：提取左因子

- 对一组生成式提取共同前缀

$$A \to \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_n | \gamma_1 | \ldots | \gamma_j \quad \Rightarrow \quad \begin{array}{l} A \to \alpha B \mid \gamma_1 | \ldots | \gamma_j \\ B \to \beta_1 \mid \beta_2 | \ldots | \beta_n \end{array}$$

应用

```
[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>
```

消除左递归

消除回溯语法

```
[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 E2'
[8] E2' → OP3 E2
[9]      | ϵ
[10] E3 → NUM
[11]     | <LPAR> E <RPAR>
[12] NUM → <UNUM>
[13]     | <SUB> <UNUM>
[14] OP1 → <ADD>
[15]     | <SUB>
[16] OP2 → <MUL>
[17]     | <DIV>
[18] OP3 → <POW>
```

First集合计算

- 对于生成式$A \to \beta_1\beta_2 \ldots \beta_n$ 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] E → E1 E'
[2] E' → OP1 E1 E'
[3]    | ϵ
[4] E1 → E2 E1'
[5] E1' → OP2 E2 E1'
[6]    | ϵ
[7] E2 → E3 E2'
[8] E2' → OP3 E2
[9]    | ϵ
[10] E3 → NUM
[11]    | <LPAR> E <RPAR>
[12] NUM → <UNUM>
[13]    | <SUB> <UNUM>
[14] OP1 → <ADD>
[15]     | <SUB>
[16] OP2 → <MUL>
[17]     | <DIV>
[18] OP3 → <POW>
```

	<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]			

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

	<UNUM>	<ADD>	<SUB>	<MUL>	<DIV>	<POW>	<LPAR>	<RPAR>	ϵ
E	[1]		[1]				[1]		
E’		[2]	[2]					[3]	[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]			

无回溯语法的必要性质

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

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

- 同一非终结符A 的任意两个语法推导$(A \to \beta_i)$和$\left(A \to \beta_j\right)$ 所产生的的首个终结符不能相同
- $First(\beta)$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$T \cup \{\epsilon, \text{eof}\}$
- 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

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

	<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]		

练习：

- 将正则表达式CFG改写为LL(1)语法并写出应用解析表

```
[1] REGEX → UNION
[2]       | CONCAT
[3] UNION → REGEX <OR> CONCAT
[4] CONCAT → CONCAT CLOSURE
[5]        | CLOSURE
[6] CLOSURE → ITEM <STAR>
[7]         | ITEM
[8] ITEM → <LPAR>REGEX<RPAR>
[9]      | <CHAR>
```