Lecture 2.3

计算器句式分析

徐 辉

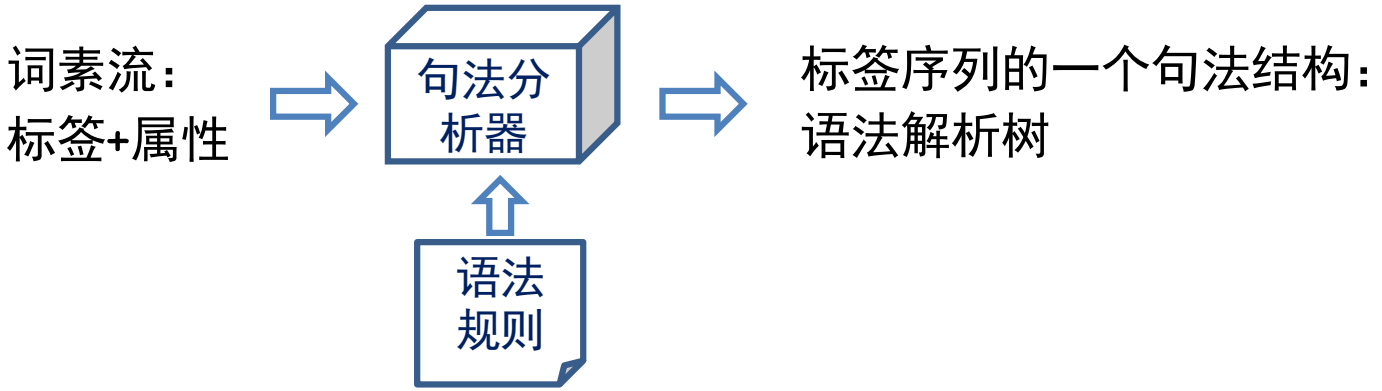
xuh@fudan.edu.cn

大纲

- 一、问题定义
- 二、上下文无关文法
- 三、自顶向下分析

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）可接受的标签序列
  - 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。
  - 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] $E \rightarrow 1 + 2 \times 3$

大纲

- 一、问题定义
- 二、上下文无关文法
- 三、自顶向下分析

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）是传统的上下文无关语法表示方法
  - <NT> ::= “abc”

括号匹配问题

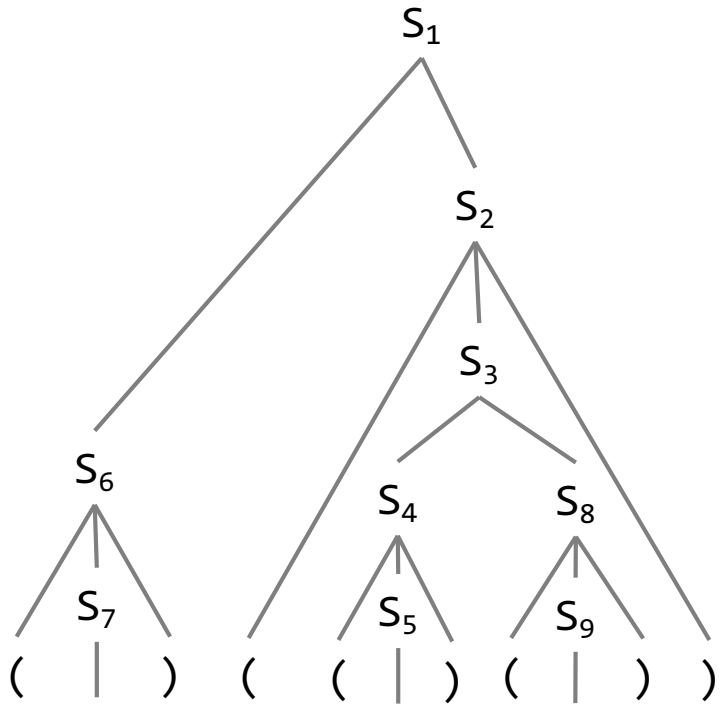
- ()(()())是该语法的一个推导吗?

语法规则

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

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



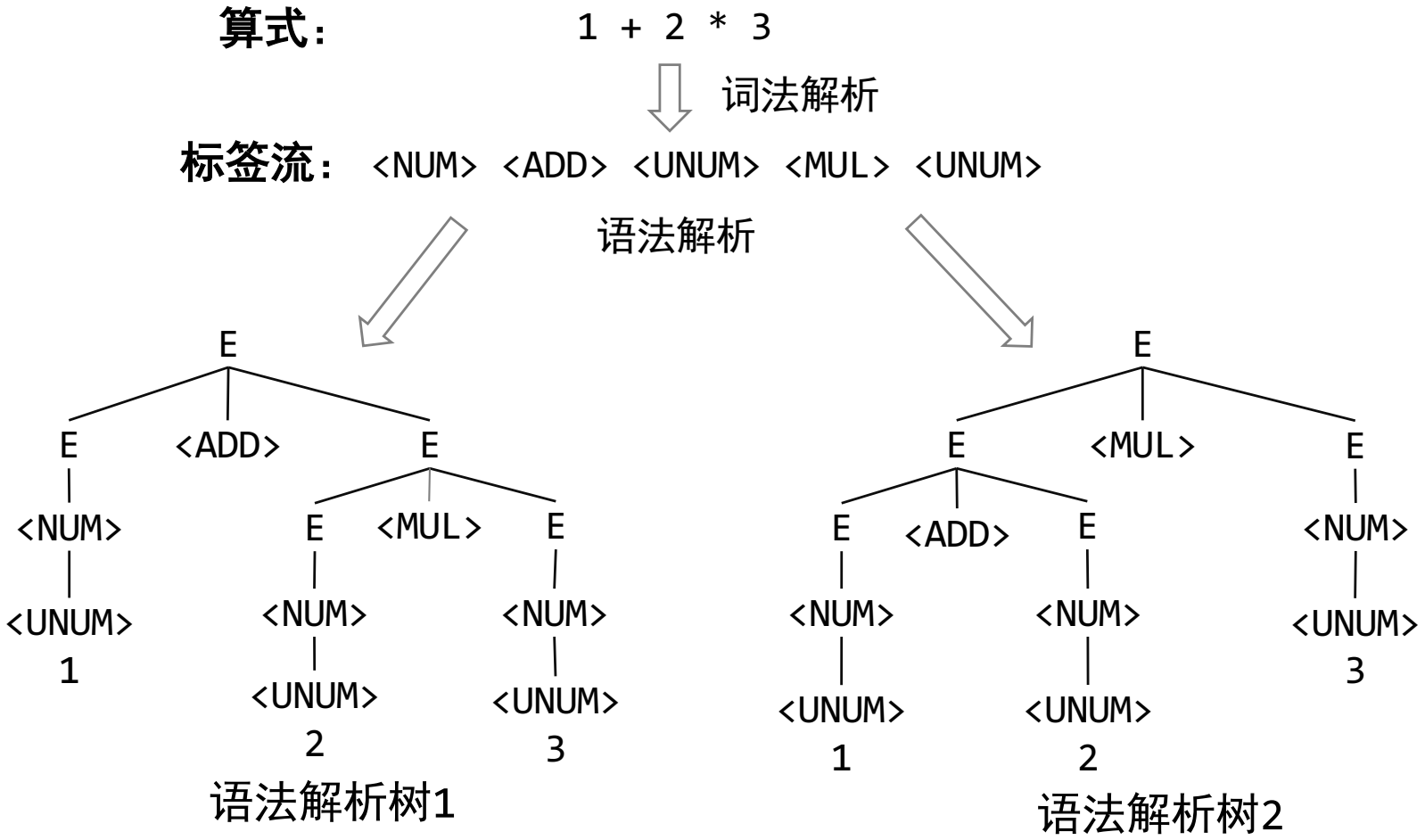
语法解析树

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 <EXP> E
[6]   | <LPAR> E <RPAR>
[7]   | NUM
[8] NUM → <UNUM>
[9]     | <SUB> <UNUM>
```

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




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

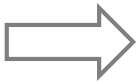
A programmer’s wife asks him to go to the grocery. She says "Get a gallon of milk. If they have eggs, get 12." The programmer returns with 12 gallons of milk.

A programmer is going to the grocery store and his wife tells him, "Buy a gallon of milk, and if there are eggs, buy a dozen." So the programmer goes, buys everything, and drives back to his house. Upon arrival, his wife angrily asks him, "Why did you get 13 gallons of milk?" The programmer says, "There were eggs!"

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

```
[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 → <EXP>
```

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？

大纲

- 一、问题定义
- 二、上下文无关文法
- 三、自顶向下分析

如何自动生成语法推导树？

- 从根节点开始，根据语法规则向下递归展开每个非终结符，直至最终生成的语法解释树与目标算式等价
- 对于任意一个无二义性问题的语法规则来说，该语法解析树最多只有一个
- 如何精准判断当前应采用哪个展开式？
  - 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

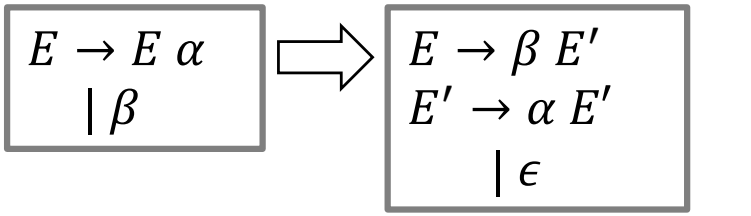
左递归问题

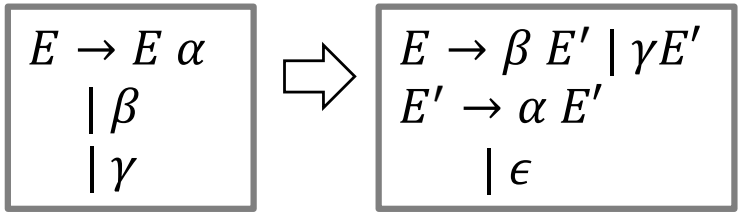
- 对CFG的一个规则来说，其右侧的第一个符号与左侧符号相同或者能够推导出左侧符号。
- 主要问题：可使搜索算法无限递归下去，不终止。

```
[1] 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 → <EXP>
```

消除左递归

- 引入新的非终结符，基本规则：

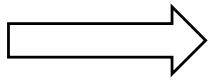




应用

```
[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 → <EXP>
```

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 → <EXP>
```

间接左递归问题

$$E \to \alpha$$
$$\alpha \to \beta +$$
$$\beta \to E$$

$$\Longrightarrow \quad E \to E +$$

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 → NT_j γ
      展开 NT_i → NT_j γ中的非终结符NT_j
  重写会造成NT_i左递归的规则
```

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] $NT_1 \rightarrow NT_i \rightarrow \cdots \rightarrow \text{term}_1\ NT_p$
  - [2] $NT_1 \rightarrow NT_j \rightarrow \cdots \rightarrow \text{term}_2\ NT_q$

消除回溯：提取左因子

- 对一组产生式提取并隔离共同前缀

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

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 → <EXP>
```

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 → <EXP>
```

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] 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 → <EXP>
```

	<UNUM>	<ADD>	<SUB>	<MUL>	<DIV>	<EXP>	<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 → <EXP>
```

	<UNUM>	<ADD>	<SUB>	<MUL>	<DIV>	<EXP>	<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

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

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

- 同一非终结符A 的任意两个语法推导$(A \to \beta_i)$和$(A \to \beta_j)$ 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>	<EXP>	<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

$$
\begin{aligned}
&< regex >::=< union > | < concat > \\
&< union >::=< regex > \text{"|"} < concat > \\
&< concat >::=< concat >< term > | < term > \\
&< term >::=< element >* | < element > \\
&< element >::= (< regex >)| < alphanum >
\end{aligned}
$$

小结

- 上下文无关文法
  - 消除二义性
- LL(1)文法及其解析算法
  - 无左递归
  - 无回溯