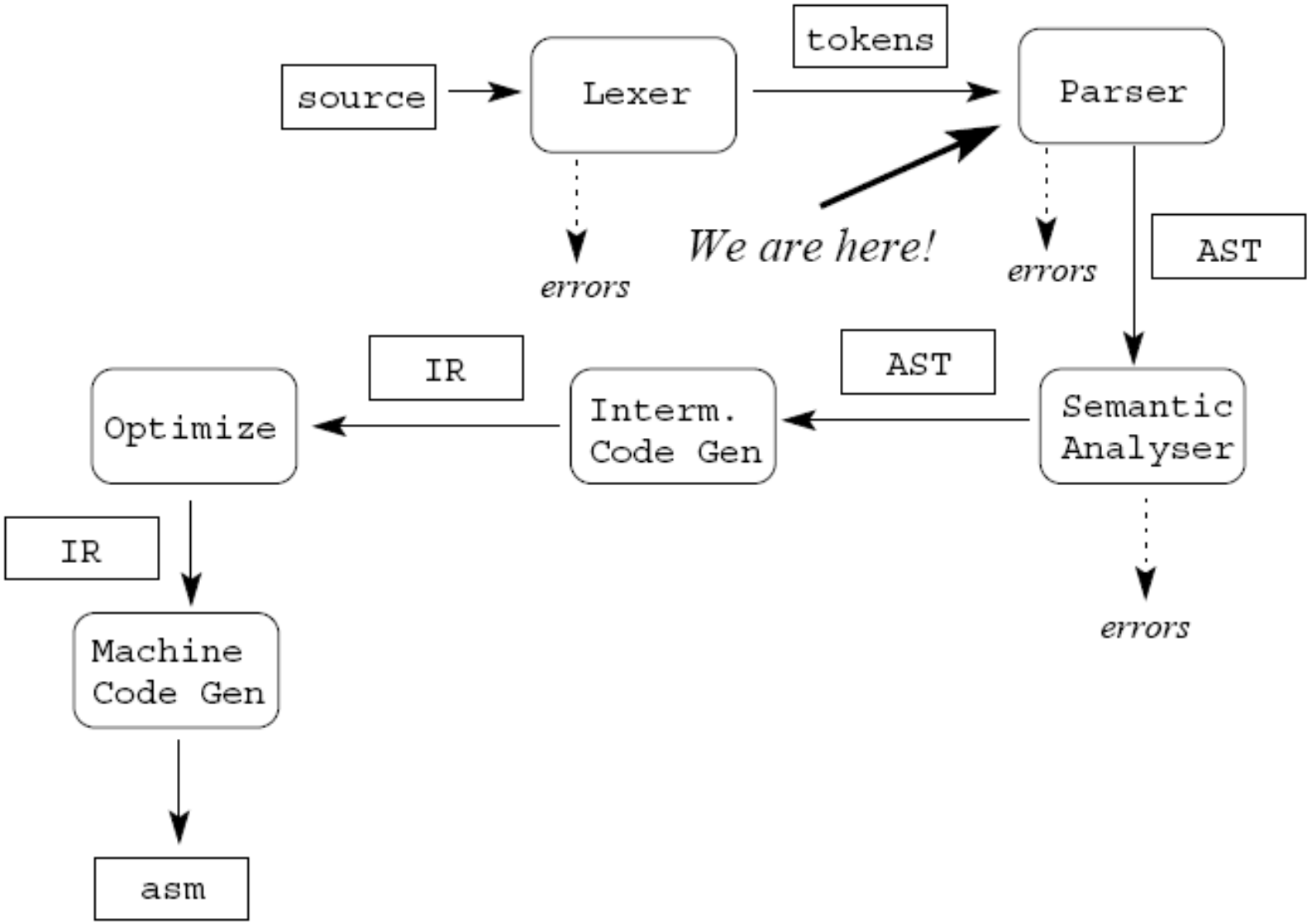
第4章 语法分析（上）

西北大学信息学院计算机科学系 付丽娜

语法分析


source
Lexer
tokens
Parser
errors
We are here!
errors
AST
IR
AST
Optimize
Interm.
Code Gen
Semantic
Analyser
errors
IR
Machine
Code Gen
asm

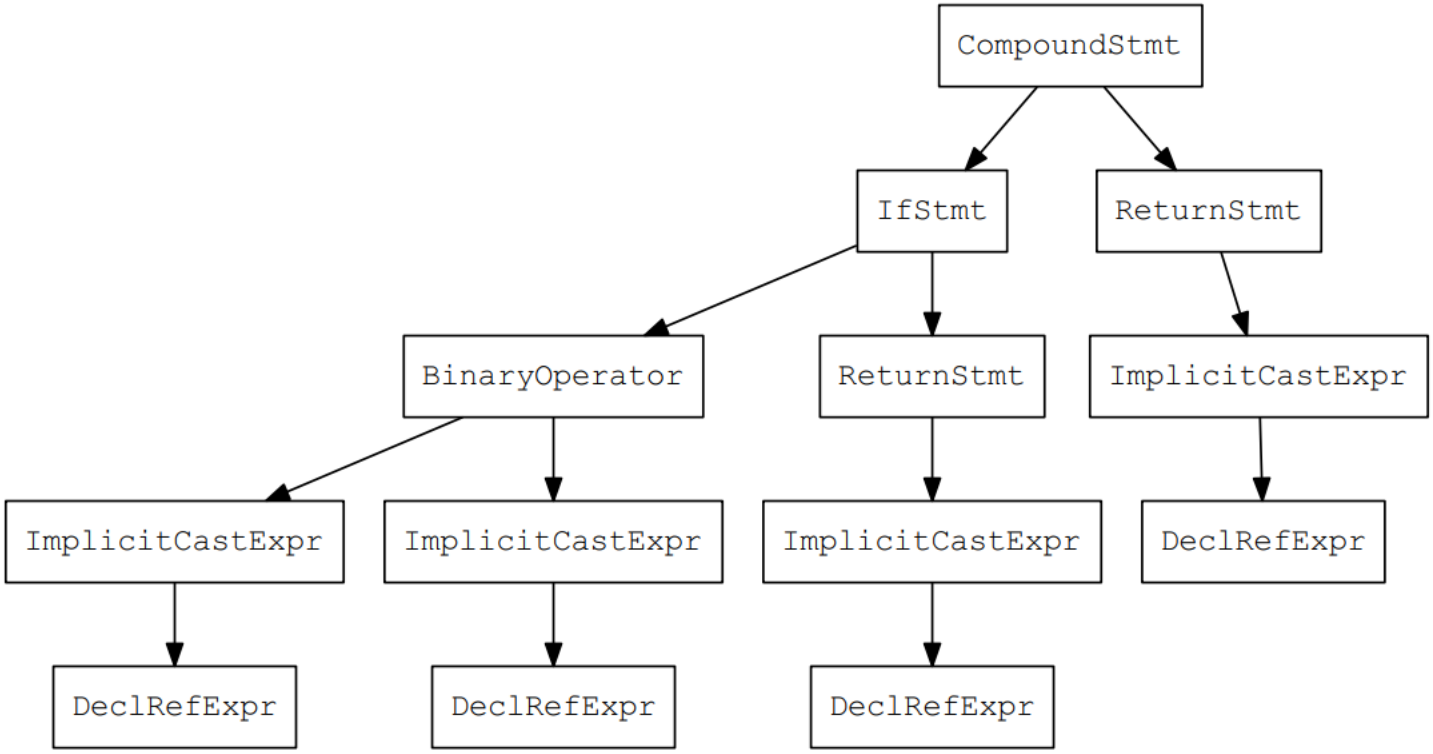
两个语法分析器的例子

❖ LLVM的C语言语法分析器

```
D:\temp>clang -fsyntax-only -Xclang -ast-dump min.c
TranslationUnitDecl 0x3ba1150 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x3ba1778 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantS
tring_tag'
| `-RecordType 0x3ba1620 'struct __NSConstantString_tag'
|   `-Record 0x3ba15d0 '__NSConstantString_tag'
|-TypedefDecl 0x3ba17c0 <<invalid sloc>> <invalid sloc> implicit size_t 'unsigned int'
| `-BuiltinType 0x3ba1230 'unsigned int'
|-TypedefDecl 0x3ba1820 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list 'char *'
| `-PointerType 0x3ba17f0 'char *'
|   `-BuiltinType 0x3ba11b0 'char'
`-FunctionDecl 0x3ba1940 <min.c:1:1, line:5:1> line:1:5 min 'int (int, int)'
  |-ParmVarDecl 0x3ba1868 <col:9, col:13> col:13 used a 'int'
  |-ParmVarDecl 0x3ba18b8 <col:16, col:20> col:20 used b 'int'
  `-CompoundStmt 0x3ba1aec <col:22, line:5:1>
    |-IfStmt 0x3ba1a9c <line:2:2, line:3:10>
    | |-BinaryOperator 0x3ba1a48 <line:2:5, col:7> 'int' '<'
    | | |-ImplicitCastExpr 0x3ba1a28 <col:5> 'int' <LValueToRValue>
    | | | `-DeclRefExpr 0x3ba19e8 <col:5> 'int' lvalue ParmVar 0x3ba1868 'a' 'int'
    | | `-ImplicitCastExpr 0x3ba1a38 <col:7> 'int' <LValueToRValue>
    | |   `-DeclRefExpr 0x3ba1a08 <col:7> 'int' lvalue ParmVar 0x3ba18b8 'b' 'int'
    | `-ReturnStmt 0x3ba1a90 <line:3:3, col:10>
    |   `-ImplicitCastExpr 0x3ba1a80 <col:10> 'int' <LValueToRValue>
    |     `-DeclRefExpr 0x3ba1a60 <col:10> 'int' lvalue ParmVar 0x3ba1868 'a' 'int'
    `-ReturnStmt 0x3ba1ae0 <line:4:2, col:9>
      `-ImplicitCastExpr 0x3ba1ad0 <col:9> 'int' <LValueToRValue>
        `-DeclRefExpr 0x3ba1ab0 <col:9> 'int' lvalue ParmVar 0x3ba18b8 'b' 'int'
```

两个语法分析器的例子

- LLVM的C语言语法分析器（查看AST视图）
  - clang -fsyntax-only -Xclang -ast-view d:\temp\min.c



两个语法分析器的例子

- Kaleidoscope（万花筒）程序——一个玩具语言的语法分析器
  - 忽略注释
  - 识别表达式
    - 常量表达式
    - 变量表达式
    - 函数调用表达式
    - 二元运算符连接的表达式
  - 识别函数声明
  - 识别函数定义（参数用空格分开）

Overview

- 本章讨论语法分析技术
  - 基本概念、手工实现、自动生成
- 程序设计语言的语法描述
  - 上下文无关文法或BNF表示法
- 文法为语言设计者和编译器编写者都提供了便利
  - 文法给出程序设计语言的精确易懂的语法规约
  - 对某些类型的文法，可以自动构造高效的语法分析器
  - 文法支持逐步加入可以完成新任务的新语言构造，从而迭代地演化和开发语言

Contents

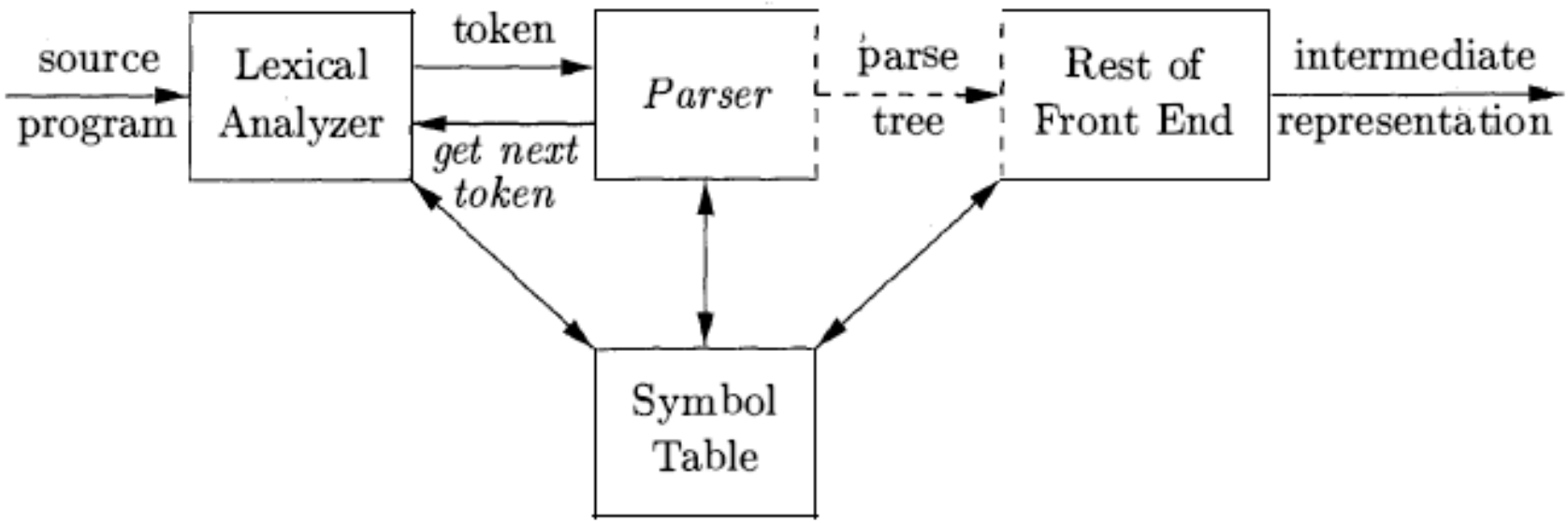
引论

语法分析概述

语法分析器的作用

- 语法分析器的作用
  - 语法分析器从词法分析器获得由词法单元组成的串，并验证这个串可以由源语言的文法生成（推导出来）
  - 语法分析器应该能够报告语法错误，并从常见错误中恢复并继续处理程序的其他部分
- 编译器模型中语法分析器的位置
  - 从概念上讲，对良构的程序，语法分析器构造出一棵语法分析树，并把它传递给编译器的其他部分进一步处理
  - 实际上，并不需要显式构造这棵语法分析树，因为对源程序的检查和翻译动作可以和语法分析过程交错完成
  - 因此，语法分析器和前端的其他部分可以用一个模块来实现

编译器模型中语法分析器的位置



语法分析器的分类

- 处理文法的语法分析器大体分3类
  - 自顶向下的
    - 从语法分析树的顶部（根结点）开始向底部（叶子结点）构造语法分析树
  - 自底向上的
    - 从叶子结点开始，逐渐向根结点方向构造语法分析树
- 自顶向下和自底向上方法的特点
  - 语法分析器的输入总是按照从左向右的方式扫描，每次扫描一个符号
  - LL文法和LR文法

代表性的文法

- 表达式文法的LR文法版本
  - 文法4.1，其中
  - E表示一组以+号分隔的项所组成的表达式
  - T表示由一组以*号分隔的因子所组成的项
  - F表示因子，可能是括号括起的表达式或标识符

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

代表性的文法

- 表达式文法的无左递归版本
  - 文法4.2
  - 用于自顶向下的语法分析

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

- 表达式文法的二义文法版本
  - 文法4.3
  - 约定+和*都是左结合，*的优先级高
  - E表示各种类型的表达式

$$E \rightarrow E + E \mid E * E \mid (\ E\ ) \mid \mathbf{id}$$

语法错误的处理

- 程序中不同层次的错误
  - 词法错误，包括标识符、关键字或运算符拼写错误和没有在字符串文本上正确地加引号
  - 语法错误，包括分号放错地方、花括号多余或缺失、C语言的*case*语句外围没有相应的*switch*
  - 语义错误，包括运算符和运算分量之间的类型不匹配，如返回类型为*void*的函数中出现了返回某个值的*return*语句
  - 逻辑错误，因程序员的错误推理而引起的任何错误，如C程序中应该用==的地方使用了=
- 如何处理语法错误通常由编译器的设计者决定
  - 大部分程序设计语言的规范没有规定编译器应如何处理错误

语法错误的处理

- 检测错误
  - 语法分析方法的精确性使我们可以高效检测出语法错误
    - 如LL和LR方法，具有可行前缀特性，一旦发现输入的某个前缀不能通过添加一些符号形成这个语言的串，就可以立刻检测到语法错误
- 报告错误
  - 报告在源程序的什么位置检测到错误

错误恢复策略

恐慌模式的恢复

- 语法分析器一旦发现错误就不断丢弃输入中的符号，一次丢弃一个符号，直到找到同步词法单元集合中的某个元素为止
  - 同步词法单元通常是分界符，如分号或}；它们在源程序中的作用清晰无二义
  - 编译器的设计者必须为源语言选择适当的同步词法单元
- 优点是简单，不会进入无限循环
- 缺点是会跳过大量输入，不检查被跳过部分的其他错误

错误恢复策略

❖ 短语层次的恢复

- 发现错误时，语法分析器在余下的输入上进行局部性纠正；
  - 常用方法包括将一个逗号替换为分号、删除一个多余的分号或插入一个遗漏的分号
- 优点是可以纠正任何输入串，但要小心选择替换方法以避免进入无限循环

错误恢复策略

❖ 错误产生式

- 预测可能遇到的常见错误，在当前语言的文法中加入特殊的产生式；这些产生式能产生含有错误的构造；
- 如果语法分析过程中使用了某个错误产生式，那么语法分析器就检测到了一个预期错误，并能据此生成适当的错误诊断信息，指出在输入中识别出的错误构造

上下文无关文法

上下文无关文法的正式定义

- 上下文无关文法（简称文法）由四个元素组成（四元组）
  - 一组**终结符号**，是组成串（Token串）的基本符号
    - “终结符号”和“**词法单元名字**”是同义词
  - 一组**非终结符号**，是表示串的**集合**的语法变量
    - 非终结符号间具有层次结构
  - 一个**开始符号**，是一个特殊的非终结符，表示的串的集合就是这个文法生成的**语言**
  - 一组**产生式**，描述将终结符号和非终结符号组合成串的方法
    ① 产生式头或左部
    ② 符号→，读作“定义为”，也写作“::=”
    ③ 产生体或右部，由零个或多个终结符号与非终结符号组成，描述了产生式左部非终结符对应的串的某种构造方法

例……上下文无关文法

- 定义简单的算术表达式的文法

$$
\begin{aligned}
\textit{expression} &\rightarrow \textit{expression} + \textit{term} \\
\textit{expression} &\rightarrow \textit{expression} - \textit{term} \\
\textit{expression} &\rightarrow \textit{term} \\
\textit{term} &\rightarrow \textit{term} * \textit{factor} \\
\textit{term} &\rightarrow \textit{term} / \textit{factor} \\
\textit{term} &\rightarrow \textit{factor} \\
\textit{factor} &\rightarrow (\textit{expression}) \\
\textit{factor} &\rightarrow \mathbf{id}
\end{aligned}
$$

符号表示的约定

- 终结符号
  - 字母表前面的小写字母，如a,b,c
  - 数字、运算符和标点符号
- 非终结符号
  - 字母表前面的大写字母，如A,B,C
  - 大小字母S，表示开始符号；大写字母E,F,T代表特定程序构造的非终结符
  - 小写、斜体名字，如*expr*或*stmt*
- 其他
  - 字母表后面的大写字母如X,Y,Z表示文法符号
  - 字母表后面的小写字母如$u,v,...,z$表示终结符号串（可为空）
  - 小写希腊字母如α,β,γ表示文法符号串

符号表示的约定

- 左部相同的一组产生式$A\rightarrow\alpha_1$，$A\rightarrow\alpha_2$，…$A\rightarrow\alpha_k$，可以写作 $A\rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$
- 除非特别说明，第一个产生式的左部文法符号就是开始符号

$$
\begin{array}{rcl}
expression & \rightarrow & expression + term \\
expression & \rightarrow & expression - term \\
expression & \rightarrow & term \\
term & \rightarrow & term * factor \\
term & \rightarrow & term / factor \\
term & \rightarrow & factor \\
factor & \rightarrow & (expression) \\
factor & \rightarrow & \mathbf{id}
\end{array}
$$

$$
\begin{array}{rcl}
E & \rightarrow & E + T \mid E - T \mid T \\
T & \rightarrow & T * F \mid T / F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

推导（Derivation）

- 推导和语法分析树
  - 将产生式看作重写（替换）规则，就可以从推导的角度精确描述构造语法分析树的方法
    - 用产生式的右部替换左部，称为推导（用于自顶向下分析）
    - 用产生式的左部替换右部，称为规约（用于自底向上分析）
- 符号“$\Rightarrow$”表示“通过一步推导出”
  - 如，$E \Rightarrow -E$，读作“E通过一步推导出$-E$”
  - 按照任意顺序对单个E不断应用各个产生式，得到一个替换的序列，如，$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(id)$，这个序列称为从E到-(id)的推导

推导

- 推导的一般性定义
  - 考虑一个文法符号序列中间的非终结符号A，比如$\alpha A\beta$，其中α和β是任意的文法符号串。假设$A\rightarrow\gamma$是一个产生式，那么我们写作 $\alpha A\beta \Rightarrow \alpha\gamma\beta$。
    - 符号$\Rightarrow$表示“通过一步推导出”
  - 当一个推导序列$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$将$\alpha_1$替换为$\alpha_n$，我们说$\alpha_1$推导出$\alpha_n$。
  - 符号$\Rightarrow^*$表示“经过0步或多步推导出”，符号$\Rightarrow^+$表示“经过一步或多步推导出”
    - 对任何串α，$\alpha \Rightarrow^* \alpha$
    - 如果$\alpha \Rightarrow^* \beta$且$\beta \Rightarrow \gamma$，那么$\alpha \Rightarrow^+ \gamma$

句型、句子和语言

- **句型**（*sentential form*）
  - 如果$S \Rightarrow^* \alpha$，其中S是文法G的开始符号，我们说α是G的一个句型
    - 句型可能包含终结符号或非终结符号，也可能是空串
- **句子**（*sentence*）
  - 文法G的一个句子是不包含非终结符号的句型。
- **文法生成的语言**
  - 一个文法生成的语言是它的**所有句子的集合**

    $L(G) = \{\alpha \mid S \Rightarrow^* \alpha$且$\alpha$是终结符号串$\}$
  - 可以由上下文无关文法生成的语言是**上下文无关语言**
  - 如果两个文法生成相同的语言，这两个文法被称为**等价**的，一个语言可能**对应了多个**文法

例……句型、句子和语言

$$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid \mathbf{id}$$

- -(id+id)是上面文法4.5的一个句子
  - 因为存在一个推导过程：

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

  - 串E,-E,-(E),-(E+E), -(id+E), -(id+id)都是这个文法的句型
  - -(id+id)是文法的一个句子

$$E \overset{*}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$$

最左推导和最右推导

- 每个推导步骤上要做两个选择
  - ① 要替换哪个非终结符号？
  - ② 用这个非终结符的哪个产生式右部替换？
- 最左推导
  - 总是选择每个句型的最左非终结符号
  - 如果$\alpha \Rightarrow \beta$是一个推导步骤，且被替换的是α中最左的非终结符号，写作 $\alpha \underset{lm}{\Rightarrow} \beta$
- 最右推导
  - 总是选择每个句型最右边的非终结符号，写作 $\alpha \underset{rm}{\Rightarrow} \beta$

最左推导和最右推导

- 最左（右）句型
  - 从开始符号出发，每一步经过最左（右）推导得到的句型
  - 最右推导也叫做规范推导

$$E \underset{lm}{\Rightarrow} -E \underset{lm}{\Rightarrow} -(E) \underset{lm}{\Rightarrow} -(E+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$$

$$E \underset{rm}{\Rightarrow} -E \underset{rm}{\Rightarrow} -(E) \underset{rm}{\Rightarrow} -(E+E) \underset{rm}{\Rightarrow} -(E+\mathbf{id}) \underset{rm}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$$

语法分析树和推导

- 语法分析树是推导的图形表示形式
  - 过滤掉了推导过程中对非终结符应用产生式的顺序
  - 语法分析树的每个内部结点都对应了一个产生式的应用
    - 内部结点的标号是产生式左部的非终结符号
    - 该结点的子结点的标号从左向右组成了在推导过程中替换这个非终结符的右部
- 语法分析树的结果或边缘
  - 一棵语法分析树的叶子结点的标号既可以是非终结符，也可以是终结符，从左到右排列这些符号就可以得到一个句型，称为这棵树的结果或边缘

例……语法分析树和推导

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

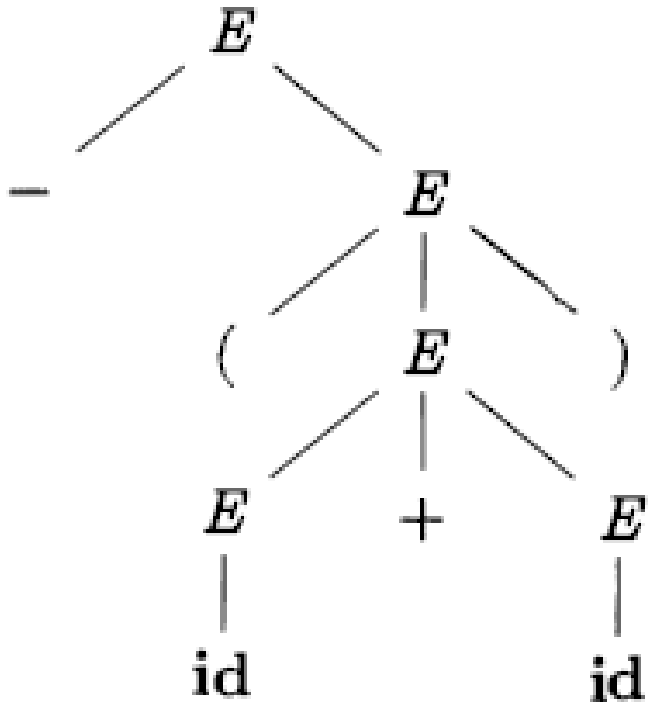
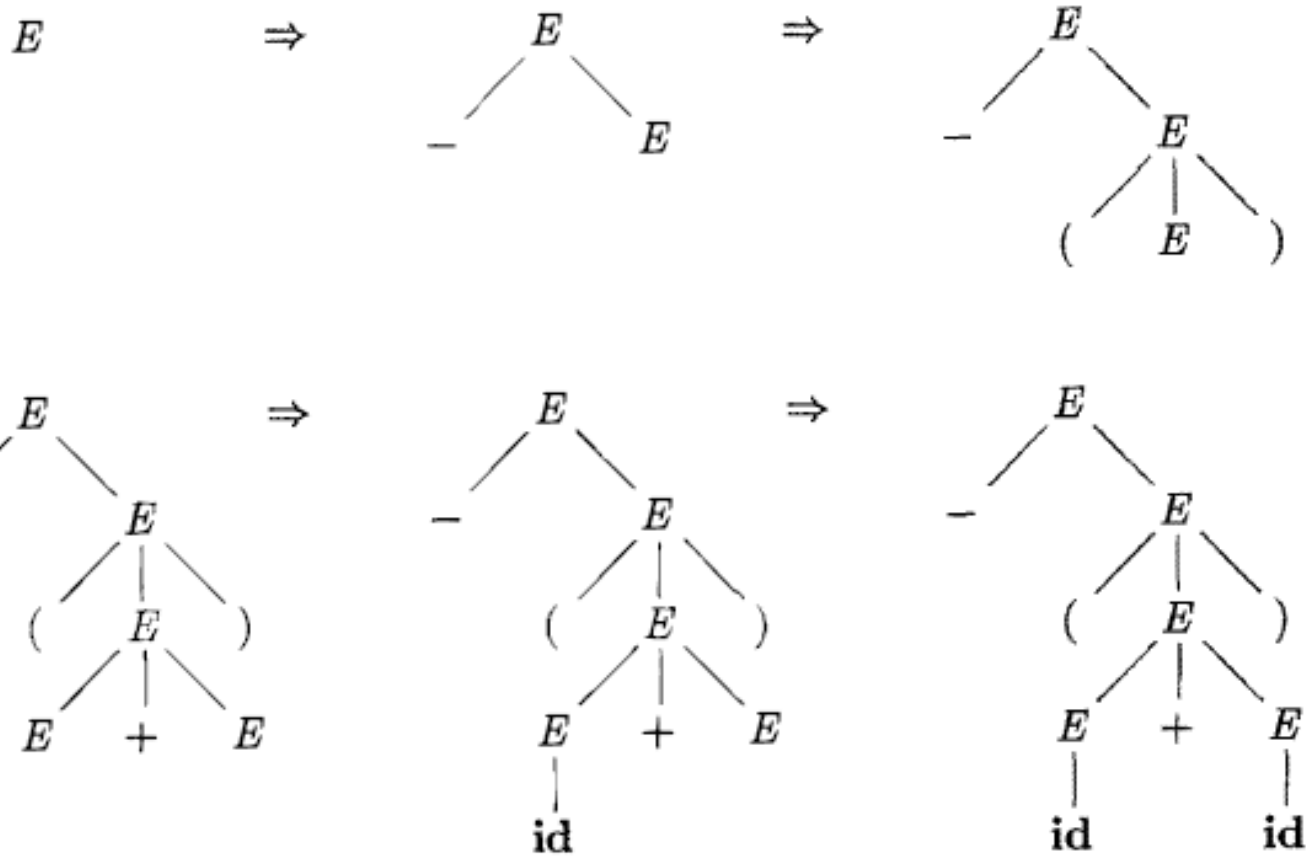


Figure 4.3: Parse tree for $-(\mathbf{id}+\mathbf{id})$

例……语法分析树和推导

- 根据推导构造语法分析树的序列

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$



语法分析树和推导

- 语法分析树和推导的对应关系
  - 如果文法无二义，那么一个句型的语法分析树表示了该句型的所有可能的推导，包括最左推导和最右推导，即一个句型的语法分析树是唯一的
  - 对无二义的文法，一个句型的最左推导/最右推导是唯一的
- 对于无二义性文法，语法分析树和最左推导/最右推导之间存在一对一的关系，所以可以通过构造最左推导或最右推导来进行语法分析

二义性

- 定义
  - 如果一个文法可以为某个句子生成多棵语法分析树，那么它就是二义性的
  - 二义性文法就是对同一个句子有多个最左推导或多个最右推导
  - 错误的表述：
    - 因为句子α存在多个推导，所以文法是二义性的
    - 因为句子α有不同的最左推导和最右推导，所以文法是二义性的
    - 因为文法存在多个不同的最左推导（最右推导），所以文法是二义性的

例……二义性

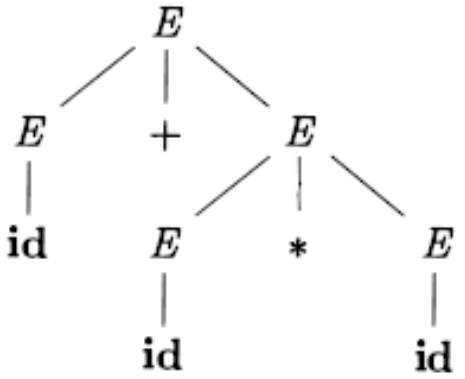
- 文法4.3是二义性文法

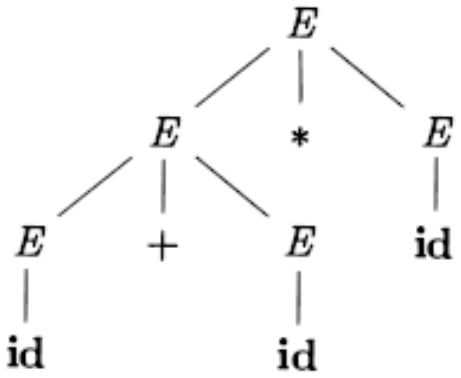
$$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$$

  - 句子*id+id*id*有两个不同的最左推导

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow \mathbf{id} + E \\ &\Rightarrow \mathbf{id} + E * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned} \qquad \begin{aligned} E &\Rightarrow E * E \\ &\Rightarrow E + E * E \\ &\Rightarrow \mathbf{id} + E * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$

  - 或者说，句子*id+id*id*有两棵不同的语法分析树





验证文法生成的语言

- 证明文法G生成语言L的过程可以分成两个部分
  1. 证明G生成的每个串都在L中
  2. 反向证明L中的每个串都确实能由G生成
- 例：
  - 证明文法 $S \rightarrow (S)\,S \mid \varepsilon$

  生成了所有具有对称括号对的串并且只生成这样的串

  1. 用关于推导步数n的**归纳法**证明从S推导出的每个句子都是括号对称的
  2. 用关于串的长度的**归纳法**证明每个括号对称的串都可以从S推导得到

为了证明从 S 推导出的每个句子都是括号对称的，我们对推导步数 n 进行归纳。

基础：基础是 $n=1$。唯一可以从 S 经过一步推导得到的终结符号串是空串，它当然是括号对称的。

归纳步骤：现在假设所有步数少于 n 的推导都得到括号对称的句子，并考虑一个恰巧有 n 步的最左推导。这样的推导必然具有如下形式：

$$S \underset{lm}{\Rightarrow} (S)S \overset{*}{\underset{lm}{\Rightarrow}} (x)S \overset{*}{\underset{lm}{\Rightarrow}} (x)y$$

从 S 到 x 和 y 的推导过程都少于 n 步，因此根据归纳假设，x 和 y 都是括号对称的。因此，串 $(x)y$ 必然是括号对称的。也就是说，它具有相同数量的左括号和右括号，并且它的每个前缀中的左括号不少于右括号。

现在已经证明了可以从 S 推导出的任何串都是括号对称的，接下来我们必须证明每个括号对称的串都可以从 S 推导得到。为了证明这一点，我们对串的长度进行归纳。

基础：如果串的长度是 0，它必然是 ϵ。这个串是括号对称的，且可以从 S 推导得到。

归纳步骤：首先请注意，每个括号对称的串的长度是偶数。假设每个长度小于 $2n$ 的括号对称的串都能够从 S 推导得到，并考虑一个长度为 $2n(n \geqslant 1)$ 的括号对称的串 w。w 一定以左括号开头。令 (x) 是 w 的最短的、左括号个数和右括号个数相同的非空前缀，那么 w 可以写成 $w=(x)y$ 的形式，其中 x 和 y 都是括号对称的。因为 x 和 y 的长度都小于 $2n$，根据归纳假设，它们可以从 S 推导得到。因此，我们可以找到一个如下形式的推导：

$$S \Rightarrow (S)S \overset{*}{\Rightarrow} (x)S \overset{*}{\Rightarrow} (x)y$$

它证明 $w=(x)y$ 也可以从 S 推导得到。 □

例……文法和语言

- 例1：文法G1定义的语言

  $S \rightarrow bA$

  $A \rightarrow aA \mid a$

  解答：可以分析得出 $L(G1) = \{ba^n \mid n \geq 1\}$

- 例2：文法G2定义的语言

  $S \rightarrow AB$

  $A \rightarrow aA \mid a$

  $B \rightarrow bB \mid b$

  解答：可以分析得出 $L(G2) = \{a^m b^n \mid m,n \geq 1\}$

例……文法和语言

- 例3：构造文法G3使得

  $L(G3) = \{a^n b^n \mid n \geq 1\}$

  解答：构造文法G3如下

  $S \rightarrow aSb \mid ab$

- 例4：构造文法G4使得

  $L(G4) = \{a^n b^m \mid n \geq 1, m \geq 0\}$

  解答：构造文法G4如下

  $S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \varepsilon$

简单程序语言文法……1

Program ::= BEGIN Stat END

Stat ::= ident := Expr

Expr ::= Expr + Expr |
Expr * Expr |
ident | number

BEGIN a := 5 + 4 * 3 END

是该文法的一个句子

因为存在下面的推导

Program ⇒ BEGIN Stat END
⇒ BEGIN ident := Expr END
⇒ BEGIN "a" := Expr END
⇒ BEGIN "a" := Expr + Expr END
⇒ BEGIN "a" := 5 + Expr END
⇒ BEGIN "a" := 5 + Expr * Expr END
⇒ BEGIN "a" := 5 + 4 * Expr END
⇒ BEGIN "a" := 5 + 4 * 3 END

简单程序语言文法……2

Program ::= program ident ; DeclSeq begin StatSeq end .

DeclSeq ::= Decl ; DeclSeq | ϵ

Decl ::= var ident : ident

Stat ::= ident := Expr | if Expr then StatSeq else StatSeq

StatSeq ::= Stat ; StatSeq | ϵ

Expr ::= ident | const

Example:

```
PROGRAM P;
   VAR I : INTEGER;
   VAR C : CHAR;
   VAR J : INTEGER;
BEGIN I := 6; J := I; END.
```

练习4.2（作业）

练习 4.2.1：考虑上下文无关文法：

$$S \rightarrow S\ S\ +\ |\ S\ S\ *\ |\ a$$

以及串 $aa+a*$。

1）给出这个串的一个最左推导。

2）给出这个串的一个最右推导。

3）给出这个串的一棵语法分析树。

！4）这个文法是否为二义性的？证明你的回答。

！5）描述这个文法生成的语言。

练习4.2（作业）

练习 4.2.2：对下列的每一对文法和串重复练习 4.2.1。

1）$S \rightarrow 0\ S\ 1 \mid 0\ 1$ 和串 000111。

2）$S \rightarrow +\ S\ S \mid *\ S\ S \mid a$ 和串 $+\ *\ aaa$。

!3）$S \rightarrow S\ (\ S\ )\ S \mid \epsilon$ 和串 (()())。

!4）$S \rightarrow S\ +\ S \mid S\ S \mid (\ S\ ) \mid S\ * \mid a$ 和串 $(a+a)*a$。

!5）$S \rightarrow (\ L\ ) \mid a$ 以及 $L \rightarrow L,\ S \mid S$ 和串 $((a,\ a),\ a,\ (a))$。

!!6）$S \rightarrow a\ S\ b\ S \mid b\ S\ a\ S \mid \epsilon$ 和串 $aabbab$。

!7）下面的布尔表达式对应的文法：

bexpr → *bexpr* **or** *bterm* | *bterm*

bterm → *bterm* **and** *bfactor* | *bfactor*

bfactor → **not** *bfactor* | (*bexpr*) | **true** | **false**

文法和语言……补充练习

- 下列文法定义的语言是什么？

① $S \rightarrow aS \mid bS \mid \varepsilon$

② $S \rightarrow aS \mid Sb \mid \varepsilon$

③ $S \rightarrow aSb \mid \varepsilon$

④ $S \rightarrow abS \mid \varepsilon$

⑤ $S \rightarrow 10A01 \mid aA \quad A \rightarrow a \mid bA$

⑥ $S \rightarrow SS \mid 0A1 \quad A \rightarrow 0A1 \mid \varepsilon$

⑦ $S \rightarrow bAdc \quad A \rightarrow AS \mid a$

补充练习……解答

① $S \rightarrow aS \mid bS \mid \varepsilon$　　a,b构成的串

② $S \rightarrow aS \mid Sb \mid \varepsilon$　　$L = \{a^n b^m \mid n \geq 0, m \geq 0\}$

③ $S \rightarrow aSb \mid \varepsilon$　　$L = \{a^n b^n \mid n \geq 0\}$

④ $S \rightarrow abS \mid \varepsilon$　　$L = \{(ab)^n \mid n \geq 0\}$

⑤ $S \rightarrow 10S01 \mid aA$　$A \rightarrow a \mid bA$

$L = \{(10)^m ab^n a (01)^m \mid m,n>=0\}$ 句子以相同长度的10串开头、01串结尾且中间嵌有a开头a结尾的串

⑥ $S \rightarrow SS \mid 0A1$　$A \rightarrow 0A1 \mid \varepsilon$（作业）

句子中0、1个数相同，并且若干连续0后必然紧接数量相同的连续的1

⑦ $S \rightarrow bAdc$　$A \rightarrow AS \mid a$（作业）

句子以ba开头、dc结尾的串，且ba和dc的个数相同

上下文无关文法……补充练习1

- 构造描述下列语言的文法：
  - $L(G) = \{a^n b^n c^m \mid n>=1, m>=0\}$
  - $L(G) = \{a^n b^{2n} c^m \mid n>=1, m>=0\}$
  - $L(G) = \{a^n c^{2m+1} b^n \mid n>=0, m>=0\}$
  - $L(G) = \{a^n b^{2m+1} c a^{2m+1} b^n \mid n>=1, m>=0$
  - **$L(G) = \{a^{n+1} b^n \mid n>=1\}$**
  - **$L(G) = \{a^{n+1} b^{n+2} \mid n>=1\}$**
  - **$L(G) = \{a^m b^k c^m a^n b^n \mid k>=2, m>=1, n>=0\}$**

补充练习1……解答

- 构造描述下列语言的文法：
  - $L(G) = \{a^n b^n c^m \mid n>=1, m>=0\}$ $S \rightarrow XY \quad X \rightarrow aXb \mid ab \quad Y \rightarrow cY \mid \varepsilon$
  - $L(G) = \{a^n b^{2n} c^m \mid n>=1, m>=0\}$ $S \rightarrow XY \quad X \rightarrow aXbb \mid abb \quad Y \rightarrow cY \mid \varepsilon$
  - $L(G) = \{a^n c^{2m+1} b^n \mid n>=0, m>=0\}$ $S \rightarrow aSb \mid C \quad C \rightarrow ccC \mid c$
  - $L(G) = \{a^n b^{2m+1} c a^{2m+1} b^n \mid n>=1, m>=0\}$

    $S \rightarrow aSb \mid aTb \quad T \rightarrow bca \mid bbTaa$
  - **$L(G) = \{a^{n+1} b^n \mid n>=1\}$ （作业）**

    $S \rightarrow aSb \mid aab$
  - **$L(G) = \{a^{n+1} b^{n+2} \mid n>=1\}$ （作业）**

    $S \rightarrow aSb \mid aabbb$
  - **$L(G) = \{a^m b^k c^m a^n b^n \mid k>=2, m>=1, n>=0\}$ （作业）**

    $S \rightarrow XZ \quad X \rightarrow aXc \mid aYc \quad Y \rightarrow bY \mid bb \quad Z \rightarrow aZb \mid \varepsilon$

上下文无关文法……补充练习2

- 构造一个上下文无关文法，描述具有中心元素t的镜像结构语言
  - L(G) = $\{\alpha t \alpha^R \mid \alpha \in \{0,1\}^*, \alpha^R$为$\alpha$的逆串$\}$
- 构造一个上下文无关文法，描述无中心元素的镜像结构语言

  L(G) = $\{\alpha\alpha^R \mid \alpha \in \{0,1\}^+, \alpha^R$为$\alpha$的逆串$\}$
- 构造一个上下文无关文法，产生语言

  L(G) = $\{\beta \mid \beta \in \{a,b,c\}^*$，且β中符号的排列是对称的$\}$

补充练习2……解答

- L(G) = {$\alpha t \alpha^R \mid \alpha \in \{0,1\}^*$，$\alpha^R$为$\alpha$的逆串}（对称）
  - **$S \rightarrow 1S1 \mid 0S0 \mid t$**
- L(G) = {$\alpha \alpha^R \mid \alpha \in \{0,1\}^+$，$\alpha^R$为$\alpha$的逆串}
  - **$S \rightarrow 1S1 \mid 0S0 \mid 00 \mid 11$**
  - 无中心元素的镜像结构语言（回文、对称与此类似）
- L(G) = {$\beta \mid \beta \in \{a,b,c\}^*$，且β中符号的排列是对称的}
  - **$S \rightarrow aSa \mid bSb \mid cSc \mid a \mid b \mid c \mid \varepsilon$**

练习4.2

练习4.2.3：为下面的语言设计文法：

1）所有由0和1组成的并且每个0之后都至少跟着一个1的串的集合。

！2）所有由0和1组成的回文（palindrome）的集合，也就是从前面和从后面读结果都相同的串的集合。

！3）所有由0和1组成的具有相同多个0和1的串的集合。

!！4）所有由0和1组成的并且0的个数和1的个数不同的串的集合。

！5）所有由0和1组成的且其中不包含子串011的串的集合。

!！6）所有由0和1组成的形如xy的串的集合，其中$x \neq y$且x和y等长。

文法分类

词法分析和语法分析

❖ 为什么用正则式而不是文法来定义语言的词法规则？

- 能够用正则表达式描述的语言都可以用文法描述，但是

① 将语言的语法结构分为词法部分和非词法部分便于将编译器前端模块化，将前端分解为两个大小适中的组件

② 语言的词法规则通常很简单，不需要使用像文法这样功能强大的表示方法来描述这些规则

✓ 与文法相比，正则表达式通常提供了更简洁更易于理解的表示词法单元的方法

✓ 根据正则表达式自动构造得到的词法分析器的效率要高于任意文法自动构造得到的分析器

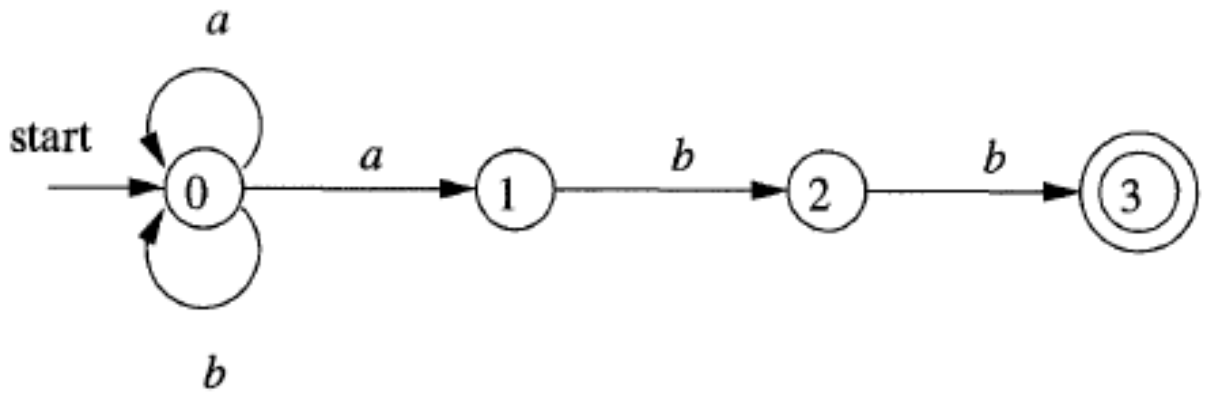
上下文无关文法和正则表达式

- 文法是比正则表达式表达能力更强的表示方法
  - 每个可以用正则表达式描述的构造都可以用文法来描述，反之不成立
  - 每个正则语言都是一个上下文无关语言，反之不成立
- 给定DFA或NFA，可以构造出与之等价的文法

1）对于 NFA 的每个状态 i，创建一个非终结符号 A_i。
2）如果状态 i 有一个在输入 a 上到达状态 j 的转换，则加入产生式 $A_i \rightarrow aA_j$。
如果状态 i 在输入 ϵ 上到达状态 j，则加入产生式 $A_i \rightarrow A_j$。
3）如果 i 是一个接受状态，则加入产生式 $A_i \rightarrow \epsilon$。
4）如果 i 是自动机的开始状态，令 A_i 为所得文法的开始符号。

例……上下文无关文法和正则表达式

- 正则表达式$(a|b)^*abb$的NFA和文法



$$
\begin{aligned}
A_0 &\rightarrow aA_0 \mid bA_0 \mid aA_1 \\
A_1 &\rightarrow bA_2 \\
A_2 &\rightarrow bA_3 \\
A_3 &\rightarrow \epsilon
\end{aligned}
$$

上下文无关文法和正则表达式

- 语言L=$\{a^n b^n \mid n \geq 1\}$可以用文法描述，但不能用正则表达式描述

用反证法来说明这一点。假设 L 是用某个正则表达式定义的语言。我们可以构造一个具有有穷多个状态(比如说 k 个状态) 的 DFA D 来接受 L。

因为 D 只有 k 个状态，对于一个以多于 k 个 a 开头的输入，D 一定会进入某个状态两次，假设这个状态是 s_i，如图 4-6 所示。

假设从 s_i 返回到其自身的路径的标号序列是 a^{j-i}。

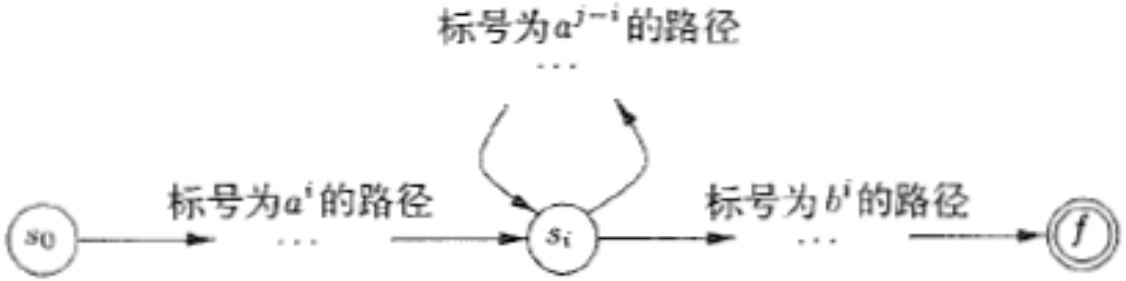
因为 $a^i b^i$ 在这个语言中，因此必然存在一条标号为 b^i 从 s_i 到某个接受状态 f 的路径。

但是，一定还存在一条从开始状态 s_0 出发，经过 s_i 最后到达 f 的路径，它的标号序列为 $a^j b^i$

因此，D 也接受 $a^j b^i$，

但 $a^j b^i$ 这个串不在语言 L 中

这和 L 是 D 所接受的语言这个假设矛盾。



文法和语言的分类

- 乔姆斯基（Chomsky）给出了文法的定义：

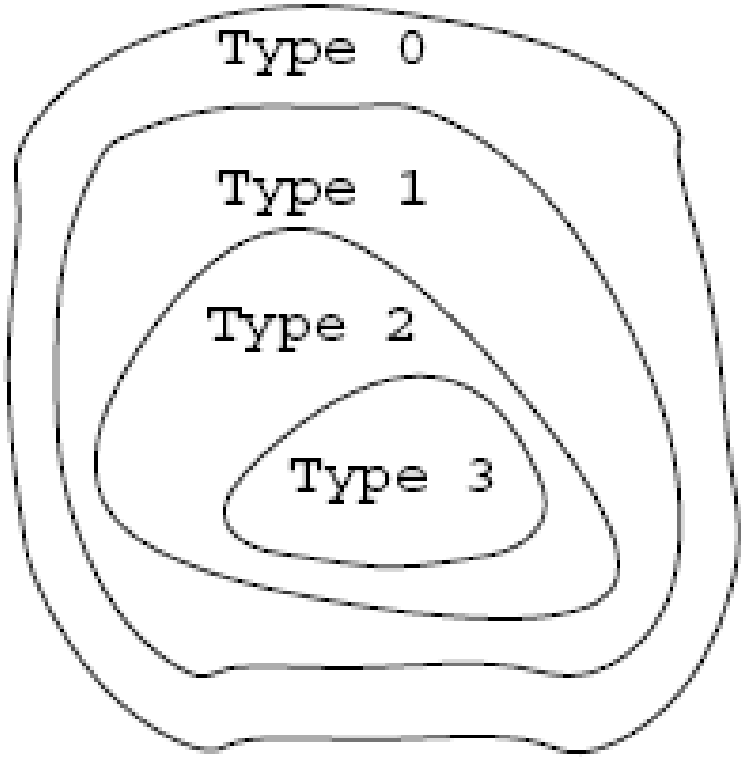
  $G = (V_N, V_T, P, S)$ （$V_N \cap V_T = \phi$， $V_N \cup V_T = V$）

- 对产生式的形式给以不同的限制可定义四类基本文法，分别称为0型文法，1型文法，2型文法，3型文法

TYPE	GRAMMAR	PSR
0	Unrestricted	$\alpha \rightarrow \beta$
1	Context Sensitive	$\alpha \rightarrow \beta$, $\lvert\alpha\rvert \leq \lvert\beta\rvert$
2	Context Free	$A \rightarrow \beta$
3	Regular	$A \rightarrow aB$ $A \rightarrow a$

文法和语言的分类

- 四类文法描述语法的能力从0型文法开始依次减弱
  - k型语言类L_k必然是k-1型语言类L_{k-1}的子类 (其中k=1,2,3)



自顶向下的语法分析

自顶向下的语法分析

❖ 基本思想

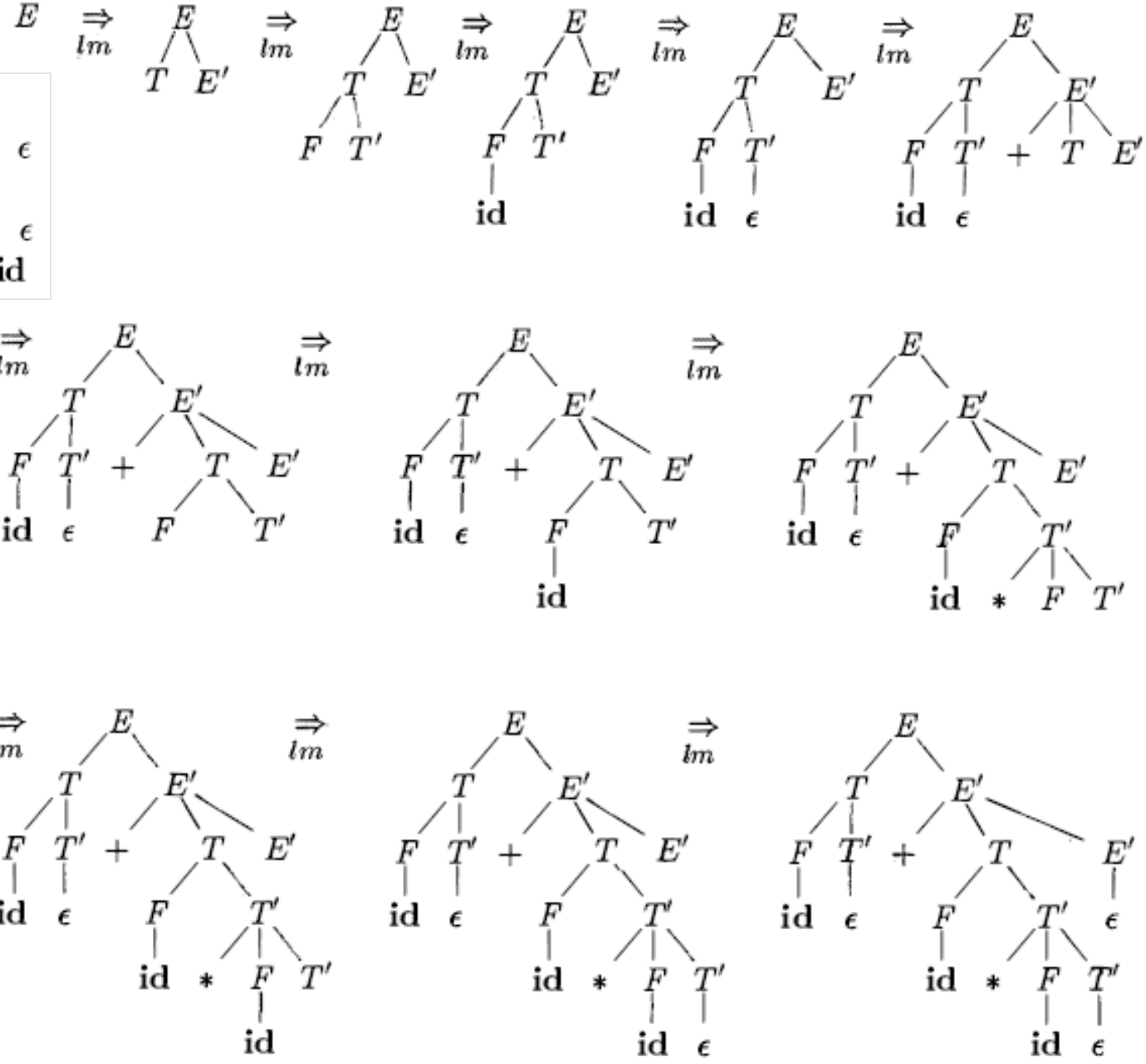
- 自顶向下语法分析可以看作是为输入串构造语法分析树的问题，从语法分析树的根结点开始，按照先根次序（深度优先）创建这棵语法分析树的各个结点
- 自顶向下语法分析也可以被看作是寻找输入串的最左推导的过程
- 关键问题是每步都要确定对一个非终结符A应用哪个产生式

自顶向下的语法分析

- 两种技术（LL(1)分析法的两种实现方式）
  - 递归下降语法分析
    - 对于一个文法G，对其每一个非终结符U构造一个递归子程序，一般的，以非终结符的名字来命名这个子程序。所有子程序构造完成后，对指定文法，运行文法开始符号对应的子程序，返回匹配结果。
  - 预测分析技术
    - 预测分析法是LL(1)分析法的另一种实现方法，它不需要构造每一个子程序，而是通过一张表来关联非终结符和终结符，这张表就是预测分析表，预测分析表可以说是预测分析法的核心部分。

id+id*id

E	$\rightarrow$	$T\ E'$	
E'	$\rightarrow$	$+\ T\ E'$	$\mid\ \epsilon$
T	$\rightarrow$	$F\ T'$	
T'	$\rightarrow$	$*\ F\ T'$	$\mid\ \epsilon$
F	$\rightarrow$	$(\ E\ )$	$\mid$ **id**



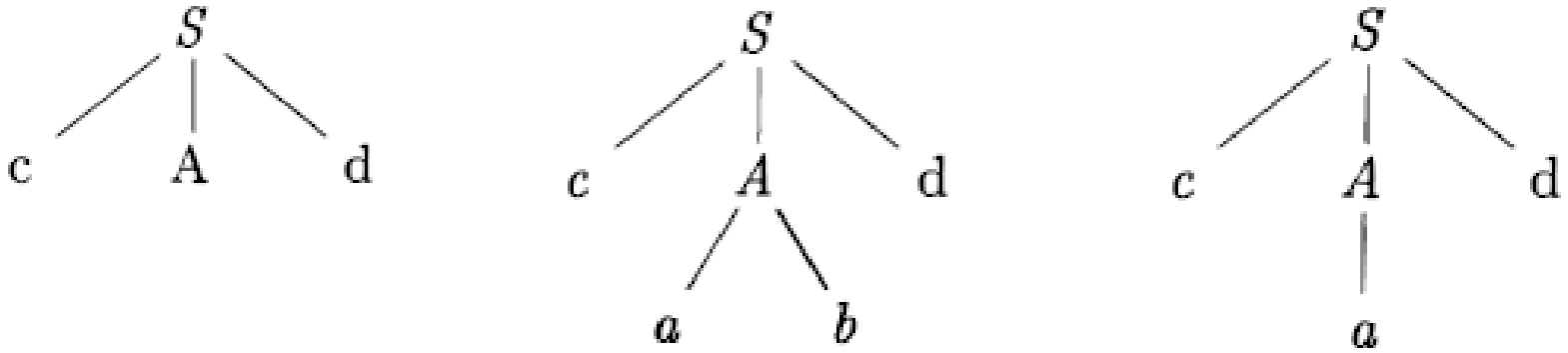
递归下降语法分析的问题

回溯

- 通用的递归下降分析技术可能需要回溯，即可能需要重复扫描输入

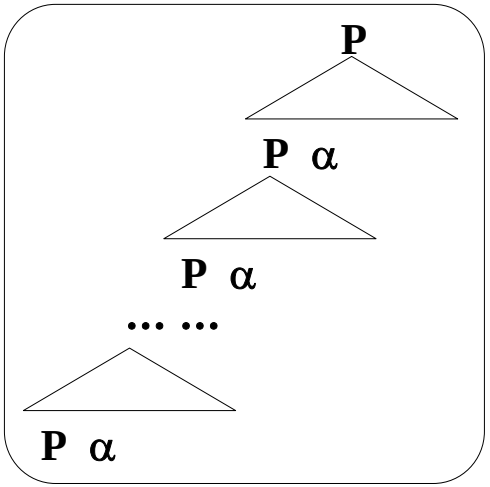
$$S \rightarrow c\ A\ d$$
$$A \rightarrow a\ b \mid a$$

the input string $w = cad$



左递归的消除

- 左递归文法
  - 如果一个文法中有一个非终结符号P使得对某个串α存在一个推导$P \Rightarrow^{+} P\alpha$，那么这个文法就是左递归的
  - 自顶向下语法分析方法不能处理左递归文法，要消除左递归



- 立即（直接）左递归的消除

将 A 的全部产生式分组如下：

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

其中 β_i 都不以 A 开头。然后，将这些 A 产生式替换为：

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

非终结符号 A 生成的串和替换之前生成的串一样，但不再是左递归的。

一个消除直接左递归的例子

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

左递归的消除

- 系统消除左递归的算法
  - 文法中不存在环（$A \Rightarrow^{+} A$）或ε产生式（$A \rightarrow \varepsilon$）

1) arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$.
2) **for** (each i from 1 to n) {
3) **for** (each j from 1 to $i-1$) {
4) replace each production of the form $A_i \rightarrow A_j\gamma$ by the productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all current A_j-productions
5) }
6) eliminate the immediate left recursion among the A_i-productions
7) }

左递归的消除……算法思想

1. 按某种顺序将 $P_i \in V_N$ 排列为 $P_1, P_2, \ldots, P_n$
2. 观察 $P_1, P_2, \ldots, P_n$ 的形式
   a) 若 $P_i \to P_j\gamma$, 且 $i < j$, 则不可能有左递归

      如， $P_1 \to P_2\alpha$， $P_2 \to P_3\beta$， $P_3 \to \delta$

   b) 若 $i \geq j$, 则可能存在左递归

      如直接： $P_1 \to P_1\alpha$， $P_2 \to P_2\beta$

      间接： $P_1 \to P_2\alpha$， $P_2 \to P_3\beta$， $P_3 \to P_1\gamma \mid P_2\delta$
3. 对间接左递归，设法改造为 a) 的形式

   如，将 P_1 的右部候选带入 P_3 的右部候选，得 $P_3 \to P_2\alpha\gamma \mid P_2\delta$

   再将 P_2 的右部候选带入上式，得 $P_3 \to P_3\beta\alpha\gamma \mid P_3\beta\delta$
4. 消除所产生的直接左递归
5. 消除所产生的无用产生式

例……左递归的消除

❖ 消除下面文法中的左递归

$$S \to A\ a \mid b$$
$$A \to A\ c \mid S\ d \mid \epsilon$$

We order the nonterminals S, A.

$S \to A\ a \mid b$ ⇒ $S \to A\ a \mid b$

$A \to A\ c \mid S\ d \mid \epsilon$ ⇒ $A \to A\ c \mid A\ a\ d \mid b\ d \mid \epsilon$

$A \to b\ d\ A' \mid A'$

$A' \to c\ A' \mid a\ d\ A' \mid \epsilon$

⇐

$S \to A\ a \mid b$

$A \to b\ d\ A' \mid A'$

$A' \to c\ A' \mid a\ d\ A' \mid \epsilon$

例……左递归的消除

- 例 文法 $S \to Qc \mid c \quad Q \to Rb \mid b \quad R \to Sa \mid a$
  - 设非终结符的排序为 R, Q, S, 将产生式重排为

    $R \to Sa \mid a \quad Q \to Rb \mid b \quad S \to Qc \mid c$ (Q, S需改造)

    将R的候选代入Q： $Q \to Sab \mid ab \mid b$ (1)

    将(1)式代入S： $S \to Sabc \mid abc \mid bc \mid c$ (2)

    消除(2)式的直接左递归，得

    $S \to abcS' \mid bcS' \mid cS'$

    $S' \to abcS' \mid \varepsilon$

    $Q \to Sab \mid ab \mid b$

    $R \to Sa \mid a$

    因为S为开始符号，所以R, Q成为无用产生式，故消除

  - 化简后，有 $S \to abcS' \mid bcS' \mid cS'$， $S' \to abcS' \mid \varepsilon$

例……左递归的消除（课堂练习）

- 若设非终结符的排序为 S, Q, R， 有

  $S \to Qc \mid c \quad Q \to Rb \mid b \quad R \to Sa \mid a$

  - 将S的候选代入R，有 $R \to Qca \mid ca \mid a$ (1)
  - 将Q的候选带入(1), 有 $R \to Rbca \mid bca \mid ca \mid a$ (2)
  - 消除(2)的直接左递归

    $R \to bcaR' \mid caR' \mid aR'$， $R' \to bcaR' \mid \varepsilon$
  - $S \to Qc \mid c$

    $Q \to Rb \mid b$

    $R \to bcaR' \mid caR' \mid aR'$,

    $R' \to bcaR' \mid \varepsilon$
- 不同的排序可导致不同的文法形式，容易证明得到的两个文法等价

提取左公因子

❖ 提取左公因子的算法

输入：文法 G。

输出：一个等价的提取了左公因子的文法。

方法：对于每个非终结符号 A，找出它的两个或多个选项之间的最长公共前缀 α。

如果$\alpha \neq \epsilon$，即存在一个非平凡的公共前缀，那么将所有 A 产生式

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n \mid \gamma$，替换为

$$A \to \alpha A' \mid \gamma$$

$$A' \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

其中，γ 表示所有不以 α 开头的产生式体；A'是一个新的非终结符号。

不断应用这个转换，直到每个非终结符号的任意两个产生式体都没有公共前缀为止。

例……提取左公因子

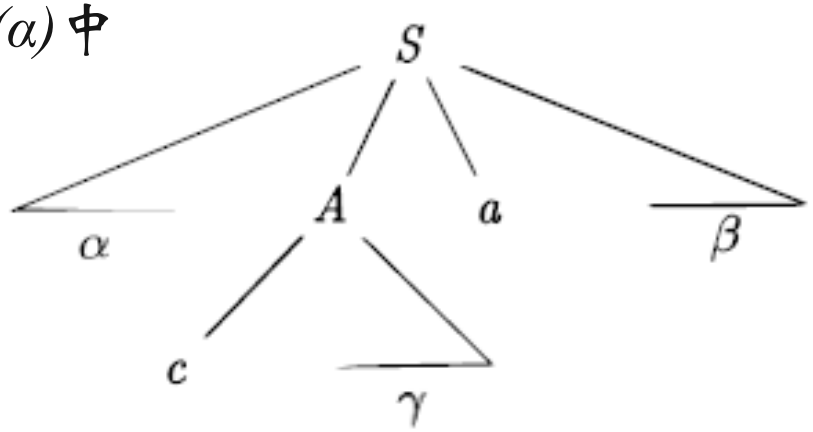
- 例4.11

$$S \rightarrow i\ E\ t\ S \mid i\ E\ t\ S\ e\ S \mid a$$
$$E \rightarrow b$$

$$S \rightarrow i\ E\ t\ S\ S' \mid a$$
$$S' \rightarrow e\ S \mid \epsilon$$
$$E \rightarrow b$$

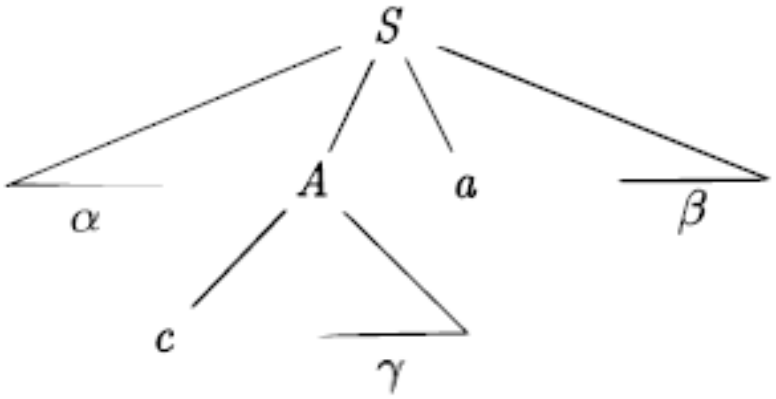
FIRST和FOLLOW

- *FIRST*和*FOLLOW*是和文法*G*相关的两个函数
  - 自顶向下分析过程中，用这两个函数可以根据下一个输入符号选择应该哪个产生式
  - 恐慌模式错误分析中，由*FOLLOW*产生的词法单元集合可以作为同步词法单元
- *FIRST*(α)的定义
  - 可以从任意文法符号串α推导得到的串的首终结符号集合
  - 如果α⇒*ε，那么ε也在*FIRST*(α)中
    - 例：*c*在*FIRST*(*A*)中



FIRST和FOLLOW

- FIRST在分析中的使用
  - 考虑A有两个产生式$A\rightarrow\alpha|\beta$，FIRST(α)和FIRST(β)不相交；那么只需要查看下一个输入符号a，就可以在这两个A产生式中进行选择：如果a出现在FIRST(α)中，则选择产生式$A\rightarrow\alpha$；如果a出现在FIRST(β)中，则选择产生式$A\rightarrow\beta$；否则a在此出现是语法错误
- FOLLOW(A)的定义
  - 对非终结符A，FOLLOW(A)被定义为可能在某个句型中紧跟在A右边的终结符号的集合
  - 如果A是某些句型最右部的符号，那么$也在FOLLOW(A)中
    - 例：a在FOLLOW(A)中



计算FIRST集合

❖ 计算文法符号X的$FIRST(X)$

计算各个文法符号 X 的 FIRST(X)时，不断应用下列规则，直到再没有新的终结符号或 ϵ 可以被加入到任何 FIRST 集合中为止。

1）如果 X 是一个终结符号，那么 FIRST(X) = X。

2）如果 X 是一个非终结符号，且 $X \to Y_1 Y_2 \cdots Y_k$ 是一个产生式，其中 $k \geqslant 1$，那么如果对于某个 i，a 在 FIRST(Y_i)中且 ϵ 在所有的 FIRST(Y_1)、FIRST(Y_2)、…、FIRST(Y_{i-1})中，就把 a 加入到 FIRST(X)中。也就是说，$Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$。如果对于所有的 $j=1, 2, \cdots, k$，ϵ 在 FIRST(Y_j)中，那么将 ϵ 加入到 FIRST(X)中。比如，FIRST(Y_1)中的所有符号一定在 FIRST(X)中。如果 Y_1 不能推导出 ϵ，那么我们就不会再向 FIRST(X)中加入任何符号，但是如果 $Y_1 \stackrel{*}{\Rightarrow} \epsilon$，那么我们就加上 FIRST($Y_2$)，依此类推。

3）如果 $X \to \epsilon$ 是一个产生式，那么将 ϵ 加入到 FIRST(X)中。

计算FIRST集合

- 对任意串$\alpha = X_1X_2...X_n$，计算$FIRST(\alpha)$

我们可以按照如下方式计算任何串 $X_1X_2\cdots X_n$ 的 FIRST 集合。

向 FIRST($X_1X_2\cdots X_n$) 加入 $F(X_1)$ 中所有的非 ϵ 符号。

如果 ϵ 在 FIRST(X_1) 中，再加入 FIRST(X_2) 中的所有非 ϵ 符号；

如果 ϵ 在 FIRST(X_1) 和 FIRST(X_2) 中，加入 FIRST(X_3) 中的所有非 ϵ 符号，依此类推。

最后，如果对所有的 i，ϵ 都在 FIRST(X_i) 中，那么将 ϵ 加入到 FIRST($X_1X_2\cdots X_n$) 中。

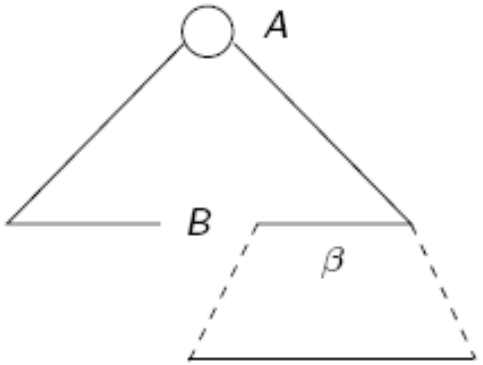
计算FOLLOW集合

计算所有非终结符号 A 的 FOLLOW(A)集合时，不断应用下面的规则，直到再没有新的终结符号可以被加入到任意 FOLLOW 集合中为止。

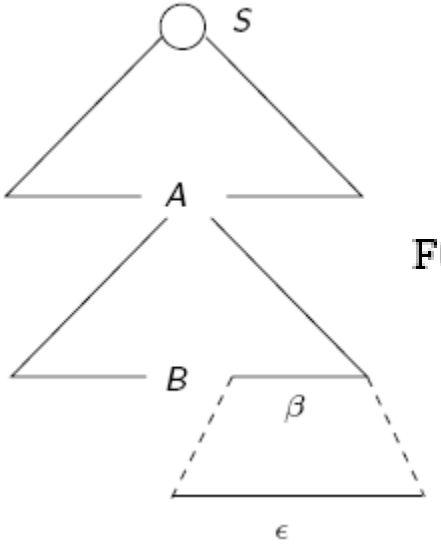
1）将 $ 放到 FOLLOW(S)中，其中 S 是开始符号，而 $ 是输入右端的结束标记。

2）如果存在一个产生式 $A \rightarrow \alpha B\beta$，那么 FIRST（$\beta$）中除 ϵ 之外的所有符号都在FOLLOW(B)中。

3）如果存在一个产生式 $A \rightarrow \alpha B$，或存在产生式 $A \rightarrow \alpha B\beta$ 且 FIRST（β）包含 ϵ，那么 FOLLOW(A)中的所有符号都在 FOLLOW(B)中。



$\text{FIRST}(\beta) \subseteq \text{FOLLOW}(B)$



$\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$

例……FIRST和FOLLOW

- 例：对 $\forall X \in V_N$，构造$FIRST(X)$和$FOLLOW(X)$

$FIRST(E) = \{(, \mathbf{id}\}$

$FIRST(E') = \{+, \varepsilon\}$

$FIRST(T) = \{(, \mathbf{id}\}$

$FIRST(T') = \{*, \varepsilon\}$

$FIRST(F) = \{(, \mathbf{id}\}$

$FIRST(TE') = \{(, \mathbf{id}\}$

$FIRST(+TE') = \{+\}$

$FIRST(FT') = \{(, \mathbf{id}\}$

$FIRST(*FT') = \{*\}$

$FIRST((E)) = \{(\}$

$FIRST(\varepsilon) = \{\varepsilon\}$

$FOLLOW(E) = \{), \$\}$

$FOLLOW(E') = \{), \$\}$

$FOLLOW(T) = \{+,), \$\}$

$FOLLOW(T') = \{+,), \$\}$

$FOLLOW(F) = \{*, +,), \$\}$

$$E \rightarrow TE'$$

$$E' \rightarrow +\ TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *\ FT' \mid \varepsilon$$

$$F \rightarrow (E) \mid \mathbf{id}$$

例……FIRST和FOLLOW

- 例：对 $\forall X \in V_N$，构造$FIRST(X)$和$FOLLOW(X)$

$S \rightarrow BC\boldsymbol{c} \mid \boldsymbol{g}DB$

$B \rightarrow \boldsymbol{b}CDE \mid \boldsymbol{\varepsilon}$

$C \rightarrow D\boldsymbol{a}B \mid \boldsymbol{ca}$

$D \rightarrow \boldsymbol{d}D \mid \boldsymbol{\varepsilon}$

$E \rightarrow \boldsymbol{g}D\boldsymbol{f} \mid \boldsymbol{c}$

FIRST(S) = {b, d, a, c, g}

FIRST(B) = {b, ε}

FIRST(C) = {d, a, c}

FIRST(D) = {d, ε}

FIRST(E) = {g, c}

FOLLOW(S) = {$}

FOLLOW(B) = {$, d, a, c, g}

FOLLOW(C) = {c, d, g}

FOLLOW(D) = {b, $, g, c, a, f}

FOLLOW(E) = {$, d, a, c, g}

LL(1)文法

- *LL(1)*文法的特点
  - 可以构造预测分析器，即不需要回溯的递归下降语法分析器
  - 足以描述大部分程序设计语言构造
  - 左递归的文法和二义性文法都不可能是LL(1)的
- *LL(1)*文法的定义

一个文法 G 是 LL(1) 的，当且仅当 G 的任意两个不同的产生式 $A \to \alpha \mid \beta$ 满足下面的条件：

1）不存在终结符号 a 使得 α 和 β 都能够推导出以 a 开头的串。

2）α 和 β 中最多只有一个可以推导出空串。

3）如果 $\beta \stackrel{*}{\Rightarrow} \epsilon$，那么 α 不能推导出任何以 FOLLOW(A) 中某个终结符号开头的串。类似地，如果 $\alpha \stackrel{*}{\Rightarrow} \epsilon$，那么 β 不能推导出任何以 FOLLOW(A) 中某个终结符号开头的串。

LL(1)文法的定义和判定

- 若一个文法G满足以下条件，则称文法G为$LL(1)$文法
  1. G不含左递归
  2. 若$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, 则 $FIRST(\alpha_i) \cap FIRST(\alpha_j) = \varnothing, i \neq j$
  3. $\forall A \in V_N$, 若 $\varepsilon \in FIRST(\alpha_i)$，则 $FIRST(A) \cap FOLLOW(A) = \varnothing$
- 对文法G能构造出不含多重定义项的预测分析表，当且仅当G是$LL(1)$的
  - G不含左递归且没有左公因子是G是$LL(1)$文法的必要条件

练习4.3

练习 4.3.1：下面是一个只包含符号 a 和 b 的正则表达式的文法。它使用 + 替代表示并运算 的字符 | ，以避免和文法中作为元符号使用的竖线相混淆：

$$rexpr \rightarrow rexpr \ + \ rterm \mid rterm$$

$$rterm \rightarrow rterm \ rfactor \mid rfactor$$

$$rfactor \rightarrow rfactor \ * \mid rprimary$$

$$rprimary \rightarrow \mathbf{a} \mid \mathbf{b}$$

1）对这个文法提取左公因子。

2）提取左公因子的变换能使这个文法适用于自顶向下的语法分析技术吗？

3）提取左公因子之后，从原文法中消除左递归。

4）得到的文法适用于自顶向下的语法分析吗？

递归下降的语法分析

- 递归下降语法分析程序
  - 由一组过程组成，每个非终结符号有一个对应的过程
  - 程序从开始符号对应的过程开始执行，如果该过程的过程体扫描了整个输入串，它就停止执行并宣布语法分析成功完成
  - 非终结符A对应过程的典型伪代码

```
    void A() {
1)      Choose an A-production, A → X1X2 ··· Xk;
2)      for ( i = 1 to k ) {
3)          if ( Xi is a nonterminal )
4)              call procedure Xi();
5)          else if ( Xi equals the current input symbol a )
6)              advance the input to the next symbol;
7)          else /* an error has occurred */;
        }
    }
```

例……递归下降分析程序

```
PROCEDURE S ();
    IF curr_tok = if THEN
       match(if); E();
       match(then); S();
    ELSIF curr_tok = id THEN
       match(id); match(:=); E();
    ELSE syntax error ENDIF;
PROCEDURE E ();
    IF curr_tok = id THEN match(id);
    IF curr_tok = num THEN match(num);
    ELSE E(); match(+); E();
    ENDIF;
```

$$
\begin{array}{rcl}
S & \rightarrow & \underline{\texttt{id}}\ \underline{:=}\ E \\
 & | & \underline{\texttt{if}}\ E\ \underline{\texttt{then}}\ S \\
E & \rightarrow & E \underline{+} E \\
 & | & \underline{\texttt{id}} \mid \underline{\texttt{num}}
\end{array}
$$

递归下降语法分析的问题

左递归和无限循环

- 左递归的文法会使递归下降语法分析器进入无限循环，也就是说，当试图展开一个非终结符号A时，可能会没有读入任何输入符号就再次试图展开A

$$E \rightarrow E\underline{+}E \mid \underline{\text{id}} \mid \underline{\text{num}}$$

```
PROCEDURE E ();
    IF curr_tok = id THEN match(id);
    IF curr_tok = num THEN match(num);
    ELSE E(); match(+); E();
    ENDIF;
```

递归下降语法分析的问题

- 公共左因子导致的回溯
- 一个非终结符号的First集合中含空字ε，而First集合和Follow集合有交集

预测分析表

- 预测分析表M是一个二维数组
  - $M[A,a]$定义了用非终结符号A去匹配输入符号a时应该选择的A产生式
  - 其中A是非终结符，a是终结符号或输入结束标记$\$$

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

预测分析表的构造

预测分析表的构造思想

- 只有当下一个输入符号a在$FIRST(\alpha)$中时才选择产生式$A\rightarrow\alpha$
- 当$\alpha=\varepsilon$时，或更一般的$\alpha\Rightarrow^*\varepsilon$时，也就是$\varepsilon\in FIRST(\alpha)$，如果当前输入符号$a$在$FOLLOW(A)$中，仍选择$A\rightarrow\alpha$

构造算法

输入：文法 G。

输出：预测分析表 M。

方法：对于文法 G 的每个产生式 $A\rightarrow\alpha$，进行如下处理：

1）对于 FIRST(α)中的每个终结符号 a，将 $A\rightarrow\alpha$ 加入到$M[A, a]$中。

2）如果 ϵ 在 FIRST(α)中，那么对于 FOLLOW(A)中的每个终结符号 b，将 $A\rightarrow\alpha$ 加入到 $M[A, b]$中。如果 ϵ 在 FIRST(α)中，且 \$ 在 FOLLOW(A)中，也将 $A\rightarrow\alpha$ 加入到 $M[A, \$]$中。

在完成上面的操作之后，如果 $M[A, a]$中没有产生式，那么将 $M[A, a]$设置为 **error**（我们通常在表中用一个空条目表示）。

例……预测分析表的构造

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow \underline{+}\ T\ E' \mid \epsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow \underline{*}\ F\ T' \mid \epsilon \\
F &\rightarrow \underline{(}\ E\ \underline{)} \mid \underline{\mathtt{id}}
\end{aligned}
$$

FIRST(E)	=	$\{(, \mathtt{id}\}$	FOLLOW(E)	=	$\{), \$\}$
FIRST(E')	=	$\{+, \epsilon\}$	FOLLOW(E')	=	$\{), \$\}$
FIRST(T)	=	$\{(, \mathtt{id}\}$	FOLLOW(T)	=	$\{+,), \$\}$
FIRST(T')	=	$\{*, \epsilon\}$	FOLLOW(T')	=	$\{+,), \$\}$
FIRST(F)	=	$\{(, \mathtt{id}\}$	FOLLOW(F)	=	$\{+, *,), \$\}$

$E \rightarrow TE'$ $\quad \text{FIRST}(TE') = \text{FIRST}(T) = \{\underline{(}, \underline{\text{id}}\}$

	$\underline{\text{id}}$	$\underline{+}$	$\underline{*}$	$\underline{(}$	$\underline{)}$	$\$$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'						
T						
T'						
F						

$E' \rightarrow \underline{+}TE'$ $\quad \text{FIRST}(\underline{+}TE') = \{\underline{+}\}$

	$\underline{\text{id}}$	$\underline{+}$	$\underline{*}$	$\underline{(}$	$\underline{)}$	$\$$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow \underline{+}TE'$				
T						
T'						
F						

$E' \rightarrow \epsilon$ $\quad$ $\texttt{FOLLOW}(E') = \{), \$\}$

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T						
T'						
F						

$T \rightarrow FT'$ $\quad$ $\texttt{FIRST}(FT') = \{(, \texttt{id}\}$

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'						
F						

$T' \rightarrow *FT'$. $\quad \text{FIRST}(*FT') = \{\underline{*}\}$

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow \underline{+}TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'			$T' \rightarrow *FT'$			
F						

$T' \rightarrow \epsilon \quad \text{FOLLOW}(T') = \{\underline{+}, \underline{)}, \$\}$

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow \underline{+}TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F						

$F \to \underline{(}E\underline{)}$ $\quad\quad$ $\texttt{FIRST}(\underline{(}E\underline{)}) = \{\underline{(}\}$

$F \to \underline{\texttt{id}}$ $\quad\quad$ $\texttt{FIRST}(\underline{\texttt{id}}) = \{\underline{\texttt{id}}\}$

	$\underline{\texttt{id}}$	$\underline{+}$	$\underline{*}$	$\underline{(}$	$\underline{)}$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to \underline{+}TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \underline{\texttt{id}}$			$F \to \underline{(}E\underline{)}$		

预测分析表和LL(1)文法

- 预测分析表的构造算法可以应用于任何文法G
  - 对于*LL(1)*文法，分析表中的每个条目都唯一指定了一个产生式，或者标明一个语法错误
- 对文法*G*能构造出不含多重定义项的预测分析表，当且仅当*G*是*LL(1)*的
  - *G*不含左递归且没有左公因子是*G*是*LL(1)*文法的必要条件
- *LL(1)*文法改写的题型
  1. 消除左递归，提取左公因子
  2. 证明得到的文法是*LL(1)*的

例……构造预测分析表

$S \rightarrow BCc \mid gDB$

$B \rightarrow bCDE \mid \varepsilon$

$C \rightarrow DaB \mid ca$

$D \rightarrow dD \mid \varepsilon$

$E \rightarrow gDf \mid c$

FIRST(S) = {b, d, a, c, g} FOLLOW(S) = {$}

FIRST(B) = {b, ε} FOLLOW(B) = {$, d, a, c, g}

FIRST(C) = {d, a, c} FOLLOW(C) = {c, d, g}

FIRST(D) = {d, ε} FOLLOW(D) = {b, $, g, c, a, f}

FIRST(E) = {g, c} FOLLOW(E) = {$, d, a, c, g}

	a	b	c	d	f	g	$
S	$S \rightarrow BCc$	$S \rightarrow BCc$	$S \rightarrow BCc$	$S \rightarrow BCc$		$S \rightarrow gDB$	
B	$B \rightarrow \varepsilon$	$B \rightarrow bCDE$	$B \rightarrow \varepsilon$	$B \rightarrow \varepsilon$		$B \rightarrow \varepsilon$	$B \rightarrow \varepsilon$
C	$C \rightarrow DaB$		$C \rightarrow ca$	$C \rightarrow DaB$			
D	$D \rightarrow \varepsilon$	$D \rightarrow \varepsilon$	$D \rightarrow \varepsilon$	$D \rightarrow dD$	$D \rightarrow \varepsilon$	$D \rightarrow \varepsilon$	$D \rightarrow \varepsilon$
E			$E \rightarrow c$			$E \rightarrow gDf$	

非LL(1)文法的例子

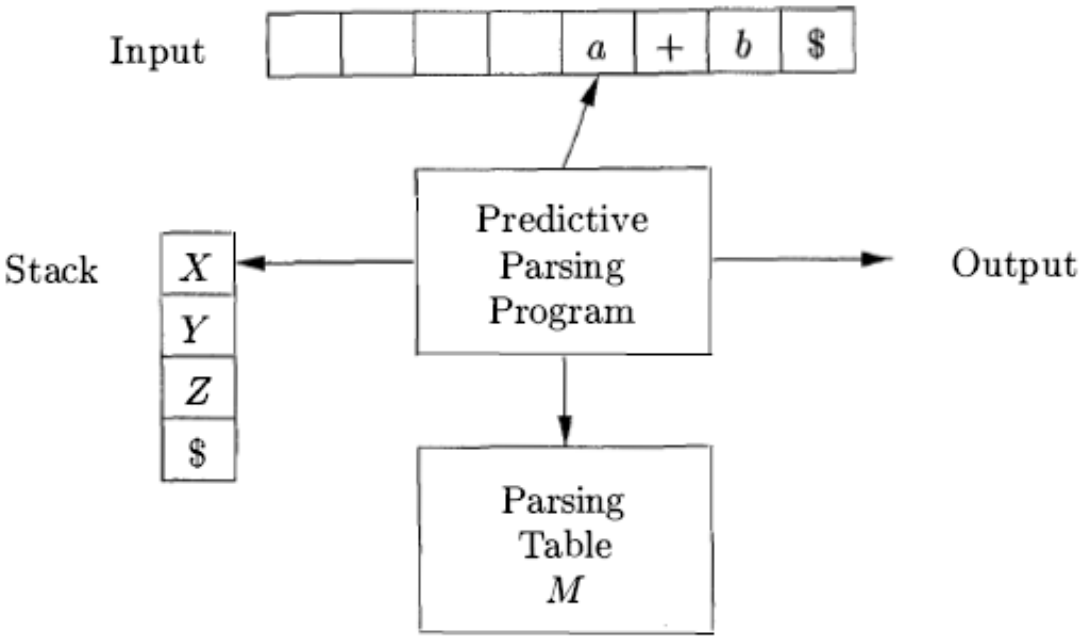
- 二义性文法不是LL(1)文法
  - 也存在很多无二义性的文法不是LL(1)的

$$
\begin{aligned}
S &\rightarrow iEtSS' \mid a \\
S' &\rightarrow eS \mid \epsilon \\
E &\rightarrow b
\end{aligned}
$$

NON - TERMINAL	INPUT SYMBOL					
	a	b	e	i	t	$\$$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow \epsilon$ $S' \rightarrow eS$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				

非递归的预测分析

- 表驱动的预测分析器模型
  - 栈存放文法符号，用$表示栈底，开始时，先将S推入栈顶
  - 输入缓冲区用$表示输入串结束(句末符)
  - $\forall A \in V_N, a \in V_T \cup \{\$\}$ 预则分析表 $M[A, a]=$ { $A \to \alpha$ (α推进栈)；出错标志



预测分析器

预测分析器总控程序的动作

- 设栈顶符号为X，当前输入符号为a，执行下列动作之一

1. 若$X = a = \$$，则宣布分析成功，停止分析过程
2. 若$X = a \neq \$$，则X退栈，让输入符号a指向下一个符号

   否则，若$X \in V_T$，则调用错误诊断*ERROR()*
3. 若$X \in V_N$，则查看分析表M

   $M[X, a] = X \rightarrow x_1x_2...x_k$，弹出$X$，将$x_k, x_{k-1}, ..., x_1$推入栈

   $M[X, a] = X \rightarrow \varepsilon$，仅弹出X，并不推入任何符号

   $M[X, a]$ = 出错标志，调用错误诊断*ERROR()*

表驱动的预测语法分析算法(总控程序)

输入：一个串 w，文法 G 的预测分析表 M。

输出：如果 w 在 $L(G)$ 中，输出 w 的一个最左推导；否则给出一个错误指示。

方法：最初，语法分析器的格局如下：输入缓冲区中是 $w\$$，而 G 的开始符号 S 位于栈顶，它的下面是 $\$$。预测分析算法使用预测分析表 M 生成了处理这个输入的预测分析过程。

```
set ip to point to the first symbol of w;
set X to the top stack symbol;
while ( X ≠ $ ) { /* stack is not empty */
        if ( X is a ) pop the stack and advance ip;
        else if ( X is a terminal ) error();
        else if ( M[X, a] is an error entry ) error();
        else if ( M[X, a] = X → Y1 Y2 ··· Yk ) {
                output the production X → Y1 Y2 ··· Yk;
                pop the stack;
                push Yk, Yk−1, ..., Y1 onto the stack, with Y1 on top;
        }
        set X to the top stack symbol;
}
```

例……表驱动的预测分析过程

MATCHED	STACK	INPUT	ACTION
	$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
	$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $E \to TE'$
	$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id}$	$T'E'\$$	$+\,\mathbf{id} * \mathbf{id}\$$	match **id**
$\mathbf{id}$	$E'\$$	$+\,\mathbf{id} * \mathbf{id}\$$	output $T' \to \epsilon$
$\mathbf{id}$	$+\,TE'\$$	$+\,\mathbf{id} * \mathbf{id}\$$	output $E' \to +\,TE'$
$\mathbf{id} +$	$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	match $+$
$\mathbf{id} +$	$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
$\mathbf{id} +$	$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$T'E'\$$	$* \mathbf{id}\$$	match **id**
$\mathbf{id} + \mathbf{id}$	$* FT'E'\$$	$* \mathbf{id}\$$	output $T' \to * FT'$
$\mathbf{id} + \mathbf{id} *$	$FT'E'\$$	$\mathbf{id}\$$	match $*$
$\mathbf{id} + \mathbf{id} *$	$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$T'E'\$$	$\$$	match **id**
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$E'\$$	$\$$	output $T' \to \epsilon$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$\$$	$\$$	output $E' \to \epsilon$

例……预测分析表的构造

- 已知文法G(S)

  S →S*aP | aP | *aP

  P → +aP | +a

  1. 将文法改写为*LL(1)*文法*G'(S)*
  2. 构造文法*G'*的预测分析表

- 解答

  （1）消除左递归

  （2）提取左公因子

  （3）计算First和Follow集合

  （4）构造预测分析表

例……解答

解：（1）消除文法G中的左递归

$S \rightarrow aPS' \mid *aPS'$

$S' \rightarrow *aPS' \mid \varepsilon$

$P \rightarrow +aP \mid +a$

（2）提取公共左因子，得到G'

$S \rightarrow aPS' \mid *aPS'$

$S' \rightarrow *aPS' \mid \varepsilon$

$P \rightarrow +aP'$

$P' \rightarrow P \mid \varepsilon$

（3）计算每个非终结符的First和Follow集合

FIRST(S)={a,*}　FIRST(S')={*, ε}

FIRST(P)={+}　FIRST(P')={+, ε}

FOLLOW(S) = {$}　FOLLOW(S') = {$}

FOLLOW(P) = FOLLOW(P') = {*,$}

（4）构造G'的预测分析表如下

	a	*	+	$
S	$S \rightarrow aPS'$	$S \rightarrow *aPS'$		
S'		$S' \rightarrow *aPS'$		$S' \rightarrow \varepsilon$
P			$P \rightarrow +aP'$	
P'		$P' \rightarrow \varepsilon$	$P' \rightarrow P$	$P' \rightarrow \varepsilon$

预测分析表中没有多重定义项，所以G'是LL(1)的

LL(1)……补充练习（5、6、7课堂自主练习）

- 文法消除了左递归提取了左公因子就是LL(1)文法吗？改写判定

① $A \rightarrow baB \mid \varepsilon \quad B \rightarrow Abb \mid a$

② $A \rightarrow aABe \mid a \quad B \rightarrow Bb \mid d$

③ $S \rightarrow Aa \mid b \quad A \rightarrow SB \quad B \rightarrow ab$

④ $S \rightarrow AS \mid b \quad A \rightarrow SA \mid a$

⑤ $S \rightarrow Ab \mid Ba \quad A \rightarrow aA \mid a \quad B \rightarrow a$

⑥ $S \rightarrow ab \mid ba \mid \varepsilon$

⑦ $S \rightarrow Cd \quad C \rightarrow aB \mid bA \quad A \rightarrow a \mid aC \mid bAA \quad B \rightarrow b \mid bC \mid aBB$

补充练习……解答

- 文法消除了左递归提取了左公因子就是LL(1)文法吗？改写判定

① $A \rightarrow baB \mid \varepsilon \quad B \rightarrow Abb \mid a$ （本不是）

② $A \rightarrow aABe \mid a \quad B \rightarrow Bb \mid d$ （改写后是）

③ $S \rightarrow Aa \mid b \quad A \rightarrow SB \quad B \rightarrow ab$ （改写仍不是）

④ $S \rightarrow AS \mid b \quad A \rightarrow SA \mid a$ （改写仍不是）

⑤ $S \rightarrow Ab \mid Ba \quad A \rightarrow aA \mid a \quad B \rightarrow a$ （改写仍不是）

⑥ $S \rightarrow ab \mid ba \mid \varepsilon$ （是）

⑦ $S \rightarrow Cd \quad C \rightarrow aB \mid bA \quad A \rightarrow a \mid aC \mid bAA \quad B \rightarrow b \mid bC \mid aBB$ （改写仍不是）

练习4.4（5、6、7）是作业

- 文法消除了左递归提取了左公因子就是LL(1)文法吗？改写判定。

1）$S \rightarrow 0\ S\ 1 \mid 0\ 1$ 和串 000111。

2）$S \rightarrow +\ S\ S \mid *\ S\ S \mid a$ 和串 $+\ *\ aaa$。

! 3）$S \rightarrow S\ (\ S\ )\ S \mid \epsilon$ 和串(()())。

! 4）$S \rightarrow S\ +\ S \mid S\ S \mid (\ S\ ) \mid S\ * \mid a$ 和串 $(a+a)*a$。

! 5）$S \rightarrow (\ L\ ) \mid a$ 以及 $L \rightarrow L, S \mid S$ 和串((a, a), a, (a))。

!! 6）$S \rightarrow a\ S\ b\ S \mid b\ S\ a\ S \mid \epsilon$ 和串 $aabbab$。

! 7）下面的布尔表达式对应的文法：

$bexpr \rightarrow bexpr$ **or** $bterm \mid bterm$

$bterm \rightarrow bterm$ **and** $bfactor \mid bfactor$

$bfactor \rightarrow$ **not** $bfactor \mid (\ bexpr\ ) \mid$ **true** $\mid$ **false**

消除二义性

- 构造无二义的表达式文法

1. 为每个优先级创建一个非终结符，如$P_1, P_2, \ldots, P_n$，其中P_n的优先级最高
2. 对优先级为i的运算符op如下构造产生式
   - 如果op是左结合的，则产生式为 $P_i \rightarrow P_i\ op\ P_{i+1} \mid P_{i+1}$
   - 如果op是右结合的，则产生式为 $P_i \rightarrow P_{i+1}\ op\ P_i \mid P_{i+1}$
3. 对于初等表达式如数字、标识符、括号等，创建非终结符P_{n+1}，构造产生式为

$$P_{n+1} \rightarrow num \mid id \mid (P_1)$$