编译原理-期中考试

CH0 概论

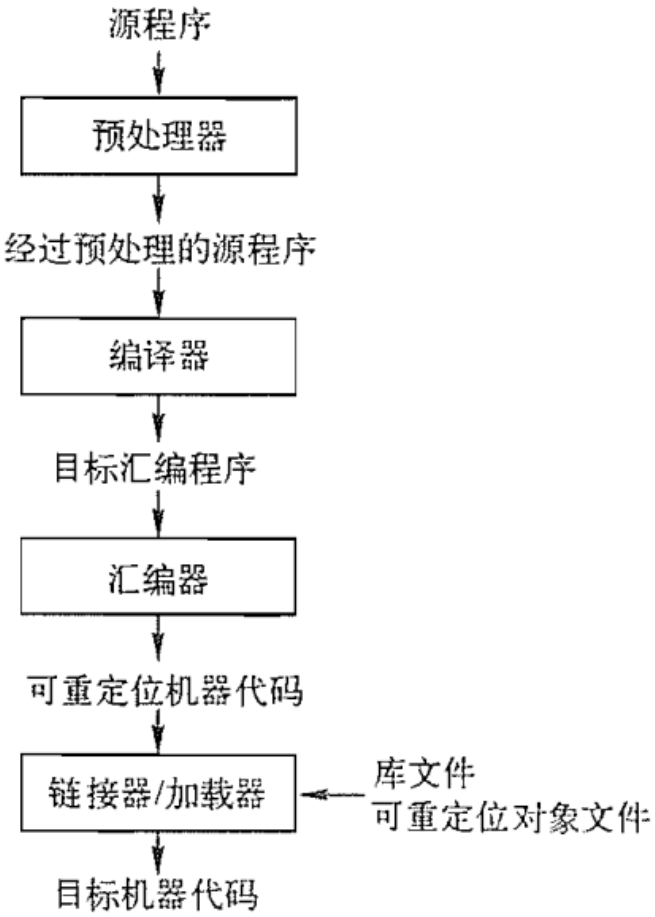
- 内容概览
  - 形式化语言与自动机
  - 类型论和类型系统
  - 程序分析原理
- 编译器的功能？
  - 翻译
  - 优化
  - 分析（稳定性，安全性）
- 编译器的种类
  - 交叉编译器
  - 增量编译器
  - 预先编译器
  - 即时编译器

CH1 基本概念

编译过程

- 语言处理器
  - 编译器
    - 将源语言编写的程序翻译成一个等价的、目标语言编写的程序
    - 报告翻译过程中源程序的错误
  - 解释器
    - 根据用户输入逐个语句执行源程序

    - 比编译器慢，但是错误诊断更好



- 编译过程
    - 分析——前端
        - 接受源程序，生成符号表和中间表示
        - 词法分析
        - 语法分析
        - 语义分析
        - 中间代码生成
            - 生成一个明确的、能够被轻松翻译为目标机器上的语言的、低级/类机器语言的代码
            - 线性表示形式：三地址代码

        - 静态检查：语法检查及类型检查
        - 机器无关的代码优化（程度不定）
    - 综合——后端
        - 接受符号表和中间表示，生成目标语言汇编程序
- 符号表
    - 记录源程序中使用的所有变量的名称、属性（类型、作用域）
    - 每个程序块/作用域设置单独的符号表
- 编译器的构造工具
    - 词法分析器的生成器
    - 语法分析器的生成器
    - 语法制导的翻译引擎
    - 代码生成器的生成器
    - 数据流分析引擎

程序设计语言

静态与动态

静态策略：程序设计语言支持编译器在编译时刻静态决定某个问题

动态策略：程序设计语言只允许在运行程序 的时候决定某个问题

- 作用域策略
    - 静态作用域：基于程序结构，在编译时刻静态决定
    - 动态作用域：依赖只允许在程序运行时才能知道的因素

```
#define a (x+1)
int x = 2;
void b() { int x = 1; printf("%d\n", a); }
void c() { printf("%d\n", a); }
void main() { b(); c(); }
```

图 1-12　一个其名字的作用域必须动态确定的宏

错误类别

- 词法错误：拼写错误/字符串文本引号（影响词素识别的错误）
- 语法错误：分号/花括号多余或缺失 （影响语法分析的错误）
- 语义错误：运算符和运算分量之间类型不匹配（影响语义分析）
- 逻辑错误：程序可能是良构的，但是没有正确反映程序员的意图

文法

使用上下文无关文法描述程序设计语言的语法结构

文法类别

- 零型文法
  - 产生式左边有非终结符，右边有终结符
- 一型文法（上下文有关文法）
  - 产生式左边可以有多个字符，但必须有一个非终结符；式子右边可以有多个字符，可以是终结符，也可以是非终结符，但必须是有限个字符且左边长度必须小于右边（$\alpha \rightarrow \epsilon$例外）
  - $L_1 = \{wcw|w属于(a|b)*\}$
- 二型文法（上下文无关方法）

- 式子左边必须是非终结符，个数不限；式子右边可以有多个字符，可以是终结符，也可以是非终结符，但必须是有限个字符
- $L_2 = \{a^n b^m c^n d^m | n > 0, m > 0\}$

- 三型文法（正则表达式）
  - 右线性文法：式子左边只能有一个字符，而且必须是非终结符；式子右边最多有二个字符。如果有二个字符必须是（终结符+非终结符）的格式，如果是一个字符，那么必须是终结符
  - 左线性文法：式子左边只能有一个字符，而且必须是非终结符；式子右边最多有二个字符。如果有二个字符必须是（非终结符+终结符）的格式，如果是一个字符，那么必须是终结符
  - $L_3 = \{a^n b^n c^n | n > 0\}$

四种类型文法描述能力比较：0型>1型>2型>3型文法

wcw

$$L_1' = \{wcw^R \mid w \in (a|b)^*\}$$

$$S \to aSa \mid bSb \mid c$$

$a^n b^m c^n d^m$

$$L_2' = \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

$$S \to aSd \mid aAd$$

$$A \to bAc \mid bc$$

$a^n b^n c^n$

$$L_2'' = \{a^n b^n c^m d^m \mid n \geq 1, \ m \geq 1\}$$

$$S \to AB$$

$$A \to aAb \mid ab$$

$$B \to cBd \mid cd$$

设计一个文法：字母表$\{a, b\}$上 a 和 b 的个数相等的所有串的集合

- 二义文法：　$S \to a\ S\ b\ S \mid b\ S\ a\ S \mid \varepsilon$

  $aabbabab$　　$aabbabab$

- 二义文法：　$S \to a\ B \mid b\ A \mid \varepsilon$

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

  $B \to b\ S \mid a\ B\ B$

  $aabbabab$　$aabbabab$　　$aabbabab$

- 非二义文法：　$S \to a\ B\ S \mid b\ A\ S \mid \varepsilon$

  $A \to a \mid b\ A\ A$

  $a\ abb\ abab$　$B \to b \mid a\ B\ B$

文法的基本概念

- 上下文无关文法的定义
  - 终结符集合（即词法单元名称）
  - 非终结符集合
  - 产生式集合
    - 左部：头
    - →：“可以具有如下形式”
    - 右部：体
  - 开始符号（一个非终结符）

- **上下文无关是什么意思？**
  - 指对于文法推导的每一步 $\alpha A \beta \Rightarrow \alpha \gamma \beta$

  文法符号串γ仅依据A的产生式推导，不依赖A的上下文α和β

优点

- 文法给出了精确的、易于理解的语法说明
- 可以给语言定义出层次结构
- 可以基于文法自动产生高效的分析器
- 以文法为基础实现语言便于对语言修改

缺点

- 表达能力不足够，只能描述编程语言中的大部分语法

- 正规集都是上下文无关语言
- 推导：将非终结符替换为该非终结符的某个产生式的体
  - ⇒ :"通过一步推导出"
  - $\Rightarrow^*$:"通过0步或多步推导出"
  - $\Rightarrow^+$:"通过一步或多步推导出"
  - 两种推导
    - 最左推导$\Rightarrow_{lm}$：总是替换句型中最左的非终结符
    - 规范推导/最右推导$\Rightarrow_{rm}$：总是替换句型中最右的非终结符
- 句型
  - 由开始符号通过0步或多步推导出的串，既可以包含非终结符，也可以包含终结符，也可以是空串
  - 最左句型：由开始符号经过0步或多步最左推导得到的句型
  - 最右句型：由开始符号经过0步或多步最右推导得到的句型
- 句子
  - 不包含非终结符的句型
- 文法生成的语言

    - 句子的集合
    - 开始符号表示的终结符串集合
    - 证明文法G生成语言L的方法
        1. 证明G生成的每个串都在L中（推导步数归纳）
        2. 证明L的每个串都能由G生成（串的长度归纳）
- 上下文无关语言：上下文无关方法生成的语言
- 语法分析树/推导树
    - 每个内部结点表示一个产生式的运用，
    - 内部结点是产生式的头，其子结点从左到右组成了产生式体中的符号
    - 叶子结点既可以是非终结符，也可以是终结符，也可以是空串
    - 结果/边缘：任意时刻从左到右排列这些符号得到的句型
- 抽象语法树/语法树
    - 每个内部结点表示一个运算，其子结点表示运算的分量
    - 每个内部结点是自己创建的一个构造的运算符，其子结点是这个构造的具有语义信息的组成部分/运算分量
    - 没有对应单产生式/空串产生式的结点
- 注释语法分析树：显示各个属性的值的语法分析树
- FIRST(α)集合
    - α——任意文法符号串
    - First(α)——可以从α推导得到的串的首符号的集合
    - 计算方法
        - 若α是终结符串，则$FIRST(\alpha) = \alpha$
        - 若$\alpha = X$, 且$X \to Y_1Y_2\cdots Y_k$，则
            - $FIRST(X) \leftarrow FIRST(Y_1) \cup FISRT(X)$
            - 若$Y_1Y_2\cdots Y_i \Rightarrow^* \epsilon$，则$FIRST(X) \leftarrow FIRST(Y_{i+1}) \cup FISRT(X)$

    - 若$\alpha = X_1X_2\cdots X_k$,则
        - $FIRST(\alpha) \leftarrow FIRST(X_1) \cup FISRT(\alpha)$
        - 若$X_1X_2\cdots X_i \Rightarrow^* \epsilon$，则$FIRST(\alpha) \leftarrow FIRST(X_{i+1}) \cup FISRT(\alpha)$
- FOLLOW(A)集合
    - A——任意非终结符
    - FOLLOW(A)——可能在某些句型中紧跟在A右边的终结符号集合
    - 计算方法
        - 若A是开始符号，则$FOLLOW(A) \leftarrow \{\$\} \cup FOLLOW(A)$
        - 若$S \rightarrow \alpha A\beta$, 那么$FIRST(\beta)$中除了$\epsilon$之外的所有符号都在$FOLLOW(A)$中
        - 若$S \rightarrow \alpha A\beta$,且$\epsilon \in FIRST(\beta)$,则$FOLLOW(A) \leftarrow FOLLOW(S) \cup FOLLOW(A)$

二义性

存在一个句子是两棵/以上的语法分析树的结果；存在多个最左推导；存在多个最右推导

- 使用二义文法——消除二义性
    - 层次化描述：优先级与结合性
    - 重命名：悬空else结构的改良

before:

stmt→ if expr then stmt | if expr then stmt else stmt | other

after:

stmt→matched_stmt | unmatched_stmt

matched_stmt→if expr then matched_stmt else matched_stmt | other

unmatched_stmt→if expr then stmt | if expr then matched_stmt else unmatched_stmt

LL文法

LL(k):可以通过固定向前看k个字符预测分析出需要应用的产生式的文法

- L——从左向右扫描
- L——产生最左推导
- k——每一步只需向前看k个输入符号来决定语法分析动作
- 严格定义

  对G中的任意两个不同的产生式$A \to \alpha|\beta$

  1. 不存在终结符号a，使得α和β都能推导出以a开头的串
  2. （同上）α和β不能都推导出空串
  3. 若$\alpha \Rightarrow^* \epsilon$，则$\beta$不能推导出任何以$FOLLOW(A)$中终结符开头的串
- 等价定义

  对G中的任意两个不同的产生式$A \to \alpha|\beta$

  - $FIRST(\alpha) \cap FIRST(\beta) = \varnothing$(1+2)
  - 若$\epsilon \in FIRST(\alpha)$，则$FISRT(\beta) \cap FOLLOW(A) = \varnothing$
- fact
  - 左递归的文法不是LL(1)文法
  - 二义性的文法不是LL(1)文法
- 改写技巧
  - 提取左公因子
  - 消除左递归的方法
    - α, β是不以A开头的文法符号串
    - 情况一

$$A \to A\alpha|\beta \Rightarrow \begin{cases} A \to \beta R \\ R \to \alpha R|\varepsilon \end{cases}$$

- 情况二

$$A \to A\alpha|A\beta|\gamma \Rightarrow \begin{cases} A \to \gamma R \\ R \to \alpha R|\beta R|\varepsilon \end{cases}$$

- 通用方法

```
1)  按照某个顺序将非终结符号排序为 A1, A2, ..., An.
2)  for ( 从 1 到 n的每个 i ) {
3)      for ( 从1 到 i − 1的每个 j ) {
4)          将每个形如 Ai → Ajγ 的产生式替换为产生式组 Ai → δ1γ | δ2γ | ··· | δkγ,
            其中 Aj → δ1 | δ2 | ··· | δk 是所有的Aj产生式
5)      }
6)      消除 Ai 产生式之间的立即左递归
7)  }
```

LR文法

LR(k)文法：使用LR(k)语法分析时无冲突的文法

- SLR(1)文法/LR(0)文法
  - SLR(1)文法都是无二义的，但是反之不成立
- 二义方法都不是LR文法

文法与正则表达式

- 文法的表达能力更强：每个可以用正则表达式表示的构造都可以用文法描述，但是反之不成立；每个正则语言都是上下文无关语言
- 非上下文无关文法：计三个参数的个数
- 上下文无关方法：计两个参数的个数
- 正则表达式：计一个参数的个数

正则表达式

- 字母表、串、语言

    - 子串：删除前缀或后缀
    - 子序列：删除任意符号
- 串的运算
    - 连接/乘积
    - 幂：$x^n = x^{n-1}x \quad (x^0 = \epsilon)$
    - 优先级：幂>连接
- 语言的运算
    - 并
    - 连接
    - 幂：$L^n = L^{n-1}L$（L^0是$\{\varepsilon\}$）
    - 闭包*:$L^* = L^0 \cup L^1 \cup L^2 \cdots$
    - 正闭包+:$L^+ = L^1 \cup L^2 \cdots$
- 正则表达式的归纳定义
    - 空串是正则表达式
    - 单个符号$\mathrm{a} \in \Sigma$是正则表达式
    - 四种构造方法：或、连接、*、( )
        - 代数定律

定律	描述
$r\|s = s\|r$	|是可以交换的
$r\|(s\|t) = (r\|s)\|t$	|是可结合的
$r(st) = (rs)t$	连接是可结合的
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$	连接对|是可分配的
$\epsilon r = r\epsilon = r$	ϵ是连接的单位元
$r^* = (r\|\epsilon)^*$	闭包中一定包含ϵ
$r^{**} = r^*$	*具有幂等性

    - 扩展的运算符号+ ？
- 正则表达式等价：两个正则表达式表示的语言相同
- 正则集合/正规集：可以用正则表达式表示的语言
    - 判定：可以用下列任意一种表示方法表达的语言是正规语言

正规文法仅有三种形式的产生式：
（1）A->aB （2）A->a （3）A->ε

正规语言有三种等价的表示方法：
（1）正规文法 （2）正规式 （3）有限自动机

- 正则定义：为正则表达式命名
    - 自底身上的定义

CH2 词法分析

接受**字符流**，根据**模式**生成**词素(lexeme)序列**，每个词素用**词法单元(token)**表示

基本概念

- 词素：一个字符序列
- 模式：某个词法单元的词素可能具有的形式（通常用正则表达式表示）
- token: `<token_name, attribute_value>`
    - token_name: 语法分析中使用的抽象符号
    - attribute_value: 指向符号表中的条目
        - 一个词法单元至多有一个相关的属性值，但可以是一个组合了多种信息的结构化数据

实现功能

- 两个阶段
  - 扫描阶段
  - 词法分析阶段
- 识别**关键字和标识符**
  - 关键字：语言中具有专门意义及用途的字符串
  - 标识符：变量名字
  - 保留字：不可用作一般标识符的字符串
  - C中的关键字都是保留字，但是也有语言中的关键字不是保留字
- 识别常量
- 识别终结符（标点符号/运算符）
- 剔除空白和注释
- 使用输入缓冲区预读下一个字符
- 将编译器生成的出错消息与源程序的位置联系起来

基本步骤

- 用正则表达式表示各个词素的模式
- 将模式转换为状态转换图
  - 手工构造
  - 算法构造
    - 用正则表达式构造NFA
    - （NFA转换为DFA）
- 根据状态转换图构造词法分析器代码

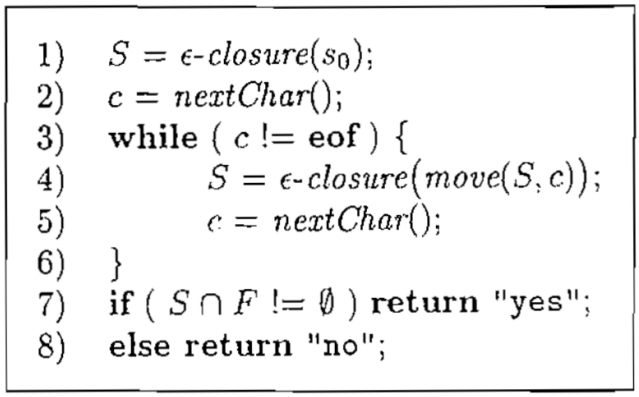
状态转换图

- 组成部分
  - 开始状态：用标有开始的边表示
  - 接受状态/最终状态集合：表明已经找到一个词素（双圆圈表示）（需要回退的最终状态需要加星号）
  - 其它状态（有穷的）
  - 输入字母表
  - 转换函数
- 转换图表示
  - 结点——状态
  - 边——转换函数
- 转换表表示
  - 行——状态
  - 列——字母表
- 注意：为关键字/保留字建立单独的状态转换图；或者将其填入符号表中表明他们不是普通标识符

不确定的有穷自动机NFA

- 特征
  - 对边上的标号无任何限制
  - 空串也可以做标号
- 从正则表达式构造NFA
  - 基本规则
    - 空串
    - 一个字母的正则表达式
    - 两个正则表达式的并
    - 两个正则表达式的连接
    - 一个正则表达式的闭包

    - 归纳得到NFA
- 直接模拟NFA：在转换比直接模拟更耗时的情况下



```
1)  S = ε-closure(s0);
2)  c = nextChar();
3)  while ( c != eof ) {
4)          S = ε-closure(move(S, c));
5)          c = nextChar();
6)  }
7)  if ( S ∩ F != ∅ ) return "yes";
8)  else return "no";
```

    - 读入每一个符号时，存储可以转换到的所有状态的集合，读取结束后检查读入当前字符可以转换到的所有状态中是否有接受状态
    - 实现方法：堆栈+标记数组+状态转换表
    - 时间复杂度$O(k(m+n))$
        - k-输入串长度
        - m-NFA边数
        - n-NFA结点数

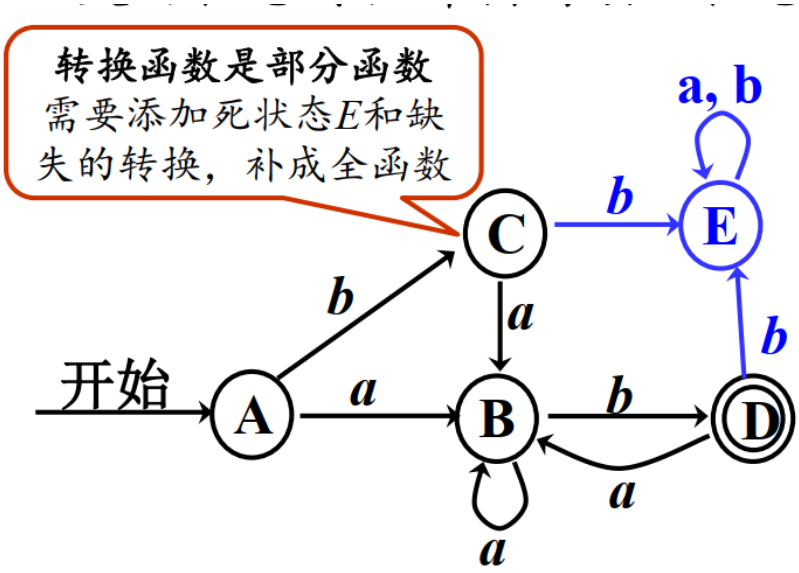
确定的有穷自动机DFA

- 特征
    - 对于每个状态及自动机输入字母表中的每个符号，有且只有一条离开该状态、以该符号为标号的边
- 子集构造法：NFA→DFA
    - DFA的每个状态是NFA的状态的真子集
- 模拟DFA

```
s = s0;
c = nextChar();
while ( c != eof ) {
        s = move(s, c);
        c = nextChar();
}
if ( s在F中 ) return "yes";
else return "no";
```

    - 时间复杂度$O(k)$

- 最小化DFA
    - 加入死状态，转换函数变成全函数



错误恢复

所有词法单元的模式都无法和剩余输入的某个前缀相匹配

- 恐慌模式：从剩余的输入中不断删除字符，直到词法分析器能够在剩余输入的开头发现一个正确的词法单元为止

- 一次变换
  - 尝试通过一次变换将剩余输入变为一个合法的词素
  - 删除一个字符
  - 插入一个字符
  - 替换一个字符
  - 交换两个相邻字符

词法分析器生成工具（3.8—3.9）

根据用户用正则表达式描述的词法单元模式，生成词法分析器

合并词法分析与语法分析的优劣

CH3 语法分析

语法：该语言程序的正确形式

- 功能
  - 使用词法单元的第一个分量token_name创建树形的中间表示（如语法树）
  - 验证输入串可以由源语言的文法生成：找出从文法的开始符号推导出输入的终结符号串的方法
  - 报告语法错误，并从常见的错误中恢复并处理程序的其余部分
- 三种方法
  - 通用的
  - 自顶向下（构造语法分析树结点的顺序）
  - 自底向上

  语法分析器的输入总是按照从左向右的方式被扫描

  高效的自顶向下/自底向上方法只能处理部分文法（LL/LR）

自顶向下

- 递归下降语法分析的通用形式
  - 尝试+回溯以确定正确的产生式
  - 每个非终结符一个分析函数
- 递归下降语法分析的特例——预测分析器
  - 向前看固定k个字符直接确定正确的产生式
  - 转换图：每个非终结符一个图
  - 预测分析表生成方法

> 方法：对于文法 G 的每个产生式 $A\to\alpha$，进行如下处理：
> 1）对于 FIRST(α) 中的每个终结符号 a，将 $A\to\alpha$ 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 $A\to\alpha$ 加入到 $M[A, b]$ 中。如果 ϵ 在 FIRST(α) 中，且 \$ 在 FOLLOW(A) 中，也将 $A\to\alpha$ 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）。 □

  - 非递归形式：由表驱动
  - 错误恢复
    - 恐慌模式：不断丢弃输入中的符号，直到找到某个同步词法单元（优化同步集合的设置P145）
    - 短语层次恢复：在预测分析表的空白条目处填写指向处理程序的指针

自底向上

移进-归约方法

- 归约：最右推导的反向步骤
- 句柄：若$S \Rightarrow_{rm}^{*} \alpha Aw \Rightarrow_{rm} \alpha\beta w$，则$\beta$是句柄
  - 和某个产生式体匹配的最左子串不一定是句柄
  - 句柄右边的串w一定只包含终结符号
  - 无二义的文法中，每个右句型有且只有一个句柄

    - 句柄总是出现在栈的顶端
- 移进-归约分析器的四种动作
    - 移进
    - 归约
    - 接受
    - 报错
- 不能使用移进-归约分析的两种冲突
    - 移进-归约冲突：即使知道了栈中的所有内容和接下来的k个输入符号，仍然无法判断应该移进还是归约
        - 看归纳产生式左部符号的FOLLOW集合中是否有待移进字符
    - 归约-归约冲突：不能根据栈中内容和下一个输入符号确定应该使用哪个产生式进行归约
        - 看两个产生式左部符号的FOLLOW集合是否有交集

LR分析

LR分析方法是移进-归约方法的特例

- 特点
    - 由表驱动
    - 无回溯
    - 可以用LR分析的文法不一定可以用LL分析，可以用LL分析的一定可以用LR分析
- 方法
    - 写增广文法
    - LR(0)——SLR(1)
        - 项：加点的产生式
            - 特例$A \to \epsilon$只生成一个项$A \to \cdot$
        - 项集
            - 内核项：初始项$S` \to \cdot S$ 或 点还在最左端的项

        - 非内核项
    - 根据转移函数完成LR语法分析表

① 如果[$A\to\alpha\cdot a\beta$]在 I_i 中并且 GOTO(I_i, a) = I_j，那么将 ACTION[i, a]设置为“移入 j”。这里 a 必须是一个终结符号。

② 如果[$A\to\alpha\cdot$]在 I_i 中，那么对于 FOLLOW(A)中的所有 a，将 ACTION[i, a]设置为“归约 $A\to\alpha$”。这里 A 不等于 S'。

③ 如果[$S'\to S\cdot$]在 I_i 中，那么将 ACTION[i, \$]设置为“接受”。

    (若下一个字符为a且状态j上有一个在a上的转换，则移进a，否则归约)

    - 操作
        - 输入符号入栈
        - 若出现句柄，则弹出归约，再入栈，继续执行
        - 若不出现句柄则继续执行
- 规范LR(1)
    - 项：加点的产生式+搜索符
    - LR(1)项$[A\to\alpha\cdot\beta,a]$对活前缀γ有效的条件：存在一个推导$S\Rightarrow_{rm}^{*}\delta Aw\Rightarrow_{rm}\delta\alpha\beta w$, 其中$\gamma=\delta\alpha$且（要么a是w的第一个符号，要么w为$\epsilon$ && $a=\$$）
- LALR(1)
    - 状态数与SLR相同，语法分析表与SLR规模近似
    - 在不产生冲突的情况下合并同心项目集（行为一致）
        - 不会产生新的移进-归约冲突
        - 可能产生归约-归约冲突
    - LALR语法分析器比规范LR报错慢（可能多执行一些归约）
    - 一种高效生成LALR分析表的方法

- 活前缀
    - 活前缀：一个最右句型的（真）前缀，可以通过在其后增加一些终结符号来得到一个最右句型

    - 识别活前缀的NFA
        - 一个项一个状态
    - 识别活前缀的DFA
- 压缩分析表
    - 相同的行使用同一个指针
- 错误恢复
    - 恐慌模式
        - 从栈顶开始向下找到一个状态s（可以读入A后转移），则丢弃若干输入符号，直到发现一个可以合法跟在A之后的符号a
        - 试图消除包含语法错误的短语
    - 短语层次错误恢复
        - 修改栈顶状态（和/或）第一个输入符号
        - 在LR分析表的空白条目处填写指向处理程序的指针

错误恢复

已扫描的输入部分不可能存在正确的后续符号串

- 检测到一个就退出/检测到错误恢复后继续检测，直到错误达到某个上限再停止
- 恐慌模式
    - 不断丢弃输入中的符号，直到找到某个同步词法单元
    - 同步词法单元由编译器决定，通常是界限符（如分号、花括号）
    - 保证不会进入无限循环
- 短语层次恢复
    - 替换剩余输入串的前缀、删除、插入，使语法分析器可以继续分析
    - 需要注意避免进入无限循环（一种方法：保证任何恢复作业最终都会消耗掉某个输入符号）
    - 难以处理实际错误发生在被检测位置之前的情况

- 错误产生式
    - 事先加入特殊的产生式，可以产生含有错误的构造；若分析中使用了错误产生式，则可以生成相应的错误信息
- 全局纠正
    - 对于输入串X和文法G，得到最小改动序列进行恢复
    - 开销太大，只具有理论价值

CH3.5 语法翻译

L属性翻译——从左向右翻译

语法翻译主要应用于语法树的构造（一个文法在面对不同目的时，有不同的语法制导定义和翻译方案）

- 语法制导定义
    - 将文法符号与属性集合相关联；将产生式与语义规则相关联
    - 属性：与某个程序构造相关的任意的量，在语法制导定义中与文法符号相关联
        - 综合属性：由子结点及本身的属性得到的属性（只需要一次自底向上的遍历就可以计算出）
        - 继承属性：由父结点、左边的兄弟结点及本身的属性得到的属性
        - 终结符号也可以有综合属性
            - 终结符号的属性值是词法分析器的词法值
            - 语法制导定义语法规则中不计算终结符号的属性值
        - 计算次序
            - 综合属性：任何自底向上的顺序计算
            - 分析树方法
                - 属性依赖图→拓扑排序
                    - 一个属性一个结点
                    - 一条边：计算第二个属性实例时需要第一个属性实例的值

        - （手工）基于规则的方法：无视具体输入，根据语义规则静态确定计算次序
        - （自动）忽略规则的方法
    - 语义规则: 若语法分析树上结点及其子结点满足某个产生式，则可以通过语义规则计算出变个结点属性的值
    - S属性定义：只包含综合属性
    - L属性定义：在产生式体所关联的各个属性之间，依赖图的边总是从左到右的
        - 所有LL文法都可以改成L属性定义，对LR文法则不一定
- 翻译方案：将程序片段附加到产生式上
    - 程序执行时机$B \to X\{a\}Y$
    - 自底向上: X的此次出现位于语法分析栈栈顶时立刻执行动作a
    - 自顶向下：试图展开非终结符Y的本次出现/输入终结符Y之前执行动作a
    - 可能需要拓广文法，确定开始符号，完成初始化
- 自顶向下语法分析中翻译
    - 分析函数的参数是非终结符的继承属性
    - 分析函数的返回值是非终结符的综合属性的集合
- 自底向上语法分析中翻译
    - 栈操作代码
    - 属性值存储和状态栈中的文法符号一一对应放在栈中（非终结符用占位符）
    - 归约时使用top指针指向产生式右部最后一个符号
    - 分析程序在语义动作执行后修改top
- 对于任何上下文无关文法，都可以构造出$O(n^3)$完成长度为n的终结符串的语法分析器
- 递归下降的语法分析器——预测分析器
    - 控制流可以由向前看符号无二义的确定
    - 检查向前看符号，模拟被选中产生式的体

- 预测翻译器：扩展预测分析器得到

CH4 语义分析

语义：含义

使用语法树和符号表中的信息，检查源程序是否与语言定义的语义一致；收集类型信息，把信息存在语法树/符号表中

- 类型检查：检查 每个运算符是否有对应的运算分量