1. **引论**

1、翻译器：能够完成从一种语言到另外一种语言的保义变换的软件。这两种语言分别称为源语言和目标语言。

2、**编译器**：一种翻译器，它的目标语言比源语言低级。

3、**解释器**：不同于编译器的另一类语言处理器。

4、代码生成：取源程序的一种中间表示作为输入并把它映射到一种目标语言

5、前端：只依赖于源程序，独立于目标机器（词法分析，语法分析，语义分析，中间代码生成）

后端：依赖于目标机器，与源程序无关，只与中间语言有关（代码优化，代码生成）

好处：提高开发编译器的效率

6.源于几个阶段的活动常用**一遍**扫描来实现，一遍扫描包括读一个输入文件和写一个输出文件。

**课后习题：**

1.编译器和解释器的区别：解释器不是编译器那样通过翻译来生成目标程序，而是直接执行源程序所指定的运算。

1. **词法分析**
2. 串：字符表符号的有穷序列
3. 语言：字母表上的一个串集，属于该语言的串成为该语言的句子或字。
4. 语言运算的定义：
   1. 和：*L*∪*M* = {*s* | *s* ∈*L* 或 *s* ∈ *M* }
   2. 连接：*LM* = {*st* | *s* ∈ *L* 且 *t* ∈ *M*}
   3. 指数：*L*0是{ε}，*Li*是*Li* -1*L*
   4. 闭包：*L*\* = *L*0 ∪ *L*1 ∪ *L*2 ∪…
   5. 正闭包： *L*+ = *L*1 ∪ *L*2 ∪…
5. 正规式：按照一组定义规则，由较简单的正规式构成的，每个正规式 r 表示一个语言 L(r)

正规式表示的语言叫正规语言或正规集

1. 运算符的优先级：\* > 连接运算 > |
2. 如果两个正规式r和s表示同样的语言，就说r和s等价（r=s）

**2.3 有限自动机**

1、不确定的有限自动机：缺点： 1）输入字符包括 ε

2）一个状态对于某个字符，可能有多条输出边

2、NFA的转换表:优点：快速定位；缺点：字母表过大或大部分转换状态为空集时浪费空间

3、DFA：NFA的特殊形式 优点：1）输入字符不包括 ε

2）一个状态对于某个字符，只可能存在唯一条输出边

4、子集构造法：从NFA构造出可识别同样语言的DFA

5、DFA的化简：加“死状态”

**2.4 从正规式到有限自动机**

*1、1）N*(*r*)的状态数最多是*r*中符号和算符总数的两倍。

*2）N*(*r*)只有一个开始状态和一个接受状态，接受状态没有向外的转换。

*3）N*(*r*)的每个状态有一个用Σ的符号标记的指向其它结点的转换，或者最多两个指向其它结点的ε转换

**2、**Lex(构造词法分析器)程序包括三个部分：

声明

％％

翻译规则

％％

辅助过程

**课后习题：**

**2.3**

1. 首尾均为0的二进制串
2. 0，1组成的二进制串，包括空串
3. 倒数第3位为0的二进制串
4. 包含且仅包含3个1的二进制串
5. 1的个数和0的个数均为偶数的二进制串
6. **语法分析**
7. 正规式来定义一些简单的语言，能表示给定结构的固定次数的重复或者没有指定次数的重复。
8. 上下文无关文法是四元组（*VT* , *VN* , *S*, *P*）

*VT* : 终结符集合(非空有限集合，记号名是其同意词)

*VN* : 非终结符集合（非空有限集合, ）

*S* : 开始符号(非终结符)

*P* : 产生式集合， 产生式形式:*A* → α *,* 

1. 句子是只含终结符的句型。
2. 最左推导，最右推倒（规范推倒）
3. NFA->上下文无关文法： 确定终结符集合

为每个状态引入一个非终结符Ai

如果状态i有一个a转换到状态j,引入产生式Ai🡪aAj,如果i是接受状态，则引入Ai🡪 ε

1. 一个文法是左递归的：A⇒+Aa

直接左递归 *A*→*Aa* |*b*

消除直接左递归 *A* → *b A*′ *A*′→ *a A*′ | ε （P50更详细）

1. 有左因子的文法 *A* →*αβ*1 *| αβ*2

提左因子 *A → α A′ A′* *→ β*1 *| β*2

1. 有限自动机不能计数，上下文无关文法可以记两项的数
2. 文法 *G* = (*VT* , *VN*, *S*, *P*)

0型文法（短语文法）：*α* → *β*，*α , β* ∈ (*VN* ∪*VT*)\*, | *α* | ≥ 1

1型文法（上下文有关文法）：| *α* | ≤ | *β* |，但*S* → ε可以例外，此时S不出现在任何产生式的右侧。

2型文法（上下文无关文法）：*A* → *β*，*A*∈*VN* , *β* ∈ (*VN* ∪*VT*)\*

3型文法（正规式）：*A* → *aB*或*A* → *a*，*A, B*∈*VN* , *a* ∈*VT*

**3、3 自上而下分析**

1、自上而下分析的宗旨是，对任何输入串，试图用一切可能的办法，从文法开始符号（根结点）出发，自上而下，从左到右的为输入串建立分析树。或者说，为输入串寻找最左推导。

2、使用自上而下分析时，文法应该没有左递归，否则有可能使分析过程陷入无限循环。

3、LL(1)文法：1）FIRST集合计算方法

* + 若X🡪a..， 则将终结符ａ加入FIRST(X)中
  + 若X🡪ε，则将ε加入FIRST(X)中
  + 若X🡪Y…，且Y属于非终结符，则将FIRST(Y)\{ε}加入到FIRST(X)中
  + 若X🡪Y1Y2..YK,且Y1,Y2,..Yi-1都是非终结符，且Y1,Y2,..Yi-1的FIRST集合中均包含ε，则将FIRST(Yj)的所有非ε元素加入到FIRST(X)中，（j=1,2,..i）.特别地，若Y1~YK均有ε产生式，则将ε加到FIRST(X)中。

2）FOLLOW集合计算方法

* + 对文法开始符号S,置$于FOLLOW(S)中。
  + 若有A🡪αBβ，则将FIRST(β)\{ε} 加入FOLLOW(B)中。 （此处α 可以为空）
  + 若A🡪α B 或A🡪α B β,且 β ⇒\* ε（即ε 属于FIRST(β)）,则将 FOLLOW(A)加入FOLLOW(B)中（此处α 可以为空）。

3）LL(1)(自左向右，最左推导，向前查看一个输入)文法：满足以下两个条件（也是不出现回溯的两个条件）：1、对*A*的任何两个不同的选择*α* 和β，有FIRST (*α*) ∩ FIRST (β) = ∅

2、若β⇒\* ε，那么FIRST (*α*) ∩ FOLLOW (*A*) = ∅

4）判定文法是否是LL(1)文法步骤如下：

* + 如果有以下两种情况一定不是
    - 左递归
    - 公共左因子
  + 如果不是，则改写文法
    - 消除左递归
    - 提取左因子

4、递归的预测分析器：

*type* → *simple* | ↑ id | array [*simple*] of *type*

proccdure *type*;

begin

if *lookahead* in {integer, char, num} then

*simple*( )

else if *lookahead* = ′↑′ then begin

*match* (′↑′); *match* (id)

end

else if *lookahead* = array then begin

*match* (array); *match* ( ′ [ ′ ); *simple*( );

*match* ( ′ ] ′ ); *match* (of ); *type*( )

end

else *error*( )

end;

5、构造预测分析表（非递归）

（1）对文法的每个产生式*A* → *α* ，执行(2)和(3)。

（2）对FIRST(*α*)的每个终结符*a*，把*A* → *α* 加入*M*[*A*, *a*]。

（3）如果ε在FIRST(*α*)中，对FOLLOW(*A*)的每个终结符*b*（包括$）, 把*A* → *α*加入*M*[*A*, *b*]。

（4）*M*的其它没有定义的条目都是error。

**3、4 自下而上分析**

1、又称移进—规约分析（最右推倒逆过程）

2、句柄：

1. 句柄与某个产生式的右部符号串相同
2. 句柄是句型的一个子串
3. 把句柄归约成非终结符代表了最右推导逆过程的一步

性质：1）句柄的右边仅含终结符*。* 2）如果文法二义，那么句柄可能不唯一。

3、分析器的四种动作：

1. 移进动作 把下一个输入符号压栈。
2. 规约动作 分析器知道整个句柄已经完全出现在栈顶，它确定句柄的左端在栈中的位置，在决定采用哪个非终结符来代替句柄（即确定使用哪个产生式）。
3. 接受动作 分析器宣告分析成功。
4. 报错动作 分析器发现了语法错误，调用错误恢复例程。

4、移进—规约冲突：优先移进 规约—规约冲突：改文法

**3、5 LR分析器**

1、LR(k):自左向右，最右推倒的逆，向前查看k个符号；SLR,LALR,LR

2、LR文法：一个文法，如果能为他构造出所有条目都唯一的LR分析表，就说它是LR文法。

3、前缀:一个符号串的前缀是指从第一个符号开始的连续的若干个符号构成的子串。

活前缀：右句型的前缀，该前缀不超过最右句柄的右端



4、LR(0)项目（简称项目）：在右部的某个地方加点的产生式

5、闭包函数closure(I)(SLR) 1）I的每个项目均加入closure(I)

2）如果A🡪*α·Bβ*在closure(I)中，且B🡪γ是产生式，那么如果项目B🡪 *·*γ还不在closure(I)中的话，那么把它加入。

6、核心项目：E’🡪·E及所有的点不在左端的项目

非核心项目：除了E’🡪·E以外，所有点在左端的项目

1. 从DFA构造SLR分析表

状态*i*从*Ii*构造，它的*action*函数如下确定：

* 1. 如果[*A*→*α*·*aβ*]在*Ii*中，并且*goto*(*Ii*, *a* ) = *Ij*，那么置*action*[*i*, *a*]为*sj*。
  2. 如果[*A*→*α*·]在*Ii*中，那么对FOLLOW(*A*)中的所有*a*，置*action*[*i*, *a*]为*rj*,*j*是产生式 *A*→*α*的编号。
  3. 如果[*S*′→*S*·]在*Ii*中，那么置*action*[ *i*, $ ]为接受*acc*。

如果出现动作冲突，那么该文法就不是SLR(1)的。

使用下面规则构造状态i的goto函数：

对所有的非终结符A，如果goto(Ii, A) = Ij, 那么goto[i, A] = j。

8、LR(1)项目：重新定义项目，让它带上搜索符，成为如下形式[*A*→*α*·*β*, *a*]

9、闭包函数closure(I)(LR) 1、I的每个项目均加入closure(I)

2、如果[A🡪α*·Bβ, a*]*在* closure(I)中，且B🡪η是产生式，那么如果项目[B🡪*·*η*,b* ]还不在closure(I)中的话，那么把它加入(其中*b*属于FIRST(*βa*))。

10、构造LR(1)分析表：

从*Ii*构造分析器的状态*i*, 状态*i*的*action*函数如下确定

* + 如果[*A* →*α*·*aβ*, *b*]在*Ii*中，且*goto*(*Ii*, *a*) = *Ij* ，那么置*action*[*i*, *a*]为*sj*。
  + 如果[*A*→*α* ·, *a*]在*Ii*中，且*A* ≠ *S*′，那么置*action*[*i*, *a*]为*rj* .
  + 如果[*S*′→*S*·, $]在*Ii*中，那么置*action*[*i*, $] = *acc*。

如果用上面规则构造出现了冲突，那么文法就不是LR(1)的。

状态*i*的*goto*函数如下确定：

如果*goto*(*Ii*, *A*) = *Ij*, 那么*goto*[*i*, *A*] = *j*

10、同心的LR(1)项目集：略去搜索符后它们是相同的集合。

* 同心集的合并不会引起新的移进−归约冲突
* 同心集的合并有可能产生新的归约−归约冲突

11、构造LALR(1)分析表

* 构造LR(1)项目集规范族*C* = {*I*0, *I*1, …, *In*}。
* 寻找LR(1)项目集规范族中同心的项目集，用它们的并集代替它们。
* 按构造规范的LR(1)分析表的方式进行构造。

12、看LR(1)文法是否冲突：构造DFA，看是否有移进—归约（看相同的式子有没有一个点已经到了最后，而另一个还在中间）冲突或归约—归约（两个不同的式子点都在最后）冲突。

12、Yacc(语法分析器的生成器)程序包括三个部分

声明

％％

翻译规则

％％

支持例程

13、解决分析动作冲突的两大默认规则：

* + 对于归约-归约冲突，选择在Yacc 程序中最先出现的那个产生式归约
  + 对于移进-归约冲突，优先移进

1. **语法制导的翻译**
2. 综合属性：父节点的值依赖于子节点或自身。
3. 继承属性：子节点的值依赖于兄弟和父亲。
4. 每个文法符号的综合属性集和继承属性集的交集应为空。
5. S属性定义：仅使用综合属性的语法制导定义(属性计算可以自下而上的完成)。
6. 注释分析树：每个结点的属性值都标注出来的分析树。注释分析树中计算各节点属性值的过程称为注释或修饰。
7. 非终结符🡨🡪分析过程（函数）

综合属性🡨🡪过程的返回值

继承属性🡨🡪过程的参数

1. 语法树是分析树的浓缩表示：算符和关键字是作为内部结点。

8、有综合属性域的分析栈：拓展后分析栈的每个栈元素由状态域state和属性域val组成。

9、*L*属性定义：如果每个产生式*A* →*X*1 *X*2 … *Xn* 的每条语义规则计算的属性是*A*的综合属性；或者是*Xj* 的继承属性，1 ≤ *j* ≤ *n*, 但它仅依赖：

* 1. 该产生式中*Xj*左边符号*X*1, *X*2, …, *Xj*-1的属性（综合，继承）；
  2. *A*的继承属性。

10、L属性定义的翻译方案的三条限制：

* 产生式右部符号的继承属性必须在先于这个符号的动作中计算。
* 一个动作不能引用该动作右边符号的综合属性，但可用右边符号的继承属性。
* 左部非终结符的综合属性只能在它所引用的所有属性都计算完后才能计算。

11、*D* → id *L* { *addtype* (id. *entry*, *L*. *type*)}

*L* → , id *L*1 {*L*. *type* := *L*1. *Type*;

*addtype* (id. *entry*, *L*1. *type*)}

*L* → : *T* {*L*. *type* := *T*. *type*}

*T* → integer {*T*. *type* := *integer*}

*T* → real {*T*. *type* := *real*}

12、继承属性赋初值在拓广文法里。

13、为文法 *S* → ( *L* ) | *a*

*L* → *L* , *S* | *S*

写一个**语法制导定义**，它输出括号的对数。

*S*′ → *S* *print* (*S*. *num*)

*S* → ( *L* ) *S*. *num* := *L*.*num* + 1

*S* → *a* *S*. *num* := 0

*L* → *L*1 , *S* *L*. *num* := *L*1. *num* + *S*. *num*

*L* → *S* *L*. *num* := *S*.*num*

14、给出对表达式求导数的**语法制导定义**，表达式由+和\*作用于变量x和常数组成，如x\*(3\*x+x\*x),并假定没有任何简化，例如将3\*x翻译成3\*1+0\*x。

E’🡪E printf(E.s)

E🡪E1 + T E.s = E1.s ‘+’ T.s;

E.v = E1.v ’+’ T.v

E🡪T E.s = T.s;

E.v = T.v

T🡪 T1 \* F T.s = T1.v ‘\*’ F.s ‘+’ T1.s ‘\*’F.v

T.v = T1.v ‘\*’ F.v

T🡪 F T.s = F.s; T.v = F.v

F🡪(E) F.s = ‘(’E.s’)’; F.v = ‘(’E.v’)’

F🡪num F.s = ‘0’;

F.v = lookup(num.entry)

F🡪x F.s = ‘1’; F.v = ‘x’

15、为文法 *S* → ( *L* ) | *a*

*L* → *L* , *S* | *S*

写一个**翻译方案**，它输出每个*a*的嵌套深度。例如，对于( *a* , ( *a* , *a*) )，输出的结果是1 2 2。

*S*′ → {*S*. *depth* := 0 } *S*

*S* → {*L*. *depth* := *S*. *depth* + 1 } ( *L* )

*S* → a {*print* (*S*. *depth*) }

*L* → {*L*1. *depth* := *L*. *depth* }*L*1 ,

{*S*. *depth* := *L*. *depth* }*S*

*L* → {*S*. *depth* := *L*. *depth* }*S*

16、为文法 *S* → ( *L* ) | *a*

*L* → *L* , *S* | *S*

写一个**翻译方案**，它打印出每个*a*在句子中是第几个字符。例如，当句子是( *a* , ( *a* , ( *a* , *a* ) , (*a*) ) )时，打印的结果是2 5 8 10 14。

*S′* →{*S*. *in* :=0 } *S* {*S*. *in* :=1}？

*S* →({*L*. *in* := *S*. *in* +1 } *L* ) {*S*. *num*:= *L*.*num*+ 2 }

*S* → *a* {*S*. *num* :=1; *print* (*S*. *in+1*) } *print* (*S*. *in*)?

*L* →{*L*1. *in* := *L*. *in* }*L*1 ,

{*S*. *in* := *L*1. *in* + *L*1.num+1 } *S*

{*L*.num:= *L1.nu*m + S.num +1 }

L → {S. in := L. in }S {L. num := S.num }

**第六章 运行时存储空间的组织和管理**

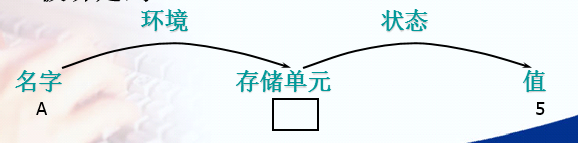
1、过程的活动：过程的一次执行称为过程的一次活动

活动记录：过程的活动需要可执行代码和存放所需信息的存储空间,后者称为活动记录

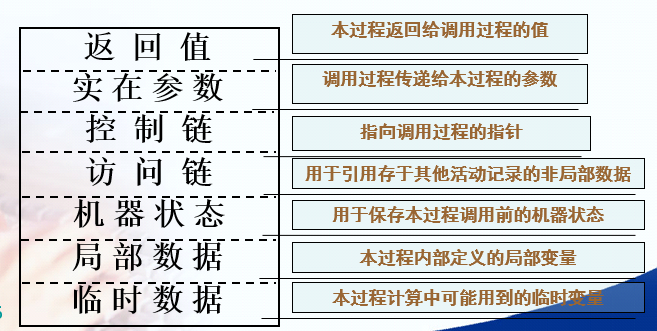
2、过程P一次活动的生存期：指的是从执行该过程体第一步操作到最后一步操作之间的操作时间，包括执行P时调用其它过程花费的时间。

3、名字到存储单元的绑定

* + 环境把名字映射到左值（存储单元），而状态把左值映射到右值（值）。
  + 赋值改变状态，但不改变环境。
  + 如果环境将名字*x*映射到存储单元*s*，我们就说*x*被绑定到*s*。



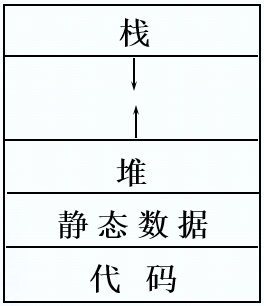
4、一般的活动记录（地址高->低）：临时数据，局部数据，保存的机器状态，访问链，控制链，返回值，实参。



5、字节是可编址内存的最小单位。

对齐问题，由于对齐而引起的无用空间称为衬垫空白区。

1. 最接近的嵌套作用域规则

并列程序块不会同时活跃

Top->

并列程序块的变量可以重叠分配

1. 运行时内存的划分：
2. 过程活动的生存期要么嵌套，要么无重叠

9、静态分配给语言带来限制

递归过程不被允许

数据对象的长度和它在内存中位置，必须是在编译时可以知道的

数据结构不能动态建立

10、栈式分配

* + 栈式分配主要用于管理过程的活动记录。
  + 局部变量的生存期是过程活动的时间。
  + 控制进入该过程时，局部变量绑定到存储单元，过程活动结束后，局部变量的空间释放。

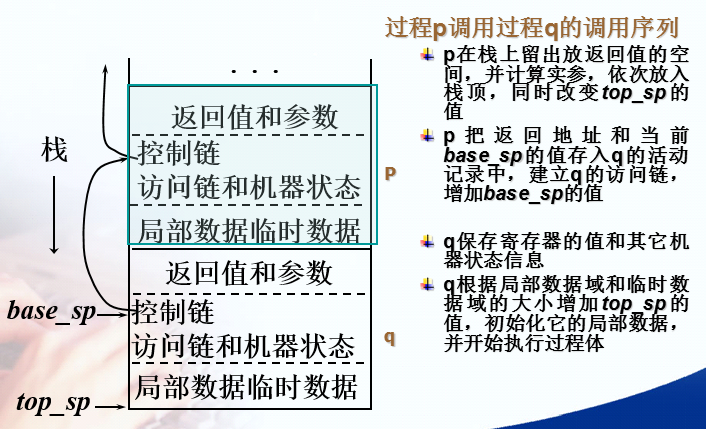
11、活动树:用树来描绘控制进入和离开活动的次序

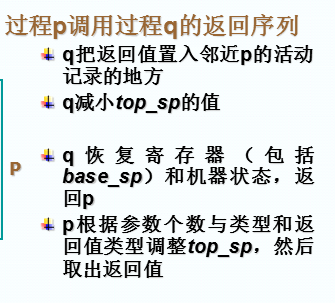
运行栈：如果控制栈中的信息包含过程活动的活动记录，控制栈就成了活动记录栈，通常称之为运行栈。

12、过程调用序列：过程调用时执行的分配活动记录，把信息填入它的域中的代码

过程返回序列：过程返回时执行的恢复机器状态，释放活动记录，使调用过程能够继续执行的代码

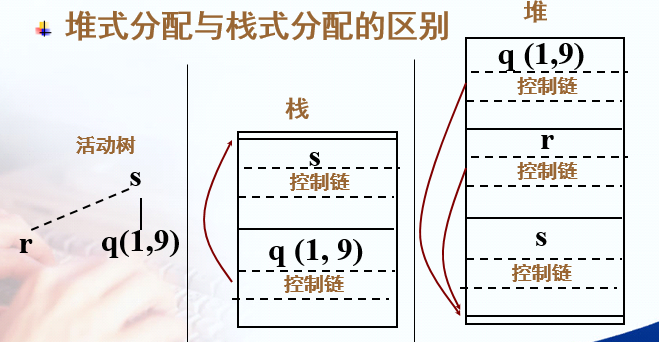
13、





14、堆式分配

* 内存分配与释放按照任意次序进行
* 堆中可能包含交错的正在使用的和已经释放的区域

15、

16、三种存储分配策略的比较



17、通常把编译时不能确定所需空间的数据分配到堆上

尽可能把他们分配在栈上而不是堆上的理由是避免垃圾收集器收集他们的开销

只有那些局部与一个过程并在该过程返回后不可访问的数据对象才能分配到栈上

18、只要存储空间可以回收，就可能出现悬空**引用问题**

19、非局部名字的访问：主旨：

静态：看外围过程（静态链=访问链） 动态：看调用过程（动态链=控制链）

20、假定过程p的嵌套深度为*np*，它引用嵌套深度为*na*的变量a，*na* ≤ *np*：从栈顶的活动记录开始，追踪访问链*np* − *na*次。

21、建立访问链:假定嵌套深度为*np*的过程p调用嵌套深度为*nx*的过程x

*np* < *nx*的情况:x肯定就声明在p中

*np* ≥ *nx*的情况:追踪访问链*np* − （*nx* – 1）次，到达了静态包围x和p的且离它们最近的那个过程的最新活动记录

22、实现动态作用域的方法

* 深访问：用控制链搜索运行栈，寻找包含该非局部名字的第一个活动记录
* 浅访问:为每个名字在静态分配的存储空间中保存它的当前值

当过程p的新活动出现时，p的局部名字*n*使用在静态数据区分配给*n*的存储单元。n的先前值可以保存在p的活动记录中，当p的活动结束时再恢复

23、参数传递：

值调用：实参的右值传给被调用过程 引用调用：实参的左值传给被调用过程

复写-恢复调用：值调用和引用调用的混合

换名调用：用实参表达式对形参进行正文替换。

1、中间代码的几种常用表示方法：后缀表示、图形表示（语法树，有向无环图）和三地址代码（*x* := *y op z*）

2、目标程序：可执行目标模块，可重定位目标模块，汇编语言程序

3、一个名字的值在基本块的某一点以后还要引用的话，我们说这个名字在该点是活跃的

