编译原理

第一章 引论

1.词法分析：识别出关键字，标识符，常数，算符，界符

2.语法分析：符号串分解成语法结构：短语，句子，句型，子程序，程序；并确定语法是否正确。

3.语义分析和中间代码产生：依赖的是语言的语义规则。语义规则由属性文法描述。①对每种语法结构进行静态语义检查（标识符是否定义、类型是否合法等），定义一个程序的意义②翻译成中间代码。中间代码把复杂的语法拆分，用四元式、三元式、间接三元式、逆波兰记号、树形表示。

4.优化：各个部分的优化

5.生成目标代码：中间代码->(优化)->目标代码（二进制，汇编指令或可重定位的指令代码）

6.符号表：记录源程序中出现的标识符（常量，变量，过程）和其各种属性（变量类型，变量地址等）。识别出标识符后立即填入符号表，但属性可能在后续补充填入。

7.遍

8.编译后端：与目标机有关的优化和目标代码生成。只依赖于中间代码。改写后端来生成适应不同目标机器的程序。

编译前端：剩下的部分。

第二章

1.文法：文法是描述语言的语法结构的形式规则（语法规则）。

2.上下文无关文法所定义的语法范畴是完全独立于上下文的。

3.语法分析树：根据语法规则从规则成分推导到具体语法句子的结构。用一个树形的图结构表示一个句型的推导。根为开始符号。

4.终结符号：组成语言的基本符号，不可再分

5.非终结符号：语法范畴（算数表达式，布尔表达式，赋值语句，分程序等），是一个类的记号，包含了若干个体。VT=VN+VT。终结符和非终结符集合不能相交。

6.开始符号，是一种非终结符号，最顶层，“句子”。S。至少在某个产生式的左边出现一次。

7.产生式：定义语法范畴的书写规则。P->a1|a2|a3……

8.推导每前进一步都必须有明确的产生式可以引用。

·直接推出：

·推导：

·a1出发经过至少一步推导出an：

·a1出发经过至少零步推导出an（即a1=an）：

9.G是一个文法，S是开始符号：

·句型：

·句子：

·语言：所产生的所有可能句子的集合（通式）

p30例子。

10.句型之间的推导不是唯一的。

·最左推导：任何一步α->β，都是α中的最左非终结符被替换。最右推导同理。

11.一棵语法树表示了句型的种种可能的推导过程，包括最左/右推导。

12.一个文法中存在某个句子拥有两个不同的最左/右推导：二义的。

·文法中不含P->的产生式；

·每个非终结符P必须有用处：存在含P的句型，并且P必须能推导出一个终结符。

P36练习。

13.有穷字母表，符号，符号串，空字，空集，笛卡儿积

·V\* = {空字，符号串的集合} = V0∪V1∪V2……

·V+ = {符号串的集合} = VV\* ， V\* = V0 ∪V+

14.判断文法是否是二义的：看它的最右推导有几种。

第三章:词法分析

1.词的种类：关键字，标识符，常数，运算符，界符

2.二元式（单词种类，单词符号的属性）

3.单词符号的属性是指单词符号的特性或特征。属性值是反映特征的值。

4.输入缓冲区，预处理，剔除空格和注释，扫描缓冲区

5.标识符：开头是字母，后面是数字或字母。取出来后对照关键字表。

·数字：开头是数字，后面是数字，直到没有数字为止

·算符和界符：注意>=等由两个字符组成的符号

6.状态转换图：有限方向图，结点代表状态。双圆圈代表终态。终态大于等于1个。

·对于简单语法，从0状态开始，遇到字母跳标识符检测，遇到数字跳数字检测，遇到特殊符号一一对应跳界符和算符检测，等等。每次遇到一次终态，说明一个“词”被分析了出来，下一步又要回到0状态开始检测下一个词。注意要返回一步。

**7.正规集：正规式**：P47例子

·空集和空字都是正规式

·任何字母表中的单个元素都是正规式，正规集就是这个元素构成的集合

·U,V,L(U),L(V)是正规的，那么对U、V进行并/笛卡尔积/闭包操作得到的都是正规式/集。由以上三步得到的表达式是字母表上的正规式。

·若两个正规式表示的正规集相同，那么二者等价。

·并运算满足交换律；并和积满足结合律、分配律。

8.**确定有限自动机DFA：**

·S有限集，每个元素是一个状态；

·∑字母表，每个元素为一个输入字符；

·δ(s,a)=s’，δ是一个映射，表示在s状态下输入a会转移到s’。

·s0，S的唯一初态；

·F，S中的终态集（可空，一定要写成集合的形式）。

·δ的全部映射可以写成状态转换矩阵，或画成状态转换图。整个图含有1个初态节点和若干个（包括0）终态节点。

·DFA的确定性体现在每一个映射δ都是单值函数。当前状态和当前输入唯一确定了下一个状态，这样就可以一直周转下去。

**9.非确定有限自动机NFA：**

允许δ是一个多值函数的时候就变成NFA。

·只有δ和DFA有区别：δ：S×∑\*→2S

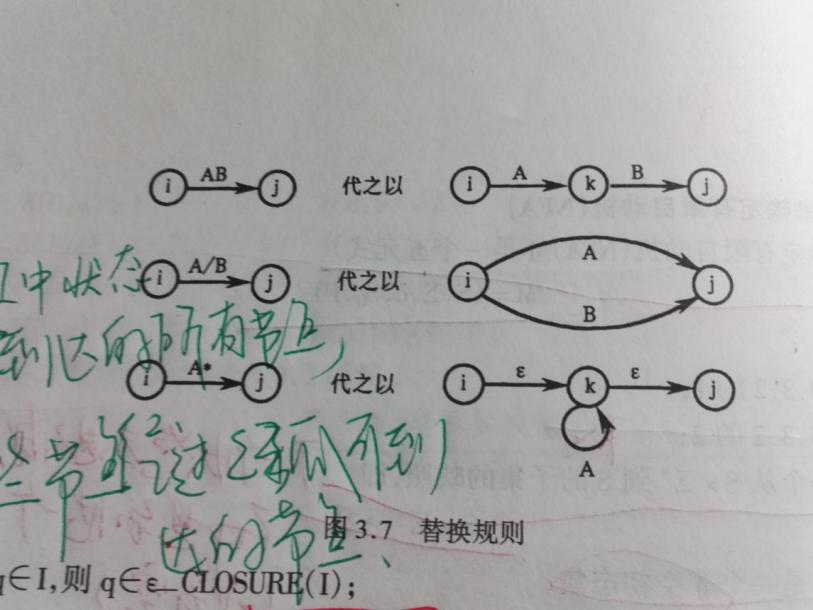
·每个状态节点输出若干个有向边连接其他节点，每条有向边用一个元素或空字做标记。

·如果一个节点既是初态也是终态，或者存在一条从初态到终态的空字通路，那么空字可以被自动机接受。

10. NFA ->DFA:最初的NFA记为M。

·引进新的初态X和终态Y节点。从X到每个初态都连一条，从每个终态到Y都连一条空弧。

·对状态转换图进行替换：这个过程会引入一些新的状态k。 重复这个过程，使每条有向边上的字母都为单个或者空字。得到的即为M’。



·把从X出发能通过空字到达的所有状态和X放在一起，形成一个新状态。然后找出这个新状态整体面对不同输入符号（比如面对a和面对b的时候）分别能找到那些节点（包括可以通过空字找到的），把它们分别记为新状态。每个新状态继续找下去。

·根据得到的新的状态转化表画状态转化图，每个含有Y的状态都被标记为终态。

11.自动机化简

·把状态分成两组：终态和非终态

·对每一组中的所有状态，分别看它们面对每个输入符号（比如a和b）后会跳转的状态的集合。如果跳转后的状态属于原来的状态集合，那么说明这个状态的是不可分的。

·如果出现了不在原来集合中的状态，那么就要根据跳转到的状态的不同，将这个集合拆分。

·如此循环，直到分成的所有状态小组都不可拆分了为止。

·对小组内有多余一个状态的情况，选择其中一个作为这个小组的代表，其他的删掉。

P57例子讲的很好。

P64练习。

第四章 自上而下分析语法

1.主旨：从文法开始符号出发，自上而下的为输入串建立一棵语法树（寻找一个最左推导）。

2.问题：消除左递归；消除回溯；

3.LL1分析法：

·左递归的消除：两个式子，形式化的消除左递归算法。

·消除回溯，提取左因子：

对文法的任何非终结符，用它去匹配输入字符串的时候，能够根据它面临的输入字符串，准确的指定它的唯一一个候选去完成任务。这意味着，如果这个候选无法完成任务，其他所有候选也无法完成任务。这个候选的成败完全代表了这个非终结符。

所以说，一个非终结符面对一个输入字符，必须有唯一确定的解。

对所有非终结符的每个候选a定义first集：

first(a) = 所有左边为a的产生式，它右边的首字符是一个终结符。这些终结符的集合构成first集。如果能推空，空字也加入集合。

·求文法符号X的first集：

如果X是终结符，那么它的first集只有它本身；

如果有产生式X->a……或X->空，那么把a和空字加入first集；

如果有产生式X->Y，那么把Y的first集全部加入到X的first集中。

为了满足要求，必须让一个非终结符的所有候选式的first集两两不相交。

如何满足：提取公共左因子P71

·空字属于first集：

语法树中轮到某一个非终结符A时，它不匹配当前输入，但是这个非终结符可以推导空字，所以可以用空字来匹配。

什么时候可以用空字自动匹配？面临的输入符号a必须得在某个句型中被允许跟在A后面时，才可以允许。所以要查看所有被允许跟在A后面的非终结符的集合follow集。也就是查看所有的句型中是否有连续的……Aa……部分，把这样的a加入A的follow集。如果A出现在某个句型的末尾，那么把结尾符号#也加入follow集（而不是空字）。

·求文法符号A的follow集：

结束符号#加入文法的开始符号S的follow中；

产生式右侧，跟在B后面的语法成分的first集加入B的follow集中。（后面的部分能推导出的first将之替换掉后，相当于紧跟在B后面的部分）

如果有A→……B，那么A的follow集加入B的follow集中（因为A可以完全替换成某某B，那么跟在A后面的就是跟在B后面的）。如果A→……Bβ，但β的first集中包含空字，那么也被允许上面的规则。（这一个很重要，所以一定要看哪些非终结符能推空！！！！）

·综上所述：满足以下条件：

①文法不含左递归（消除左递归公式）

②产生式右边的不同候选first集两两不相交（求first集的三条规则）

③每个非终结符号A，当它的某个候选first集存在空字时（可能出现空字自动匹配的情况），A的first集和follow集无交集（求follow集的三条规则）

那么成为LL1文法，这个文法不带回溯，可以自上而下分析。

·对这种文法，当需要用A进行匹配输入符号a时：

先看A的所有候选first集，包含a就用相应的去匹配；

如果都不包含a，但某个候选first集包含空字，那么再看一步：看看a在不在A的follow集中。如果在，那么A就可以用这个空字去匹配。

以上不满足，文法错误。



4.递归下降分析程序

5.预测分析表：指出了不同非终结符面对不同输入符号的时候应该使用的产生式。

·对每个文法符号构造first集和follow集。

·对任何符号串α=X1X2……Xn，先让α的first集等于X1的first集（空字除外），如果前面的X的first集**包含**空，那么可以把后一个X的first集加入α的first集中。如果每个都包含空，那么把空字也加入α的first集中。

·构造分析表：笔记本上的示例；first集中有的设置为相应的产生式，能推出空字候选式α对follow集中有的设置为A→α。

·分析输入串，写栈，一步一步的画。首先把#入栈，然后把开始符号入栈（毕竟只能从开始符号E开始），根据面对的当前输入字符，查找预测分析表，将栈顶的非终结符替换为表中相应的产生式。

P81练习

第五章：

1.归约：用一个寄存符号的栈，把输入符号依次放到栈中，一旦当栈顶形成某一个候选式时，替换为这个候选式的左部符号（把左部符号移入栈中，和其他的输入符号继续等待归约成下一个候选式）。

·面临多种归约情况的时候，应该选哪一种？精确定义可归约串，也就是最左素短语，或者句柄。

2，规范归约：

格式：S→αAβ（\*推导，可以是本身），A→xxx（+推导，不能是A），那么xxx是A在句型αAβ中的**短语**。如果A→xxx是直接推导，那么是**直接短语**。最左直接短语是**句柄**。

·短语必须在句型中。短语中包含的都是叶节点。素短语必须含有至少一个终结符，并且不含其他素短语。句柄就找最左边的直接短语。一组短语是某个父节点的全部孩子（或隔代，但是必须得往下一直找到叶节点）。

·使用句柄对句子进行归约。

·规范规约：开头是句子，结尾归约到文法的开始符号。每一步都是最右推导的逆过程，并且都是把句柄替换掉（因为它在最左）。每次在栈顶发现句柄的时候就用相应产生式的左部符号把句柄替换掉。柄，你抓住这个句子的这一截。

3.算符优先分析：

·原理：定义算符之间的某种优先关系，借助这种优先关系寻找可归约串并进行归约。

·设法得到不同算符之间的优先关系。

·只要任何产生式的右边都不含有连续两个非终结符（也就是要求其中必须有终结符来连接），就是算符文法。

·三条规则：①a=b:任何产生式中出现ab或aQb的情况。比如(E)，那么（，）平等。

②：a<b：任何产生式中出现aR,，且R→b……或R→Qb……（b在开头）

③：a>b：任何产生式中出现Rb，且R→……a，或R→……aQ（a在末尾）

任意两个终结符都满足这三种规则，那么是一个算符优先文法。

·怎样找出优先关系：首先对每个非终结符P构造以下集合：

firstvt(P)：P→a……或P→Qa……的所有的a

Lastvt(P)：P→……a或P→……aQ。

对所有产生式中的下列两种情况：

①：……aP……：a<firstvt(P)中的所有元素b。

②：……Pb……：last(P)中的所有元素a>b。

构造firstvt集算法：

①：上面的定义

②：传递规则：P→Q……，那么firstvt(Q)的所有元素都属于firstvt(P)。

lastvt完全同理。

4.LR(0)分析法

·前缀：字的任意的前面的部分

·活前缀：在右边增添一些终结符号之后，可以成为一个规范句型。

·项目：在每个产生式的右部添加圆点，圆点可以放在字符串的任意位置，圆点前面的部分代表已经“看见”的部分，也就是前面的产生式推导出的结果已经在输入串中被扫描了出来；后面的部分代表希望从输入串中看到可以由这一部分推出的内容。

·分析过程：

·把文法的各个产生式按照候选项拆开，也就是每个句子右边只有一个候选项。注意在文法的开始符号E之前加一句S’→E。这称为拓广文法，是每次分析的第一步。

·列出所有的项目。

·从第一句S’→·E开始，圆点后面的第一个标识符代表着可以读取到的下一个，找到所有的状态，这一坨状态放到一起作为一组，然后看这一组中面对不同的输入符号（a，b等等）可以把小圆点往哪里移动。形成新的组，以此类推。

·最后得到DFA。圆点在最右边的项目为归约项目（代表可以归约），S’的归约项目代表为接受项目，圆点在终结符前面的为移进项目，圆点在非终结符前面的为待约项目。

·分析表中，（状态，终结符）的值为状态面对这个终结符移进的新状态，用s+状态号表示；（状态，非终结符）的值为状态面对这个非终结符跳转到的新状态，用新状态号表示；如果一个归约项目属于某个状态（也就是有一个圆点在最后的项目），那么这个状态面对所有终结符的值都记为rj，rj是这个项目的产生式的编号。

·用分析表来分析输入串：栈中保存三元式，分别存储状态，符号，输入串

初始三元式（s0, #, a1a2an#）

每一步的结果为（s0s1sm, #X1X2Xm, aiai+1an#）

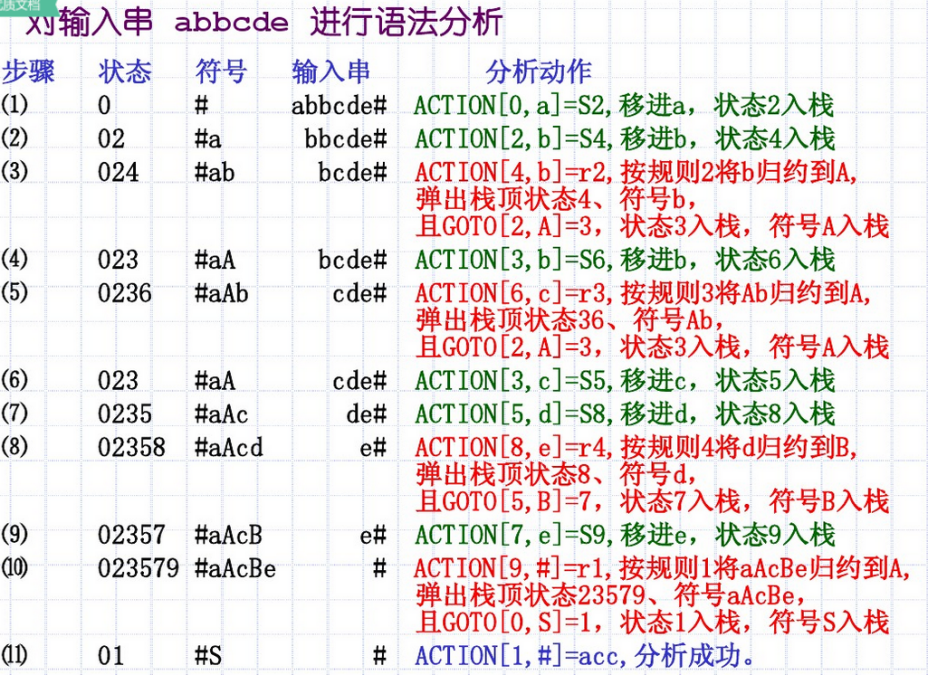
当前，查看表中的（sm, ai）项：

如果为移进，那么移进的状态s放到状态栈顶，ai放到符号栈顶，并从输入串开头删除；s0s1sm s #X1X2Xm ai ai+1an#

如果为归约，那么把符号栈中相应长度的部分替换为产生式左部，并在状态栈中删除相同长度的状态号，替换为前面的状态面临这个产生式左部的时候跳转的状态。

如果未接受，结束。

P151例子



5.SLR(0)

·LR0中的冲突：当一个状态中的不同项目有冲突时，比如含有一个归约项目要求这个状态被归约，又含有另一个移进项目要求这个状态在面对某个终结符时移进这个终结符，那么就会产生冲突。P110例子

·解决方案：比如状态中的归约项目是A→某某某，B→某某某，移进项目是面临终结符b的时候移进。我们考察A和B的follow集，也就是研究一下句型中可能跟在A或B后面的终结符。如果follow AB不相交并且不包含b，那么当这个状态面临某个输入符号a的时候：

a=b,那么移进

a在follow A当中，那么按照A式子归约

a在follow B当中，那么按照B式子归约。

·和LR0的唯一不同在于书写分析表的时候，面临存在归约项目的状态时，找到这个归约项目对应产生式左部A的follow集，把存在于这个follow集中的所有符号a，在分析表的位置（A,a）上写成归约。而不是像LR0中那样一股脑儿的只要看到归约项目就把所有终结符都设置为归约了。

·这样有可能会使得表中的某一个格子有多重定义（也就是follow集之间有交叉或者和移进项目冲突的情况）。如果有那种情况，那么不是一个规范的SLR(1)文法。具有SLR表的文法称为SLR（1）文法，1代表至多需要在分析过程中向前看1个符号。

·面临上面说的这种“多重定义”的时候（一个格子里给出了两种可能的动作），意味着当前的分析没有足够的条件来“**展望**”在这种情况时究竟应该干什么。所以有了LR（1）文法。

**6.LR(1)**

·上面的情况，如果在一个不知该归约还是移进的情况下选择了归约，那么可能存在：归约后的式子和前面已经有的内容未必构成一个活前缀，也就是只能一时痛快，但长远来看不符合要求。

·对应的状况：虽然在SLR中考察了A的follow集，但是并不是follow集中的每个元素在含A的句型中都会出现在相应非终结符的后面。可能只在某一种特定的规范句型中才会出现。比如\*R=i中，=允许出现在R的后面。但R=i中，=不允许出现在R的后面。

·调整项目：由{A→α·β}扩展一部分，变成{A→α·β， a1}。前半部分仍然是LR（0）中熟悉的项目，后面增加的终结符a1称为**向前搜索字符串**（展望串），LR（1）中的1代表了这里串的长度（只需要向前看1位）。这个串只对归约项目有意义。但是其他项目也要写，否则算到归约项目的时候不知道串是啥。在面临归约时，只有当后续的输入串和a1相同时，才允许归约。

·通俗的说：用A→α的时候，要考虑①栈中已有的符号串δα，也要向前扫描一个输入符号a。只有当δAa构成某一个句型的前缀，才说明这种归约是被允许的。

·怎样找展望串：

7.LALR

·LR1会增加很多额外的状态，LALR能把状态数削减到和SLR相同。

第六章 属性文法和语法制导翻译

1.属性文法：在上下文无关文法的基础上给每个符号增加额外的值，用来表示一些仅凭符号无法表示完全的信息（比如类型，值，地址指针等）。

2.语义规则：对文法的每个产生式都配备一组属性的计算规则。

3.综合属性：自下而上传递信息，综合孩子们的信息而来；继承属性：自上而下的传递信息，依据父亲和兄弟的属性继承而来。

4.终结符只有综合属性。非终结符可能都有。开始符号的所有继承属性是整个属性计算体系的初始值。为了利用这些初始值对所有属性进行计算，我们必须有一套规则，来对产生式右边的继承属性和左边的综合属性进行计算。这就是语义规则的一部分。

5.处理属性文法：语法分析后构造语法树，根据需要通过遍历语法树的相关节点来计算各个属性的值（按照语义规则）。包括以下的几种：

·依赖图：描述语法树中各个节点的综合属性和继承属性之间相互依赖的关系。有向图。不考虑循环依赖（不是良定义的）。

属性的计算次序：利用拓扑排序。

·树遍历：深度优先遍历，每次计算出可以计算的属性值。如此循环。

·抽象语法树：操作符和关键字作为具体操作数的父节点。从而去掉中间的非终结符。为每个运算符都建立一个节点mknode(op,left,right)，左右指向两个操作数。mkleaf(id,entry)建立标识符节点，entry指向符号表中的入口。mkleaf(num, ral)建立数节点，ral直接保存值（而不需要利用符号表。于是就明白了实验指导书上的那个部分）。

抽象语法树的语义规则利用这三个函数来规定。

6.S-属性文法自下而上计算：只含有综合属性。

·把属性的值放在状态栈中，一个状态对应一个属性。若干个状态归约之前，对这些属性进行计算。属性和状态是同步的。注意栈顶的移动。（思考实验中的替换栈顶）

7.L-属性文法自上而下翻译：

·含有综合属性和特殊的继承属性：只继承父亲和兄长的属性信息。（也就是不依赖后面的弟弟的属性）。

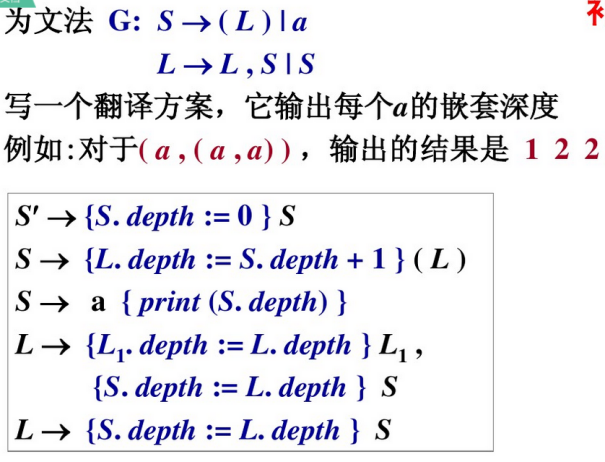
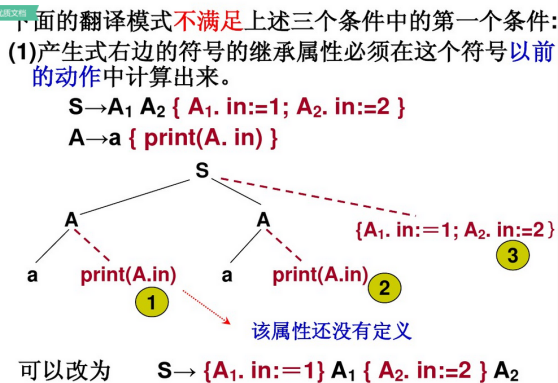
8.翻译模式：把和符号相关的属性和语义规则用{}括起来放到产生式右边的相应位置。这样在对语法分析树进行DFS的时候会成为树的一个节点。来及时的告诉产生式应该对属性做什么操作。它给出了使用语义规则进行计算的次序。

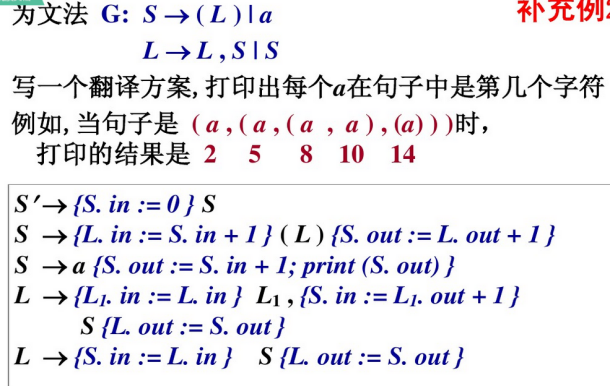
·举例：只需要综合属性时，在每个产生式的末尾写上综合属性的计算规则，如：

T→T+F{T.val = T.val +F.val}

·第二个栗子：综合属性+继承属性：右边符号的继承属性必须在这个符号之前被计算，所以这个计算的式子必须出现在这个符号前面，也就是领先于这个符号；不能使用当前符号的右侧的符号的综合属性；产生式左边非终结符的综合属性必须在它引用的所有属性都被计算的时候才能计算，所以可以放在产生式的最右边（这样会成为这个非终结符的最右边的节点，它左边的节点的属性一定先于这个最右节点被计算），也就是落后于它依赖的所有符号。

·有点意思的两个例子

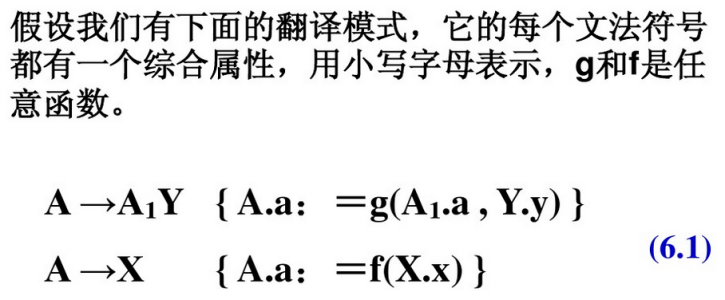


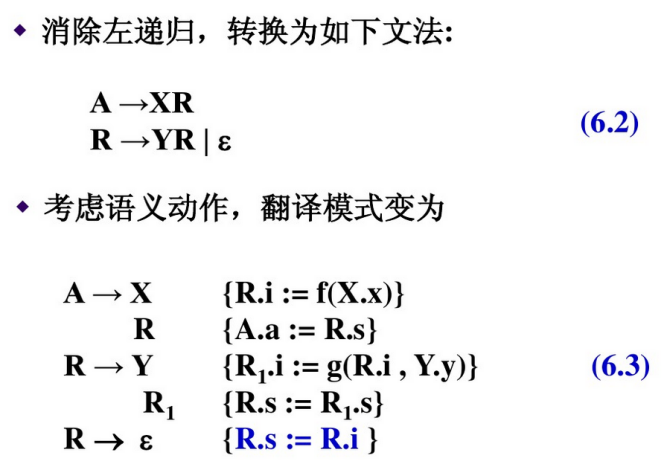


9.自顶向下翻译；带左递归的文法翻译模式为如图。都是综合属性，可以允许自下而上翻译。如果想实现自上而下翻译，就必须消除左递归，然后重新建立一种翻译模式如图。新的翻译模式思路是这样的，R有两个属性i和s。i是继承属性，用来自上而下的计算实际的加法或减法的结果。比如R.i=T.val，由于T总是出现在R的左边，所以在进入R之前已经通过T→i计算出了T的val，这样每一步R.i都可以被计算出来，并和T.val相加，这个就能够计算。怎样把这个值再传递到上面呢，这就需要另一个综合属性R.s来接受下面的值，当R遇到空字不再继续朝下推导的时候就把i的值赋给s，然后在向上返回的过程中就可以把这个s一步一步的返回给R的父亲的R们，最终把值交给E。

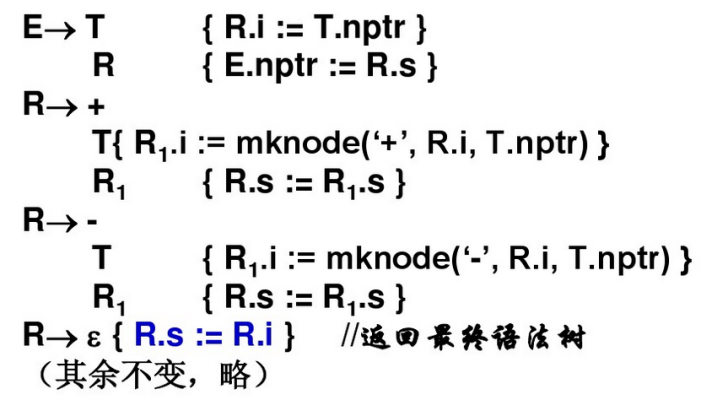
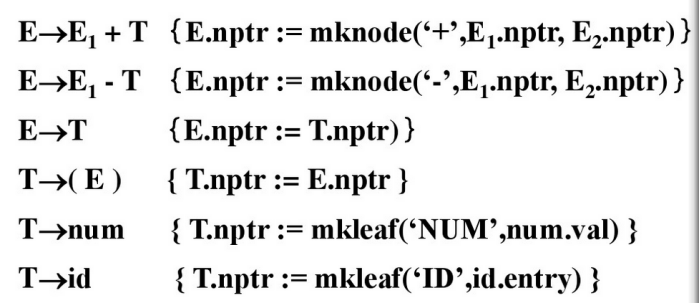


·推广到一般形式。都是当递归属性R不在继续往下生长下去了以后---遇到空字以后---把继承属性i的值交还给综合属性s，由s把结果传递到老大A--开始符号--的属性当中。





·用翻译模式来构造抽象语法树。左图为自下而上未消除左递归，右图消除左递归。



10.自下而上计算继承属性--为啥要这么纠结

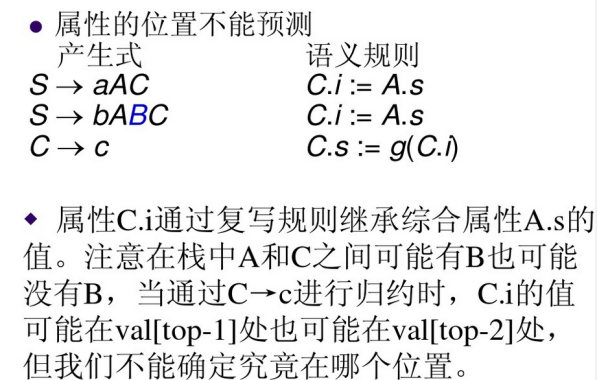
方法--把存在于产生式中间的动作，也就是大括号，替换为不同的非终结符M,N等，然后提供新的产生式M→空字等，把这个动作放在推空的结尾。很机智，就是有点繁琐。

这样一来所有的动作都出现在了某个产生式的结尾处，画成树的时候也就会出现在最右边的叶子节点上。



计算继承属性：这部分有点复杂，先不看了

通过Y.i =X.s来定义继承属性i，i本来应该是自上而下的。但是通过s的加入来自下而上的计算s，用s来定义i，那么要用到i的地方就用s去代替

不能预测属性 在栈中位置的情况：

**第七章 语义分析和中间代码产生**

词法分析——语法分析——静态语义检查和翻译

**1.静态语义检查：**

·类型检查

·控制流检查

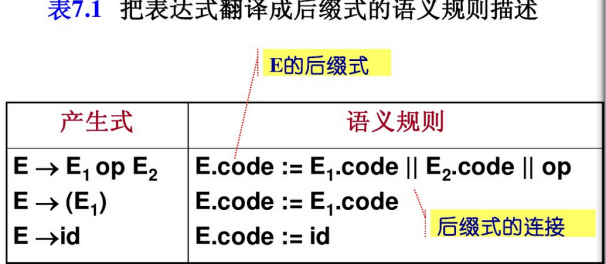
·一致性检查

·相关名字检查

**2.为什么要生成中间代码**：便于进行和机器无关的代码优化工作，可以通过改变后面的中间代码转目标代码程序来转变目标机器，而前端是不变的。

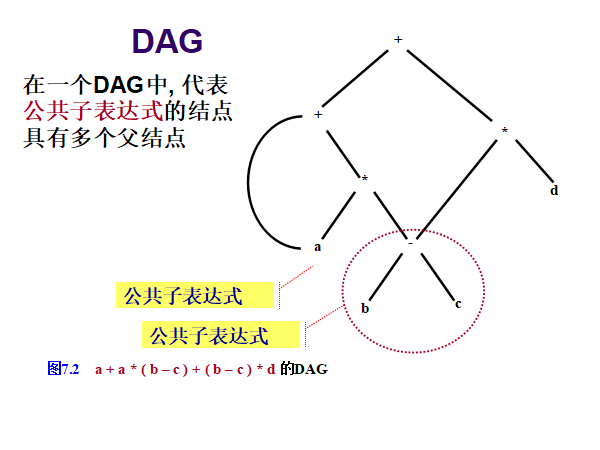
**3.中间代码的形式：后缀式，三地址代码——三元式，四元式，间接三元式，DAG图表示等。**

**4.后缀式**：逆波兰表示法。a+b——ab+；（a+b)\*c——ab+c\*;a\*(b+c)——abc+\*;

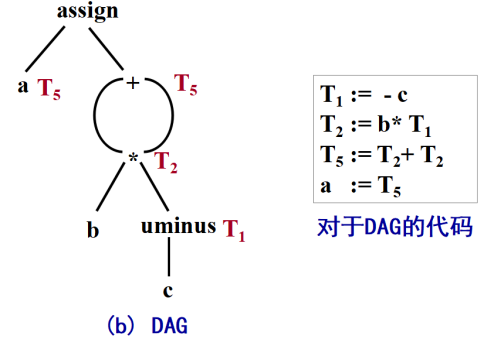


后缀式擅长用于堆栈中。比如ab+，总是两个操作数先入栈，然后遇到运算符+号的时候只需要把栈顶的两个数取出来，进行运算后再存到栈顶就可以了。很优秀！

**5.DAG无循环有向图**。举例如下。可以看出，DAG中同名的节点只有一个，如果被用到多次，那么会从这个节点伸出来多条边（比如a）。

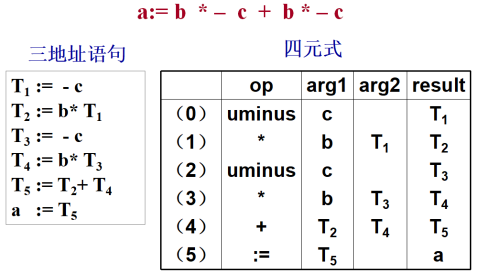


**6.三地址代码**：X:=Y OP Z.它可以看作树或DAG的一种线性表示，也就是可以把一个图转化成三地址代码段的形式。

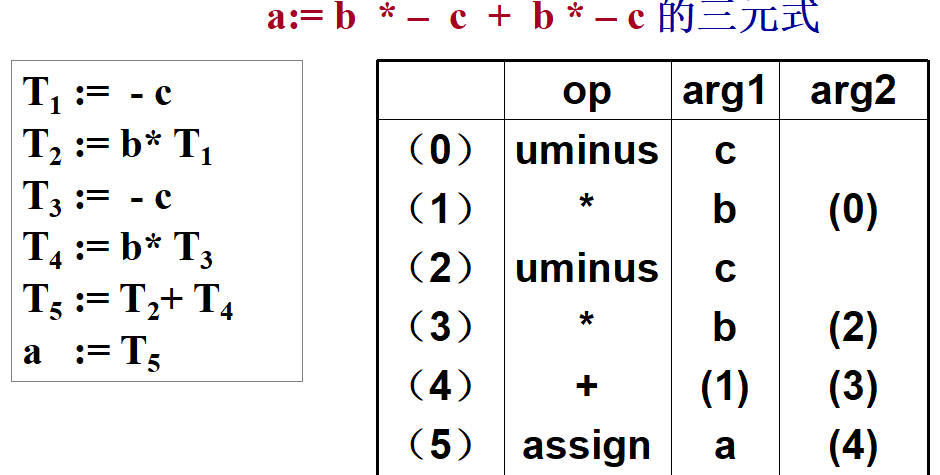


类似的，同样可以规定一个语义规则来产生三地址代码（如同产生后缀式那样）。

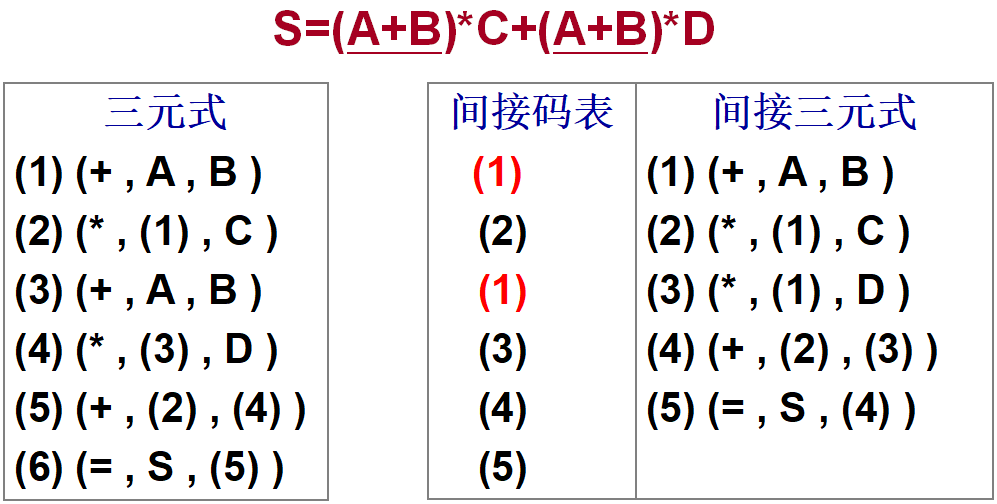
**·四元式：**



**·三元式：**为了避免在符号表中记录临时变量，所以通过语句的位置标号index来表示临时变量。这意味着每个index都有存储空间用来存储这一步的计算结果。



**·间接三元式**：用一个间接码表来引用三元式表中的代码。这样需要调整运算顺序时只需要调整间接码表的顺序，而不用修改三元式。相同的三元式也不需要重复填入三元式表中。



**7.说明语句：用来给局部的名字分配存储空间，比如常量变量过程名。**

·enter(name, type, offset)用来把name加入符号表：offset表示地址（偏移量）。非终结符T的属性type和width表示类型和数据大小。width用来计算下一步的offset。例如在第一个产生式中把offset初始化为0.

·含有嵌套时，对每个过程对应一张符号表。为此在enter方法中要指明将name加入哪一个符号表中。新表要有指针指回定义了这个过程的符号表。

·mktable(pre)：新建一张符号表，pre是它父亲表的指针。

·enter(table, name, type, offset)

·addwidth(table, width)：给table这张表增添一个记录信息width，表示这张表所有内容占用的总宽度。

·enterproc(table, proname, newtable)：在table表中为过程proname建立一个新表newtable。

**8.赋值语句：**

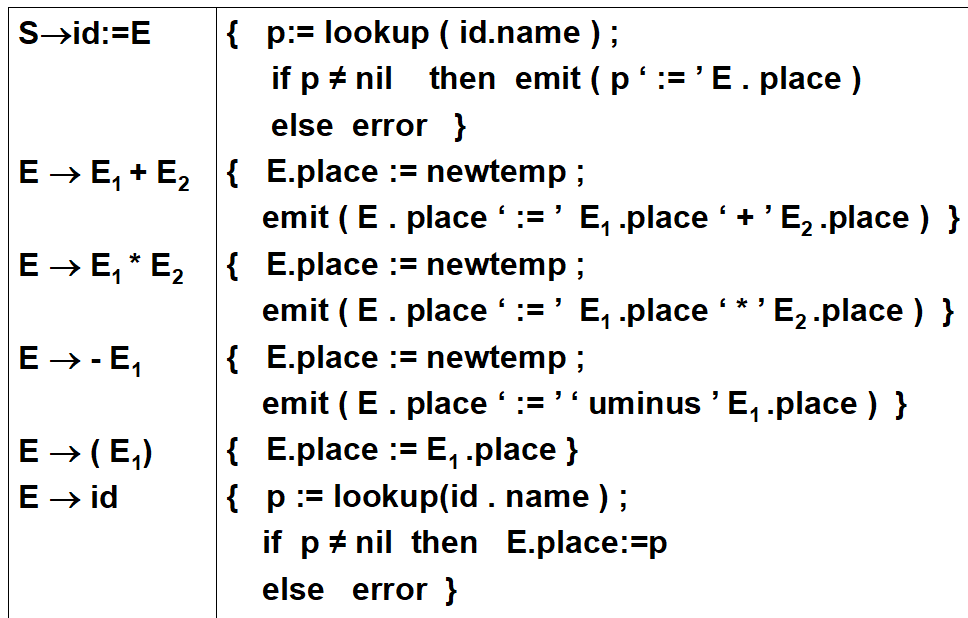
·id.name id的名字；

lookup(id.name)：检查这个名字在不在符号表中。返回指针或nil表示没有。

emit：生成的三地址码发送到输出文件中。

·翻译模式：

**归纳的说，运算的语句就是生成一个新节点newtemp作为临时变量的位置，然后给临时变量E附上运算后的值；关联具体id的语句就是首先检查id在不在符号表中，如果在就对返回的指针进行操作。**



·数组

·一维数组：数组元素放在连续区域内：A[i] = base + (i-low)\*width

整理：A[i] = i\*width + (base-low\*width)这样实现了变量部分和常量部分（括号中的）分离。常量部分可以在数组属性定义出来的时候就计算出来。后面只需要计算i\*width就可以了。

·二维数组：注意按行排列or按列排列。

按行：A[i1, i2] = base + ((i1-low)\*n2 + i2-low2)\*w

整理：A[i1, i2] = ((i1\*n2)+i2)\*w + (base - ((low1\*n2)+low2)\*w)

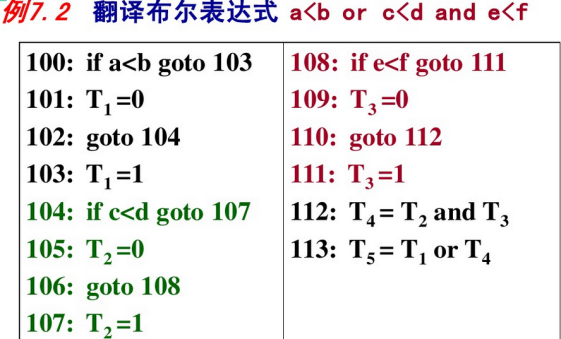
~~这里还没看

9.布尔表达式

·①如同算术表达式，一步一步计算

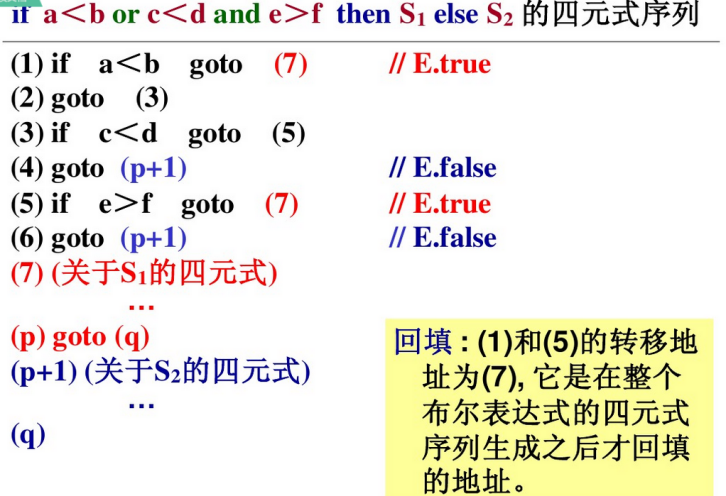
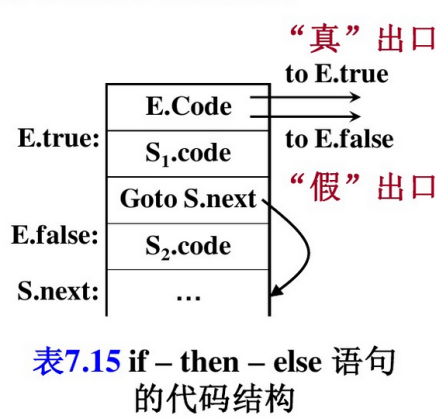
②采取“短路”语法。只计算部分表达式。

·数值表示法：



·作为条件控制的布尔式翻译：如if E then S1 else S2

特殊在于只需要判断进入S1orS2，而不需要保存E的值。



·回填：在生成形式分支的跳转指令时暂时不确定跳转目标，而是建立一个链表，把跳转指令的编号填入链表中。当确定目标后，再把目标填入跳转指令的指针指向的位置。P190翻译模式。

·回填的翻译模式：E的两个属性truelist，falselist。表示在布尔表达式的翻译过程中等待被回填真/假出口的四元式的标号。四元式的格式中，第四位的是目标值。借助这个目标值来实现链。

·例子：比如有pqr三个四元式等待被回填 真 出口。p在最前面，第四位标0；q的第四位标p指向前面的p，r的第四位标q指向前面的q。那么从r出发就能依次找到r,q,p三个需要回填 真 出口的式子。

·用到的函数/变量：

·nextquad：

第九章 运行时存储空间组织

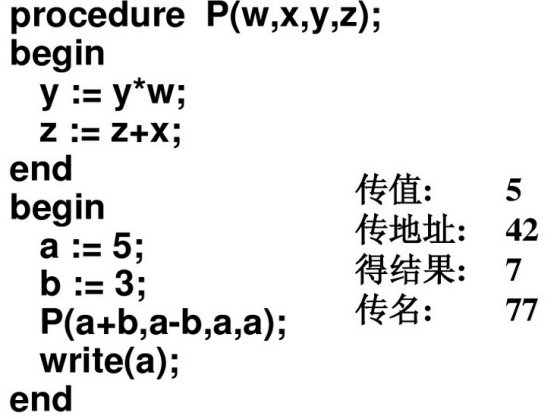
2.传递的种类

·传名 相当于把被调用过程的代码直接抄到调用的地方。

·传地址 先把实参地址放到相应形式参数对应的形式单元中，过程体用到形参的时候就到对应的形式单元中去找地址，然后根据这个地址找值。对值的操作直接改变地址指向地方的操作。调用结束后，形式单元所指的实参单元持有所希望的值。

·传结果 每个形参对应两个形式单元，第一个形式单元存放实参地址，第二个单元存放实参的值。在过程体中用到形参的时候都访问第二个单元。过程返回的时候把第二个单元的内容存放到第一个单元所指出的实参单元中。所以就是说过程没有改变每一个的值，但最后会获得过程调用后的结果。

·传值 就是仅仅使用值，不要求别的

get 学会了

3.运行时存储空间：存放目标代码，存放数据，控制栈

4.C语言的活动记录：用于管理在一次执行中所需要的信息。

·当过程调用时， 产生一个新的活动记录，将记录入栈。

·连接数据：

·返回地址：

·动态链：指向调用这个过程之前的最新的活动记录所在地址的指针

·静态链：用来访问非局部数据，指向定义这个过程的外层的活动记录

·形式单元：存放相应实参的地址或值

·局部数据：局部变量，临时变量等

5.分配策略

·静态：编译时就确定数据空间的大小，并为每个数据项目预留和确定出在运行时在存储空间中的位置；

·栈式动态分配：运行时存储器作为栈进行管理，可用的存储空间总是存在于栈顶，退出后释放栈顶的空间；

·堆式：作为堆进行管理

6.栈式存储分配：

·call P： 准备调用过程P，但还没进入P，这时候要保护现场，以top指针为基地址：1[toP] = SP,保护当前SP;3[top] = n,表示参数个数。然后是跳转指令跳转到P的地方。

·进入过程P之后，新的活动记录中，首先定义新的SP = top+1（这个过程P的SP指针），然后相对SP进行单元分配：0[sp],保存前一个过程的基地址（SP）。1[sp]：保存返回地址，也就是前面call P语句后面的语句；2[sp]为参数个数。同时更新top，top = top + L，L是这个过程P需要的低部分数据单元数（包括老SP，返回值，参数个数，形式单元，局部变量，内情变量），在编译的时候就可以确定，所以可以静态分配。这时候top指向的位置就是动态单元开始分配的地方，也就是临时单元区。

·对任何局部变量或形参X的引用可以表示为变址访问X[sp]，也就是用基地址SP和相对地址来定位。

·过程返回return（E）：首先要把返回值E存放到某个寄存器当中，然后要恢复如下一些参数：top = sp-1,top回到当前活动记录的头部（SP）的前一位；SP = 0[SP]，把SP的值更改为存储在0号地址中的前一个过程的sp的值；X = 2[top]，实际上就是刚才的1号地址，存储返回地址的单元，但是由于sp已经更改了，所以通过top来找到这个位置；然后跳转指令跳转到X的地方。

7.嵌套过程语言的栈式存储，PASCAL的规则

·过程：

过程头；

说明段；（一系列说明语句，量的定义等）

begin:

执行体；（一系列执行语句）

end

·作用域：一个名字（标识符，变量，常量）能被使用的区域范围

·允许相同的标识符在不同的过程中代表不同的东西

8.非局部名字的访问怎么实现？

·定义层次：主程序为0，以此类推。

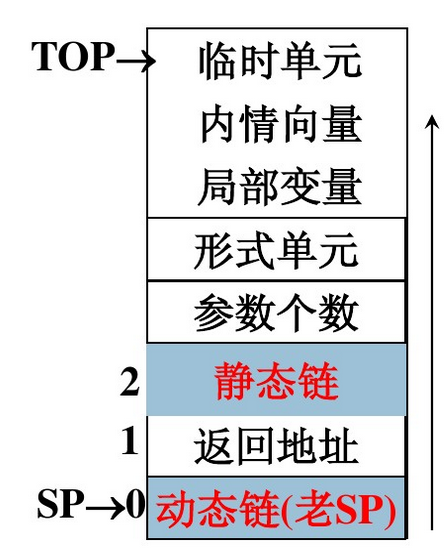
程序运行时，过程Q可以引用它所有外层过程的数据（视为Q的非局部量）。为了达到这个目的，Q必须知道它所有外层过程的当前活动记录的起始地址。所以必须设法跟踪每个外层过程的最新活动记录的位置。通过静态链或显示表（display）。

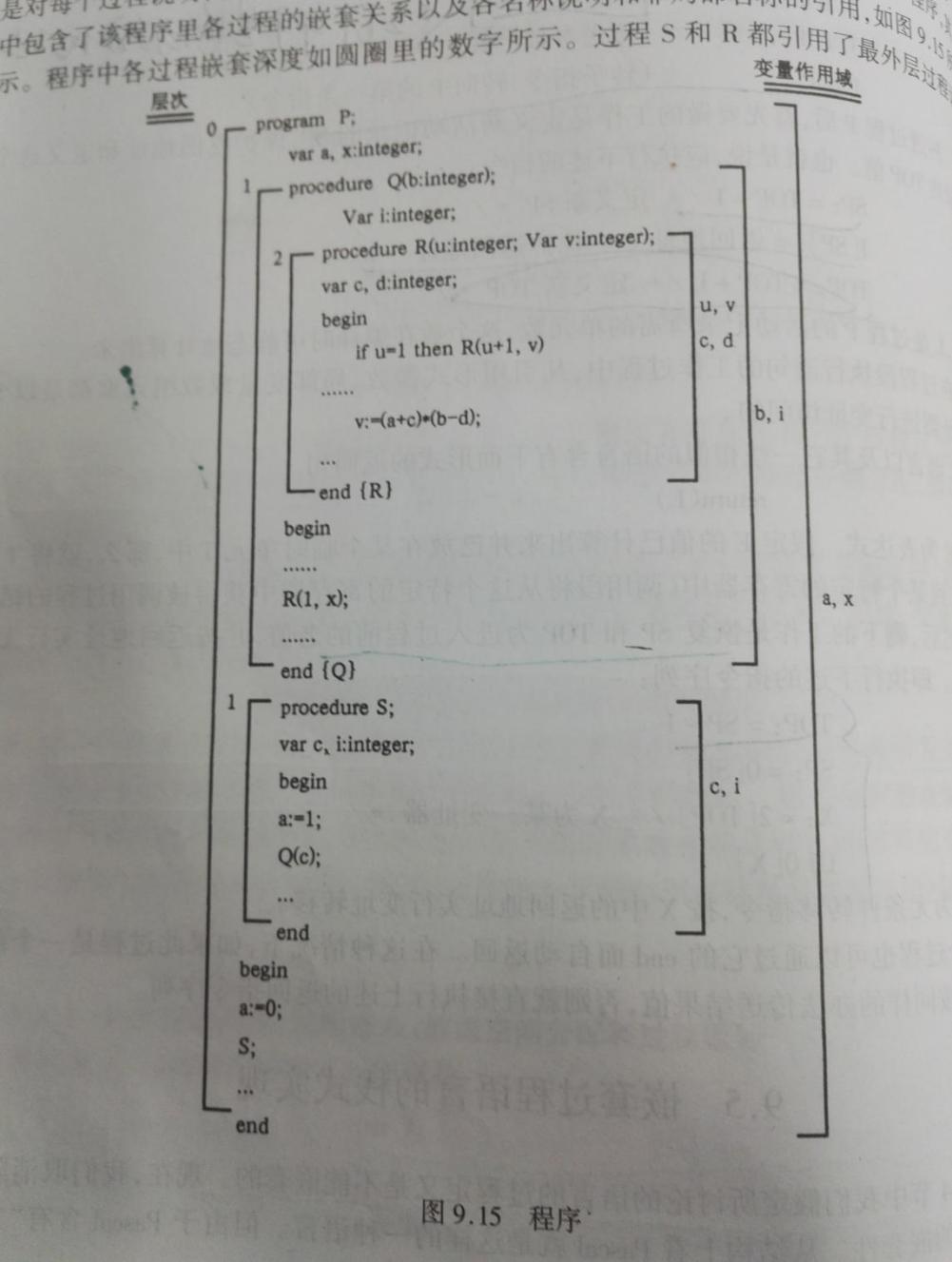
9.静态链方法：

·静态链：指向定义本过程的过程的活动记录的起始地址（也就是物理外层，本过程在程序代码中的外层）。存取链。

·动态链：指向调用本过程的过程的活动记录的起始地址（动态外层，当前运行中过程是从谁哪里跳转过来的）。控制链。

·之前的0[sp]用来存老SP的0号单元，现在用来存动态链；之前的2[sp]用来存静态链。1[sp]不变，存返回地址。如图所示：





·以这个程序为例：P(0)是Q(1)和R(1)的直接外层，Q是R(2)的直接外层。搜索比自己更高的层次中的非局部量，比如R要引用P中的a和Q中的b，那么根据层次差2-0=2和2-1=1去静态链的相应位置上找。

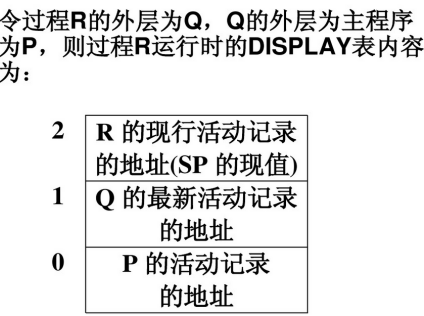
·P260图非常清晰了。不管是静态链还是动态链都是指向前一个过程的活动记录起点SP，只不过对这个“前一个”的定义有所不同。

10.嵌套层次显示表方法

·定义：进入一个过程后，在创建活动记录的同时也创建一个display并把它作为活动记录的一部分。表结构如图。按层从上往下排。换言之，它的display表有i+1个单元（i=层数），自上而下存放着当前层，直接外层……直到最外层（前面例子中的P，也就是主程序）。每个单元都存放着相应层的最新活动记录的基地址。

·由于过程的层数是一定的，所以每个过程的display表的大小是可以在编译时确定的。这样，寻找任意一个过程中定义的变量只需要知道这个变量所在的静态层数和它在相应过程中的偏移量就可以了。

绝对地址=display[静态层数]+相对地址。



以下是三种P1和P2相对层次不同时候的情况：

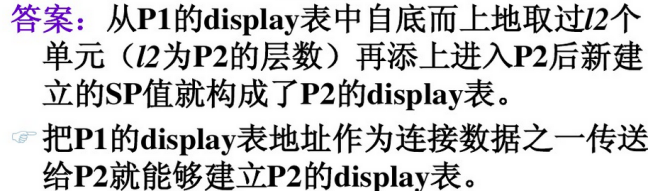
①P1和P2都是1层（L2 = 1），所以从P1的display中取1个单元（P0单元）放到P2的display中，再加上P2的地址。



②P2是2层，所以从P1中取2个单元（P0,P1，都是P2的祖先过程）放到P2的display中，再加上P2的地址。

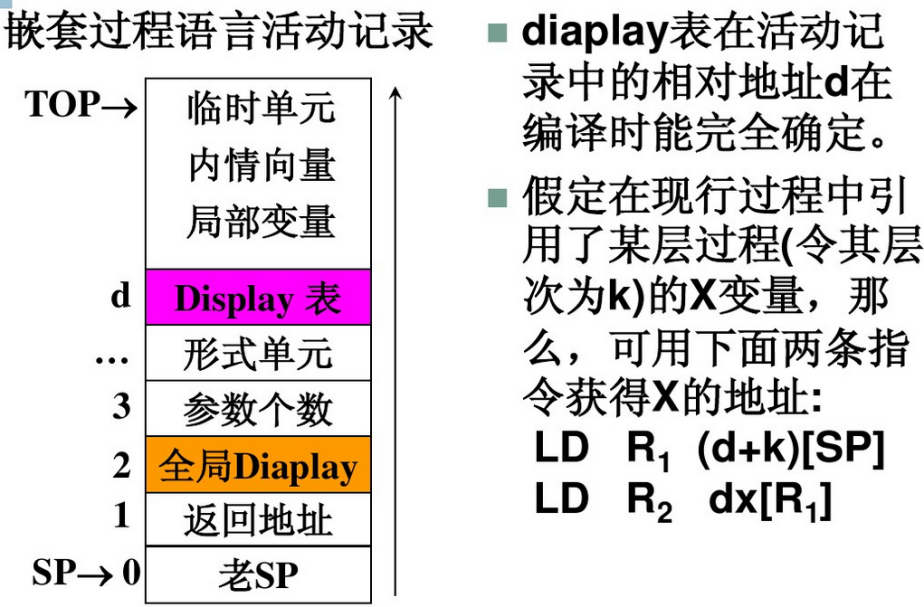


③和第一个差不多。



归纳：如上右图。

·display表在栈中的存储位置：

我只想说设计规则的人这是太牛逼了。

【没有实践就没有感悟。先斩后奏，开头极其困难，但是在后面会融会贯通。比如做机组课设的体验。】

·d[SP]找到当前过程的活动记录中存放display表的位置，(d+k)[sp]进一步借助display表的起点找到对应层次的位置，这个位置上存放的是对应层次SP所在栈中的位置（对应层次的最新活动记录的起点）R1。然后以R1作为基地址，通过dx[R1]找到R1那一段活动记录的相对地址为dx的地方的值，dx就是我们想找的X变量在它被定义的过程中的相对地址。找到的R2就是X的存放位置。

·回忆起实验指导书上，层差间接等于这里的k，offset就是这里的相对地址。融会贯通。一通百通。

·全局display中存储的值指向调用当前过程的过程的display表（当前过程需要借助前一个过程的display表来建立自己的display表）。通过静态层号来决定从前一个display表中取出几个单元来作为自己的display表的下层部分。当前display表新搭建的一层（最高的一层）表示当前过程的入口地址。

11.参数传递的实现

·传简单变量、数组元素或临时变量 par Ti：(i+3)[top]:=Ti 或addr(Ti) 传值和传地址

·如果是传数组，过程或标号那么 par Ti的作用是传地址。

·传数组：

·过程调用：call P，n：

·1[top]:=sp 保护当前sp

·3[top]:=sp+d 传送当前display地址（生成新过程活动记录中的全局display）

·4[top]:=n 传递参数个数

·跳转指令跳转到P的位置。

·转进过程P后，首先定义新的SP=top+1和top=top+L，保存返回地址1[sp]:=sp-1（这里的sp已经是更新过的sp了，要和上面的老sp区分开）。

·根据全局display建立自己的display表：首先根据全局display（2[sp]）单元中存储的值往前一个过程的display表中从下往上取出i个单元（i为层次数），然后在这i个单元的上面添加一个单元保存当前P过程的sp地址，这i+1个单元构成了P的display表。

·过程返回：return

·top = sp-1 返回当前过程活动记录起点的前一位。

·sp = 0[sp] 从0号单元中取出老的SP。

·X = 2[top] 取出返回地址

·跳转到X的位置。

P268练习

第十章 优化 P273开始 记住各种优化规则

1.本章讨论中间代码优化。下一章是目标代码优化。

2.原则：等价，有效，合算

3.范围：局部，循环，全局

4.删除公共子表达式：如果一个表达式E在前面已经计算过并且之后没有改变，那么E成为公共子表达式。消除这些多余的计算。

5.复写传播：不必要的临时变量可以删除。

6.删除无用代码：有些临时变量在经过一系列操作后没有用到，那么久删掉

7.代码外提：有些代码在循环的过程中是不变的，可以放到循环体外面。

8.强度削弱：类似于把乘法运算转化为加减运算。

9.删除归纳变量：

10.局部优化：

·基本块：一段顺序执行的语句序列，只有一个入口和出口，入口是第一个语句，出口是最后一个语句。执行时只能从入口进入，从出口退出。

·划分为基本块的算法：

·求出四元式程序中的各个基本块的入口语句

·对每一个入口语句，构造其所属的基本块

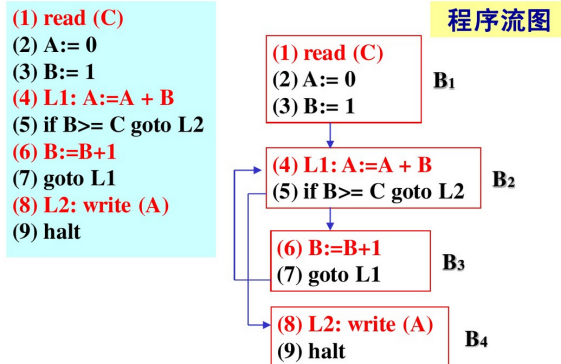
·删除未被纳入任何一个基本块的语句

·入口语句包括

· 程序的第一个语句

·能由条件转移语句和无条件转移语句转移到的目标语句

·紧跟在条件转移语句后面的语句（被条件跳过的部分）



·优化方式：

·删除公共子表达式，删除无用赋值

·合并已知量：把能在编译阶段计算的值计算出来

·临时变量改名：把某一个临时变量改名，来减少临时变量的数量。

·交换语句的位置：

·代数变换：复杂的运算变成相对简单的运算。

乘法变加法千万别忘了！

·流程图

11.基本块的DAG表示及其应用：

叶节点用标识符或常数作为标记，表示该节点的值；jie’di用addr(A)作为标记，表示变量的地址。内部节点用运算符做标记。各个节点可以有多个标记。

·绘制DAG的方法

A=B: 比如T1=3.14，那么3.14写在圆圈下面，T1写在旁边，圆圈内是n

·如果node(B)没有定义，那么建立叶节点，并设置标记B，让node(B)=n是这个节点。

·如果A没有定义，那么A和B共享这个节点。如果A有定义，那么先把A从他原来的节点上删除。

A = op B:

·如果B没有定义，同上；

·如果node（B)是常数，那么直接对

12.循环优化：代码外提，强度削弱，删除归纳变量

·代码外提：在循环入口处建立一个前置节点，外提的代码都放到前置节点处。

第十一章

1.待用信息

待用信息就是某个变量的下一个引用点。活跃信息就是在离开这个基本块以后还要被引用的信息。

·为了获得每个变量在基本块内的待用信息--也就是在哪里这个变量又被引用了，从基本块的出口向前扫描，对每个变量建立相应的待用信息链和活跃变量信息链。

·给符号表中的每个变量都增添两个字段：待用信息栏和活跃信息栏

·开始时，把每个变量都认为是非待用，并根据该变量在基本块出口之后是不是活跃的来填写这个变量的活跃信息栏。如果非待用非活跃，就是(^, ^)。i表示待用信息（下一个引用点是第i条四元式），y表示活跃（yes）。

·在基本块中从后往前读语句，依次处理每一条中间代码。对每一个中间代码i: A:=B op C:

·符号表中A的待用信息和活跃信息附加到i上

·A的待用信息和活跃信息分别置为非待用和非活跃

·B和C的待用信息和活跃信息附加到i上

·B和C的待用信息均置为i，活跃信息均置为y。

所谓附加到i上，就是给中间代码也列一张表，给每一条代码都指出 左值，左操作数，右操作数上面的 当前的 待用信息和活跃信息。



2.寄存器分配

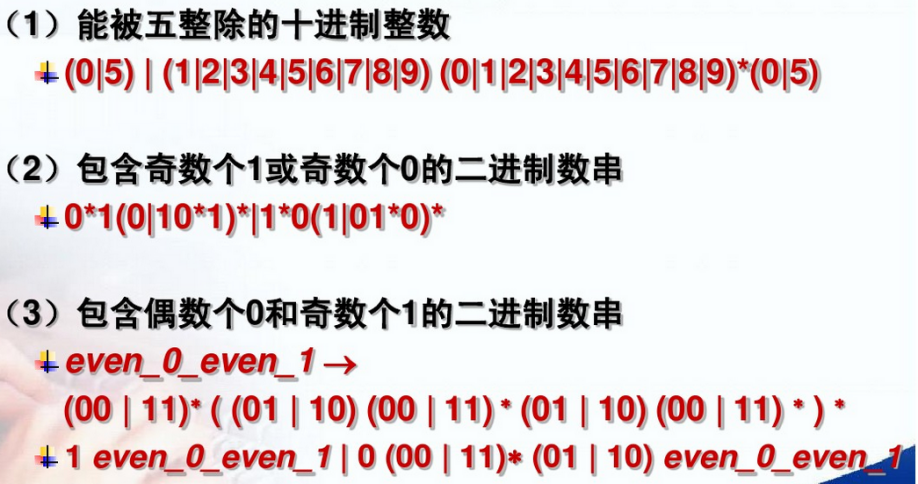
A：=B op C

·先用B所在的寄存器

·如果B不在寄存器中，那么就找一个空闲寄存器

·如果没有空闲寄存器，就选择一个已被占有的寄存器，这个寄存器应该满足：占用这个寄存器的变量已经被存储在内存当中；或者这个变量在基本块中要在最远的将来才会引用或者不会被引用到。

附录 一些题目



·规范归约就是最右推导的逆过程。先写出来最右推导，再倒过来变成规范归约。句柄就是倒过来以后每一步将要被改变的那个。

·消除文法左递归后可能会产生多余规则，要删除多余规则。

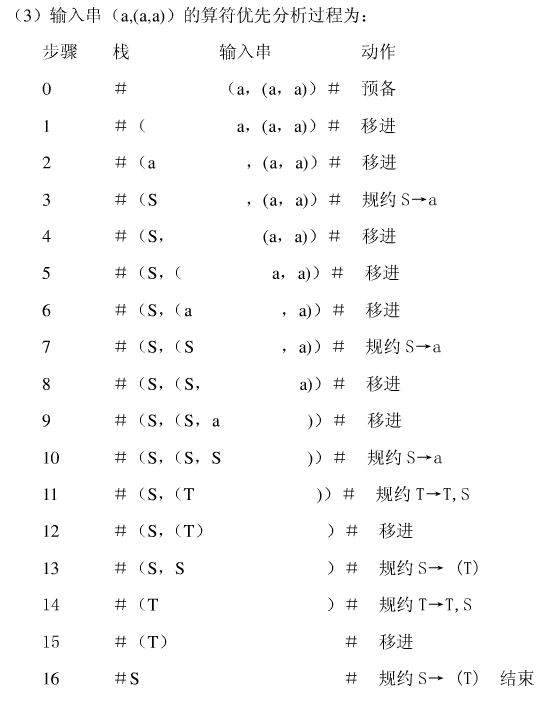
·自上而下分析的问题--某些语言不符合这个语法；消除左递归和提取左因子影响了语法的可读性，造成语义处理的困难。

·移进是把状态和输入符号都压入栈；归约是把对应数量的栈中的状态和输入符号都移出，然后让剩下的栈顶状态面临当前输入符号的时候查看应该移进哪个状态。

·翻译模式的题；先画语法树，最底层是输入串，然后往上归约，一直归约到开始符号S。然后根据规定的翻译模式写出四元式。如果翻译模式规定的生成式中需要添加M，N等标记，那么需要现在输入串中加上这些标记，然后再画语法树。

·算符优先文法；先求所有非终结符的firstvt和lastvt；然后根据前面的小于first集，last集大于后面的来画优先关系表，＃这个符号是和开始符号S的first和last集比较。注意别忘了把相等关系也填上。然后根据优先关系表分析输入串，格式如图。

首先把＃入栈。每次看栈顶的终结符和面临的输入符号的优先关系，**如果栈顶的符号优先级小于输入符号，就把输入符号移进；否则就归约。也就是说优先关系表告诉了我们何时应该停止移进输入符号，而是去查看栈顶现有的若干个元素应该去找一个产生式进行归约了。**



·LR0；

·拓广文法，从0开始编号

·写出所有的项目，从0开始编号

·构建项目集规范族，写出各个项目之间的转移关系

·根据转移关系绘制分析表，分析表的行号是项目号0到i-1，列号左半部分是终结符，表示action，右半部分是非终结符，表示goto。

·遇到非终结符直接写状态号；遇到终结符写s加状态号，表示移进

·某一行项目集中如果含有归约项目--也就是存在小圆点在式子最右边的项目的时候，要进行归约。这一行的项目面对任何终结符的时候都被允许归约。当然，如果有别的移进项目也要写上，这样就产生了冲突。

·如果有冲突，就不是LR0文法。

·在这个基础上判断是不是SLR1文法，先写出来每个非终结符的first集和follow集，面对归约项目的时候求出末位非终结符的follow集，只有这个项目集面对follow集中的非终结符的时候才能进行归约。

·在实际的输入进行分析的时候，画栈区分是自上而下还是自下而上。如果是前者，那么栈里先画的是开始符号，试图推导到输入字符中。如果是后者，那么栈里是每次读入输入字符，最终期望返回到开始符号的形式。

·活前缀；规范句型的一个前缀，这种前缀不含句柄之后的任何符号。只要输入串的已扫描部分可以归约成一个活前缀，那么就意味着扫描过的部分没有错误。