|  |
| --- |
| 第一章 |
| 高级语言程序通常采用解释方式和编译方式两种方式执行：   |  |  | | --- | --- | | 解释方式 | 优点：易于查错  缺点：效率低，运行速度慢 | | 编译方式 | 优点：只需分析和翻译一次，  缺点：在运行中发现的错误必  须查找整个程序确定 | |
| 解释程序：边解释边执行源语言程序，不产生目标语言程序。  编译程序：能够把某种语言的程序转换成另一种语言的程序，而后者与前者在逻辑上是等价的。 |
| |  |  |  |  | | --- | --- | --- | --- | | 解释程序和编译程序的区别 | | | | | 编译  程序 | 源程序的一个转换系统 | 源程序的  目标代码 | 把中间代码转  换成目标程序 | | 解释  程序 | 源程序的一个执行系统 | 源程序的  执行结果 | 执行中间代码 | |
| 编译过程：1. 词法分析 2. 语法分析 3. 语义分析中间代码生成 4. 优化 5. 目标代码生成 |
| 在编译程序使用的表格中最重要的是符号表 |
|  |

|  |
| --- |
| 第二章 高级语言及其语法描述 |
| 程序设计语言的定义 |
|  |
| |  |  | | --- | --- | | 语法 | 表示构成语言句子的各个记号之间的组合规律(构成规则)。 | | 语义 | 表示按照各种表示方法所表示的各个记号的特定含义（各个记号和记号所表示的对象之间的关系） | | 语用 | 表示各个记号或语言词句与其使用之间的关系 | |
| |  |  |  | | --- | --- | --- | |  | 定义 | 描述方式 | | 语义 | 这样的一组规则，使用它可以定义一个程序的意义。 | 基于属性文法的语法制导翻译方法 | | 语法 | 指可以形成和产生合式程序的一组规则。它包括词法规则和语法规则 | 词法规则和语法规则都可以用自然语言、语法图、BNF范式、或文法等描述 | |
| 语法描述方式：1.自然语言 2.语法图(用图解形式描述程序设计语言语法规则的工具)  3.BNF范式 |
| 一个程序语言的基本功能是描述数据和对数据的运算 |
| 高级语言的一般特性 |
| 高级语言的分类：过程式、函数式、基于规则、面向对象  程序结构：单层结构、多层结构 |
| 抽象数据类型：  （1）一个数据对象集，数据对象由一个或多个类型定义；  （2）一个作用于这些数据对象的抽象操作集；  （3）完全封装，用户除了能使用该类型的操作来处理这类数据对象之外，不能作其他的处理。  抽象数据类型有两个重要特征：信息隐蔽和数据封装，使用与实现相分离。 |
| 数据抽象的优点：程序组织和修改、可读性、可靠性、可维护性 |
| 简单语句：说明语句、赋值语句、控制语句、过程调用语句  语句：简单语句、复合语句 |
| 表达式：一个表达式是由运算量（亦称操作数，即数据引用或函数调用）和算符组成的 |
| 语句分类：从功能上说语句大体可分执行性语句和说明性语句两大类：  1.说明性语句旨在定义不同数据类型的变量或运算。  2.执行性语句旨在描述程序的动作。  2.1执行性语句又可分为赋值语句、控制语句和输入/输出语句  从形式上说，语句可分为简单句、复合句和分程序等。 |
| 说明语句：说明语句旨在定义名字的性质 |
| 程序语言的语法描述 |
| 自然语言、形式语言  形式语言是某个字母表上的字符串的集合，有一定的描述范围。 |
| 基本概念：  字母表：元素的非空有穷集合  符号：字母表中的元素  符号串：字母表中的符号组成的任何有穷序列  符号串的长度：符号串x中符号的数目, 用|x|表示  空符号串（空字）：不包含任何符号的符号串，记为ε。  子符号串：设有非空符号串u=xvy，则称v为符号串u的子符号串  前缀：符号串左部的任意子串，称为符号串的前缀；  后缀：符号串右部的任意子串，称为符号串的后缀。  符号串集合：若集合中所有元素都是某字母表上的符号串，则称之为该字母表上的符号串集合  用\*表示上的所有符号串的全体，空字也包括在其中，Ø 表示不含任何元素的空集{ }，注意区分：Ø 、ε、{ε}  连接积：    一般UV≠VU，但（UV)W=U(VW). |
| 文法  所谓文法是用来定义语言的一个数学模型。  表示语言的方法：  若语言L是有限集合，可用列举法  若L是无限集合（集合中的每个元素有限长度），用其他方法。  方法一：文法产生系统，由定义的文法规则产生出语言的每个句子  方法二：机器识别系统：当一个字符串能被一个语言的识别系统接受，则这个字符串是该语言的一个句子，否则不属于该语言。 |
| 文法的形式化定义： |
| 非终结符号：需要进一步定义的符号，不会出现在程序中。(用来代表语法范畴、一个非终结符代表一个一定的语法概念，因此非终结符是一个类（或集合）记号，而不是个体记号)  终结符号：不需要再定义，会出现在程序中(组成语言的基本符号，即在程序语言中的单词符号，如标识符，常数，算符和界符等)  开始符号S至少在某个产生式的左部出现一次.  产生式（规则）：定义语法单位的一种书写规则:  1.它的形式为：P→ α （或P::= α）  2.若干个左部相同的产生式如 P→α1 , P→α2 , P→α3 , ... , P→αn  可合并成一个,缩写为  P→α1|α2|α3|...|αn ,  其中,每个αi 称为是P的一个候选式. |
| 直接推导：从识别符号开始，把当前产生的符号串中的非终结符号替换为相应产生式右部的符号串，直到最终全由终结符号组成。这种替换过程称为推导或产生句子的过程，每一步称为直接推导或直接产生  归约是推导的逆过程    若在推导关系中，每次最先替换最左（右）的非终结符，则称为最左（右）推导；  若在归约过程中，每次最先归约最左（右）的非终结符，则称为最左（右）归约。 |
| 句型：假定G是一个文法，S是它的开始符号，如果S  α，则称α是文法G的一个句型  句子：仅由终结符组成的句型称为句子  语言：文法G所产生句子的全体 |
| Chomsky文法体系分类: |
| 3型文法:正则文法： |
| 2型文法(上下文无关文法)CFG: |
| 1型文法:CSG上下文有关文法： |
| 0型文法：无限制文法，短语文法  对应的语言：递归可枚举语言  与图灵机等价。 |
|  |
| 语法树是表示一个句型推导过程的图，是一棵倒立的树：  结点、边、根结点、末端结点、末端分支、兄弟结点  从推导构造语法树方法：把开始符号S做为语法树的根结点，对每一个直接推导画一次结点的扩展，该结点是直接推导中被替换的非终结符号，其所有儿子结点所组成的符号串是推导中所用产生式右部。直到推出或句型或句子或无法再推导时结束    同一句型（句子）可以通过不同的推导序列推导出来，所以一个句型不是只有唯一的一个推导  不同的最右（左）推导对应不同的语法树，所以一个句型不是只对应唯一的一棵语法树 |
| 二义性：  如果一个文法的句子（句型）存在两棵语法树,那么,该句子是二义性的。  如果一个文法包含二义性的句子,则称这个文法是二义性的; 否则，该文法是无二义性的。 |
| 如语言L找到一个文法是无二义的，则L是无二义的；  如未找到一个文法是无二义的，也不能断定L二义。  产生某上下文无关语言的每一个文法都是二义的，则称此语言是先天二义的.  文法的二义性是不可判定的。（因为文法的二义性由句子的语法树决定，不可能对无穷句子来判别） |

|  |
| --- |
| 第三章 词法分析 |
| 词法分析程序的实现方式：完全独立方式、相对独立方式 |
| 状态转换图：  转换图：是一个有限方向图。  结点代表状态，用圆圈表示。  初态：一张转换图的启动条件，通常有一个,用圆圈表示。  终态：一张转换图的结束条件，至少有一个，用双圈表示。  状态之间用方向弧连接。弧上的标记（字符）代表在出射结点状态下可能出现的输入字符或字符类。  状态转换图中只包含有限个状态（结点） |
| 一个状态转换图可用于接受（或识别）一定的符号串。  路:在状态转换图中从初始状态到某一终止状态的弧上的标记序列。  对于某一符号串β，在状态转换图中，若存在一条路产生β，则称状态转换图接受（或识别）该符号串β，否则称符号串β不能被接受。 |
| 能被状态转换图TG接受的符号串的集合记为L(TG)，称它为状态转换图所能识别的语言。 |
| 状态转换图识别单词符号的过程  Step1. 从初态开始；  Step2. 从输入串中读一个字符；  Step3. 判明读入字符与从当前状态出发的哪条弧上 的标记相匹配，便转到相应匹配的那条弧所指向的状态；  Step4. 重复Step3，均不匹配时便告失败；到达终态时便识别出一个单词符号。 |
|  |
| 从正规式构造等价的NFA ：Thompson算法 |
|  |
|  |

|  |
| --- |
| 第四章 语法分析-自上而下分析 |
| 具体见PPT,详细过程例子在PPT |
|  |

|  |
| --- |
| 第五章 语法分析-自下而上分析 |
| 具体见PP,详细过程例子在PPT |
|  |

|  |
| --- |
| 第六章 属性文法和语法制导翻译 |
| 属性：在上下文无关文法的基础上，在描述语义动作时，为每个文法符号（终结符和非终结符）配备若干相关的“值”，如“类型”，“地址”等，称为属性 |
| 语义规则：对文法的每个产生式配备一组属性计算规则称为语义规则，它的描述形式为b:=f(c1,c2,…ck)，其中b,c1,c2…ck为文法符号的属性，f是一个函数 |
| 属性文法：每个文法符号联系于一组属性，且对每个产生式都给出其语义规则的文法称为属性文法 |
|  |
| 综合属性：  1.在语法树中，一个结点的综合属性的值由其子结点的属性值确定。  2.使用自底向上的方法在每一个结点处使用语义规则计算综合属性的值  3.仅仅使用综合属性的属性文法称S－属性文法 |
| 继承属性：  1.在语法树中，一个结点的继承属性由此结点的父结点和/或兄弟结点的某些属性确定  2.用继承属性来表示程序设计语言结构中的上下文依赖关系很方便 |
| 1.终结符只有综合属性，由词法分析器提供  2.非终结符既可有综合属性也可有继承属性，文法开始符号的所有继承属性作为属性计算前的初始值  3.对出现在产生式右边的继承属性和出现在产生式左边的综合属性都必须提供一个计算规则。属性计算规则中只能使用相应产生式中的文法符号的属性  4.出现在产生式左边的继承属性和出现在产生式右边的综合属性不由所给的产生式的属性计算规则进行计算，它们由其它产生式的属性规则计算或者由属性计算器的参数提供  5.语义规则所描述的工作可以包括属性计算、静态语义检查、符号表操作、代码生成等等。 |
| 基于属性文法的处理过程：输入串-------->语法树-------->语法制导------>语义规则计算次序 |
| 由源程序的语法结构所驱动的处理办法就是语法制导翻译法,  包含1.依赖图 2.树遍历 3.一遍扫描 |
| 1.一条求值规则只有在其各变元值均已求得的情况下才可以使用；  2.如果一属性文法不存在属性之间的循环依赖关系，那么称该文法为良定义的  3.对于一个有向无环的依赖图,可以通过拓扑排序,得到计算语义规则的顺序。 |
| 树遍历的属性计算方法：  通过树遍历的方法计算属性的值  假设语法树已建立，且树中已带有开始符号的继承属性和终结符的综合属性  以某种次序遍历语法树，直至计算出所有属性  深度优先，从左到右的遍历 |
| 一遍扫描的处理方法：  一遍扫描的处理方法是在语法分析的同时计算属性值  所采用的语法分析方法  属性的计算次序  S－属性文法适合于一遍扫描的自下而上分析 |
| 语法制导翻译：  1.直观上说就是为每个产生式配上一个翻译子程序（称语义动作或语义子程序），并且在语法分析的同时执行它。  2.语义动作一方面规定了产生式产生的符号串的意义，另一方面又按照这种意义规定了生成中间代码应做的基本动作。  3.定义：在语法分析过程中，当一个产生式获得匹配（自上而下分析）和用于归约（自下而上分析）时，此产生式对应的语义子程序进入工作，完成既定翻译任务，产生中间代码。 |
| 语法制导翻译的作用：  1.如果语义动作不是简单的计值程序，而是某种中间代码的产生程序，那么随着语法分析的进展，这种代码也逐步生成。  2.语法制导翻译的作用  2.1产生中间代码，  2.2产生目标指令，  2.3对输入串进行解释执行。 |
| 非语法制导翻译方法：与语法制导翻译方法相对的是非语法制导翻译，即翻译程序将不受输入语言的文法的控制 |

|  |
| --- |
| 第七章 语义分析和中间代码生成 |
| 1.说明语句的翻译  2.赋值语句的翻译  3.布尔表达式的翻译  4.两遍扫描条件控制及其语句的翻译  5.一遍扫描条件控制及其语句的翻译  6.过程调用的处理 |
| 常用的中间语言  1.后缀式，逆波兰表示  2.图表示  2.1.DAG  2.2.抽象语法树  3.三地址代码  3.1.三元式  3.2.四元式  3.3.间接三元式 |
| 后缀式：  后缀式表示法：Lukasiewicz发明的一种表示表达式的方法，又称逆波兰表示法。  一个表达式E的后缀形式可以如下定义：  1. 如果E是一个变量或常量，则E的后缀式是E自身。  2. 如果E是E1 op E2形式的表达式，其中op是任何二元操作符，则E的后缀式为E1 E2 op，其中E1 和E2 分别为E1 和E2的后缀式。  3. 如果E是(E1)形式的表达式，则E1的后缀式就是E的后缀式。  逆波兰表示法不用括号。只要知道每个算符的目数，对于后缀式，不论从哪一端进行扫描，都能对它进行唯一分解。  后缀式的计算：  1.用一个栈实现。  2.一般的计算过程是：自左至右扫描后缀式，每碰到运算量就把它推进栈。每碰到k目运算符就把它作用于栈顶的k个项，并用运算结果代替这k个项。 |
| 抽象语法树：在语法树中去掉那些对翻译不必要的信息，从而获得更有效的源程序中间表示。这种经变换后的语法树称之为抽象语法树 |
| DAG:有向无循环图：  1.对表达式中的每个子表达式，DAG中都有一个结点  2.一个内部结点代表一个操作符，它的孩子代表操作数  3.在一个DAG中代表公共子表达式的结点具有多个父结点 |
| 三地址语句的种类：  x:=y op z  x:=op y  x:=y  goto L  if x relop y goto L或if a goto L  param x和call p,n，以及返回语句return y  x:=y[i]及x[i]:=y的索引赋值  x:=&y, x:=\*y和\*x:=y的地址和指针赋值 |
| 四元式：  1.一个带有四个域的记录结构，这四个域分别称为op, arg1, arg2及result  2.四元式之间的联系通过临时变量实现。  3.单目运算只用arg1域，转移语句将目标标号放入result域。  4.arg1,arg2,result通常为指针，指向有关名字的符号表入口，且临时变量填入符号表。  三元式:  1.通过计算临时变量值的语句的位置来引用这个临时变量  2.三个域：op、arg1和arg2 |
| 四元式、三元式和间接三元式比较:  1.三元式中使用了指向三元式的指针，优化时修改较难。  2.间接三元式优化只需要更改间接码表，并节省三元式表存储空间。  3.修改四元式表也较容易，只是临时变量要填入符号表，占据一定存储空间。 |
| 常量定义的翻译:  对每个常量定义的处理应包括下列工作：  （1）把等号右边的常量值登录入常量表中（若之前尚未登录）并回送常量表序号；  （2）然后为等号左边的标识符建立符号表新条目，在该条目中填入常量标志、相应类型及常量表序号。 |
| 简单算术表达式及赋值语句：  简单算术表达式及赋值语句翻译为三地址代码的翻译模式  属性id.name 表示id所代表的名字本身  过程lookup(id.name)检查是否在符号表中存在相应此名字的入口。如果有，则返回一个指向该表项的指针，否则，返回nil表示没有找到  过程emit将生成的三地址语句发送到输出文件中 |
| 布尔表达式翻译：  布尔表达式：用布尔运算符把布尔量、关系表达式联结起来的式子。  布尔运算符：and, or, not ；  关系运算符 ＜,≤,＝, ≠，＞ ,≥  布尔表达式的两个基本作用:  用于逻辑演算，计算逻辑值;  用于控制语句的条件式.  产生布尔表达式的文法:  E→E or E | E andE |  E | (E) | id rop id | id  运算符优先级：布尔运算由高到低:not and or，同级左结合，关系运算符同级，且高于布尔运算符 |
| 计算布尔表达式通常采用两种方法:  （1） 如同计算算术表达式一样,一步步算  1 or (not 0 and 0) or 0  =1 or (1 and 0) or 0  =1 or 0 or 0  =1 or 0  =1  （2） 采用优化措施  把A or B解释成 if A then true else B  把A and B解释成 if A then B else false  把 A解释成 if A then false else true  对应的，有两种布尔表达式翻译方法：  1.数值表示法：  1.1过程emit将三地址代码送到输出文件中  1.2Nextstat：给出输出序列中下一条三地址语句的地址索引  1.3每产生一条三地址语句后，过程emit便把nextstat加1  2.作为条件控制的布尔式翻译（包含两边扫描和一遍扫描）  两边扫描：  1.为给定的输入串构造一棵语法树；  2.对语法树进行深度优先遍历，进行语义规则中规定的翻译。 |

|  |
| --- |
| 第九章 运行时存储空间组织 |
| 1.目标程序运行时的活动 |
| 过程的活动：  1.过程定义：过程名＋过程体  2.过程调用：过程名出现在可执行语句中。  3.过程的活动：过程的一次执行。  4.活动的生存期：执行过程体第一步操作到最后一步操作之间的操作序列，包括该过程调用其它过程花费的时间。  说明：  1.每次控制从过程P进入过程Q后，如果没有错误，最后都返回到过程P。  1.1对于非递归的两个过程，如果a和b都是过程的活动，那么它们的生存期或者是不重叠的，或者是嵌套的。  2.如果一个过程是递归的，那么在某一时刻可能有这个过程的几个活动活跃着。  3.编译程序组织存储空间时，要考虑激活一个过程的新活动时或从过程的活动返回时，对局部名的处理，及是否允许静态或动态存储分配等。 |
| 两种程序体结构：1.单层(并列)结构 2.多层(嵌套)结构(过程可以嵌套和递归) |
| 作用域:  1.作用域：一个名字能被使用的区域范围称作这个名字的作用域。  2.允许同一个标识符在不同的过程中代表不同的名字。  3.名字作用域规则—“最近嵌套原则”  3.1一个在子程序B1中说明的名字X只在B1中有效（局部于B1）；  3.2如果B2是B1的一个内层子程序且B2中对标识符X没有新的说明，则原来的名字X在B2中仍然有效。如果B2对X重新作了说明，那么，B2对X的任何引用都是指重新说明过的这个X。 |
| 参数传递：调用和被调用过程之间的信息是通过参数（或全局量）来传递的。称之为形式参数与实在参数。  包含四种方式：1.传地址 2.得结果 3.传值 4.传名   |  |  |  | | --- | --- | --- | |  | 定义 | 方法 | | 传地址 | 把实在参数的地址传递给相应的形式参数，形式参数在过程段有相应的形式单元存放相应的实在参数的地址。 | 1.调用段预先把实在参数的地址传递到被调用段可以拿到的地方；  2.程序控制转入被调用段之后，被调用段首先把实在参数的地址抄进自己相应的形式单元中；  3.过程体对形式参数的引用或赋值被处理成对形式单元的间接访问。 | | 得结果 | 传地址的一种变形，但不等价 | 1.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。  2.在过程体中对形式参数的任何引用或赋值都看作对它的第二个单元的直接访问。  3.过程完成返回前把第二个单元的内容存放到第一个单元所指的实参单元中。 | | 传值 | 把实在参数的值传递给相应的形式参数 | 1.调用段预先把实在参数的的值计算出来并放在被调用段可以拿到的地方；  2.被调用段开始工作时，首先把实参的值抄入形式参数相应的单元；  3.被调用段中，象引用局部数据一样引用形式单元。 | | 传名 | 1.过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把其中任一出现的形式参数都替换成相应的实参。  2.如果在替换时发现过程体中的局部名和实在参数中的名字使用相同的标识符,则必须用不同标识符表示这些局部名 | 在进入被调用段的之前不对实在参数预先进行计值，而是让过程体中每当使用到相应的形式参数时才逐次对它实行计值（或计算地址）。因此，通常把实在参数处理成一个子程序（称为参数子程序），每当过程体中使用到相应的形式参数时就调用这个子程序。 | |
| 2.运行时存储器的划分 |
| 一个目标程序运行所需的存储空间包括:  1.存放目标代码的空间  2.存放数据项目的空间  3.存放程序运行的控制或连接数据所需单元（控制栈） |
| 活动记录：为了管理一个过程活动所需的信息，使用一个连续的存储块存放这些信息，这个连续存储块称为活动记录    1返回地址  2动态链：指向调用该过程前的最新活动记录地址的指针。  3静态链：指向静态直接外层最新活动记录地址的指针，用来访问非局部数据。  4形式单元：存放相应的实在参数的地址或值。  5局部数据区：局部变量、内情向量、临时工作单元（如存放对表达式求值的结果）。 |
| 存储分配策略：  1.静态分配策略(FORTRAN)： 如果在编译时能确定数据空间的大小，则可采用静态分配方法。在编译时刻为每个数据对象分配固定的存储单元，且在运行时始终保持不变。  2.动态分配策略(PASCAL) ：如果在编译时不能确定运行时数据空间的大小，则必须采用动态分配方法。允许递归过程和动态申请释放内存。  2.1栈式动态分配  2.2堆式动态分配 |
| 3.静态存储分配 |
| 整个程序所需空间在编译时可确定, 可采用静态分配策略 |
|  |
| 4.简单的栈式存储分配 |
|  |
| 5.嵌套过程语言的栈式实现 |
| 非局部名字的访问的实现：  主程序的层次为0；在i层中定义的过程，其层次为i+1;  (层数， 偏移量)  过程运行时，必须知道其所有外层过程的当前活动记录的起始地址。  嵌套层次显示表display  静态链和活动记录 |
| 嵌套层次显示表display：  1.当进入一个过程后，在建立其活动记录区的同时建立一张嵌套层次显示表diaplay，把diaplay表作为活动记录的一部分。  2.令过程R的外层为Q，Q的外层为主程序为P，则过程R运行时的DISPLAY表内容为：   |  | | --- | | R的现行活动记录的地址(SP的现值) | | Q的最新活动记录的地址 | | P的活动记录的地址 | |
| 当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？  答：从P1的display表中自底而上地取过l2个单元（l2为P2的层数）再添上进入P2后新建立的SP值就构成了P2的display表。 |
| 当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？  答：从P1的display表中自底而上地取过l2个单元（l2为P2的层数）再添上进入P2后新建立的SP值就构成了P2的display表。 |
| 当过程P1调用过程P2而进入P2后，P2应如何建立起自己的display表？  答：从P1的display表中自底而上地取过l2个单元（l2为P2的层数）再添上进入P2后新建立的SP值就构成了P2的display表。 |
| 过程调用、过程进入、过程返回：   1. 每个par Ti(i=1,2,…n)可直接翻译成如下指令:   (i+4)[TOP]:= Ti (传值)  (i+4)[TOP]:=addr(Ti ) (传地址)   1. call P，n 被翻译成:   1[TOP]:=SP (保护现行SP)  3[TOP]:=2[SP] (传送现行display地址)  4[TOP]:=n (传递参数个数)  JSR (转子指令)  3. 转进过程P后，首先定义新的SP和TOP，保存返回地址。  4. 根据"全局display"建立现行过程的display：从全局display表中自底向上地取l个单元，在添上进入P后新建立的SP值就构成了P的display。  5. 过程返回时，执行下述指令:  TOP:=SP-1  SP:=0[SP]  X:=2[TOP]  UJ 0[X] |
| 静态链：指向本过程的直接外层过程的活动记录的起始地址，也称存取链。  动态链：指向本过程的调用过程的活动记录的起始地址，也称控制链。 |

|  |
| --- |
| 第十章 优化 |
| 优化：对程序进行各种等价变换，使得从变换后的程序出发，能生成更有效的目标代码。 |
| 三个优化原则：  优化的目的是为了产生更高效的代码。由优化编译程序提供的对代码的各种变换必须遵循一定的原则：  1.等价原则：经过优化后不应改变程序运行的结果；  2.有效原则：使优化后所产生的目标代码运行时间较短，占用的存储空间较小；  3.合算原则：应尽可能以较低的代价取得较好的优化效果。 |
| 优化的三个不同级别：  1.局部优化 2.循环优化 3.全局优化  优化的种类：  1.删除多余运算(或称删除公用子表达式) 2.代码外提  3.强度消弱 4.变换循环控制条件  5.合并已知量 6.复写传播  7.删除无用赋值 |
| 局部优化 |
| 基本块及相关概念：  基本块：指程序中一顺序执行语句序列，其中只有一个入口和一个出口。入口就是其中第一个语句，出口就是其中最后一个语句。  如果一条三地址语句为x:=y+z，则称对x定值并引用y和z。  在一个基本块中的一个名字，所谓在程序中的某个给定点是活跃的，是指如果在程序中(包括在本基本块或在其它基本块中)它的值在该点以后被引用。 |
| 局限于基本块范围内的优化称为基本块内的优化，或称局部优化。  在一个基本块内通常可以实行下面的优化:  1.删除公共子表达式 2.删除无用赋值 3.合并已知量  4.临时变量改名 5.交换语句的位置 6.代数变换 |
| 划分四元式程序为基本块的算法:  1.求出四元式程序中各个基本块的入口语句:  1) 程序第一个语句，或  2) 能由条件转移语句或无条件转移语句转移到的语句，或  3) 紧跟在条件转移语句后面的语句。  2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)或到一转移语句(包括该转移语句)或一停语句(包括该停语句)之间的语句序列组成的。  3. 凡未被纳入某一基本块中的语句，可以从程序中删除。 |
| 基本块的DAG表示:基本块的DAG是一种带有下述标记或附加信息的DAG  1.图的叶结点以一标识符或常数作为标记，表示该结点代表该变量或常数的值;  2.图的内部结点以一运算符作为标记，表示该结点代表应用该运算符对其后继结点所代表的值进行运算的结果;  3.图中各个结点上可能附加一个或多个标识符(称附加标识符)表示这些变量具有该结点所代表的值。 |
| 基本块的DAG表示及应用:与各四元式相对应的DAG结点形式:见PPT 23页 |
| 基本块的DAG构造算法：  对基本块中每一四元式，依次执行以下步骤:  1. 准备操作数的结点  2. 合并已知量  3. 寻找公共子表达式  4. 删除无用赋值  1.准备操作数的结点  1.1如果NODE(B)无定义，则构造一标记为B的叶结点并定义NODE(B)为这个结点;  1.2如果当前四元式是0型，则记NODE(B)的值为n，转4。  1.3如果当前四元式是1型，则转2(1)  1.4如果当前四元式是2型，则(i)如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C)为这个结点；(ii)转2(2)。  2.合并已知量  (1) 如果NODE(B)是标记为常数的叶结点，则转2(3)；否则，转3(1)。  (2) 如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2(4)；否则，转3(2)。  (3) 执行op B (即合并已知量)。令得到的新常数为P。如果NODE(B)是处理当前四元式时新构造出来的结点，则删除它。如果NODE(P)无定义，则构造一用P作标记的叶结点n。置NODE(P)=n，转4。  (4)执行B op C (即合并已知量)。令得到的新常数为P。如果NODE(B)或NODE(C)是处理当前四元式时新构造出来的结点，则删除它。如果NODE(P)无定义，则构造一用P作标记的叶结点n。置NODE(P)=n，转4。  3.寻找公共子表达式  (1) 检查DAG中是否已有一结点，其唯一后继为NODE(B)且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。  (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。  4. 删除无用赋值  如果NODE(A)无定义，则把A附加在结点n上并令NODE(A)=n;否则，先把A从NODE(A)结点上的附加标识符集中删除（注意，如果NODE(A)是叶结点，则其A标记不删除）。把A附加到新结点n上并置NODE(A)=n。转处理下一四元式。 |
| DAG的优化：  合并已知量  删除公共子表达式  删除无用赋值  在基本块外被定值并在基本块内被引用的所有标识符，就是作为叶子结点上标记的那些标识符。  在基本块内被定值并且该值在基本块后面可以被引用的所有标识符，就是DAG各结点上的那些附加标识符。 |
| 流图定义：  每个流图以基本块为结点。如果一个结点的基本块的入口语句是程序的第一条语句，则称此结点为首结点。如果在某个执行顺序中，基本块B2紧接在基本块B1之后执行，则从B1到B2有一条有向边。即，如果  1.有一个条件或无条件转移语句从B1的最后一条语句转移到B2的第一条语句；或者  2.在程序的序列中，B2紧接在B1的后面，并且B1的最后一条语句不是一个无条件转移语句。  我们就说B1是B2的前驱，B2是B1的后继。 |
| 循环优化 |
| 对循环中的代码，可以实行：  1.代码外提 2.强度消弱 3.删除归纳变量(变换循环控制条件) 4.循环展开 5.循环合并 |
| 代码外提：  循环不变运算: 对四元式A:=B op C，若B和C是常数，或者到达它们的B和C的定值点都在循环外。  变量A在某点d的定值到达另一点u（或称变量A的定值点d到达另一点u），是指流图中从d有一通路到达u且该通路上没有A的其它定值。  把循环不变运算提到循环体外。  代码外提条件:不变运算所在的结点是L所有出口结点的必经结点  代码外提条件: A在循环中其他地方未再定值,才能把循环不变运算A:=B op C外提;  外提条件:循环中所有A的引用点都是而且仅仅是这个定值才能到达  查找循环L的不变运算的算法：  1.依次查看L中各基本块的每个四元式，如果它的每个运算对象或为常数，或者定值点在 L外，则将此四元式标记为"不变运算";  2.重复第3步直至没有新的四元式被标记为"不变运算"为止;  3.依次查看尚未被标记为“不变运算”的四元式，如果它的每个运算对象或为常数，或定值点在L之外，或只有一个到达一定值点且该点上的四元式已被标记为"不变运算"，则把被查看的四元式标记为"不变运算"。  代码外提算法：  1. 求出L的所有不变运算  2. 对每个不变运算s:A:=B op C 或 A:=op B 或 A:=B检查是否满足条件(1)或(2)  (1)  (i) s所在的结点是L所有出口结点的必经结点;  (ii) A在L中其他地方未再定值;  (iii) L中所有A的引用点只有s中的A的定值才能到达。  (2) A在离开L后不再是活跃的，并且条件(1)的(ii)和(iii)成立。所谓A在离开L后不是活跃的是指，A在L的任何出口结点的后继结点入口处不是活跃的。  3.按步骤1所找出的不变运算的次序，依次把符合2的条件(1)或(2)的不变运算s外提到L的前置结点中。但是，如果s的运算对象(B或C)是在L中定值的，那么，只有当这些定值四元式都已外提到前置结点中时，才能把s也外提到前置结点中。 |
| 强度消弱：把程序中执行时间较长的运算转换为执行时间较短的运算。  说明：  强度消弱主要是对与归纳变量有线性关系的变量赋值进行；  经过强度消弱后，循环中可能出现一些新的无用赋值；  对于消弱下标变量地址计算的强度非常有效。 |
| 删除归纳变量：  如果循环中对变量I只有唯一的形如I:=IC的赋值，且其中C为循环不变量，则称I为循环中的基本归纳变量。  如果I是循环中一基本归纳变量，J在循环中的定值总是可化归为I的同一线性函数，也即J=C1\*IC2，其中C1和C2都是循环不变量，则称J是归纳变量，并称它与I同族。  一个基本归纳变量也是一归纳变量。 |
| 强度削弱和删除归纳变量的统一算法：  1. 利用循环不变运算信息，找出循环中所有基本归纳变量。  2. 找出所有其它归纳变量A，并找出A与已知基本归纳变量X的同族线性函数关系 FA(X)。  3. 对2中找出的每一归纳变量A，进行强度削弱。  4. 删除对归纳变量的无用赋值。  5. 删除基本归纳变量。如果基本归纳变量B在循环出口之后不是活跃的，并且在循环中，除在其自身的递归赋值中被引用外，只在形如  if B rop Y goto L  中被引用，则可选取一与B同族的归纳变量M来替换B进行条件控制。最后删除循环中对B的递归赋值的代码。 |

|  |
| --- |
| 第十一章 目标代码生成 |
| 代码生成器：代码生成器的输入包括源程序的中间表示，以及符号表中的信息  1.利用符号表信息决定数据对象运行时地址  2.类型检查 |
| 1.代码生成是把语义分析后或优化后的中间代码变换成目标代码。  2.代码生成着重考虑的问题  2.1如何使生成的目标代码较短；  2.2如何充分利用计算机的寄存器，减少目标代码中访问存贮单元的次数。  2.3如何充分利用计算机的指令系统的特点 |
| 目标代码一般有以下三种形式：  1.能够立即执行的机器语言代码，所有地址已经定位；  2.待装配的机器语言模块。执行时，由连接装配程序把它们和某些运行程序连接起来，转换成能执行的机器语言代码；  3.汇编语言代码。尚须经过汇编程序汇编，转换成可执行的机器语言代码。 |
| 目标机器模型：  考虑一个抽象的计算机模型  1.具有多个通用寄存器，他们既可以作为累加器，也可以作为变址器。  2.运算必须在某个寄存器中进行。  3.含有四种类型的指令形式 |
| 四元式的中间代码变换成目标代码，并在一个基本块的范围内考虑如何充分利用寄存器：  1.尽可能留：在生成计算某变量值的目标代码时，尽可能让该变量保留在寄存器中。  2.尽可能用：后续的目标代码尽可能引用变量在寄存器中的值，而不访问内存。  3.及时腾空：在离开基本块时，把存在寄存器中的现行的值放到主存中 |
| 待用信息：如果在一个基本块内，四元式i对A定值，四元式j要引用A值，而从i到j之间没有A的其他定值，那么，我们称j是四元式i的变量A的待用信息。（即下一个引用点）  1 (x,x)表示变量的待用信息和活跃信息。其中i表示待用信息，y表示活跃，^表示非待用和非活跃；  2 在符号表中，(x，x)→(x，x)表示后面的符号对代替前面的符号对；  3 不特别说明，所有说明变量在基本块出口之后均为非活跃变量。 |
| 计算待用信息和活跃信息的算法步骤：  1. 开始时，把基本块中各变量的符号表登记项中的待用信息栏填为“非待用”，并根据该变量在基本块出口之后是不是活跃的，把其中的活跃信息栏填为“活跃”或“非活跃”；  2. 从基本块出口到基本块入口由后向前依次处理各个四元式。对每一个四元式i: A:=B op C，依次执行下面的步骤：  1) 把符号表中变量A的待用信息和活跃信息附加到四元式i上；  2) 把符号表中A的待用信息和活跃信息分别置为“非待用”和“非活跃”；  3) 把符号表中变量B和C的待用信息和活跃信息附加到四元式i上；  4) 把符号表中B和C的待用信息均置为i，活跃信息均置为“活跃”。 |
| 寄存器描述和地址描述：  寄存器描述数组RVALUE：动态记录各寄存器的使用信息  例：RVALUE[R]={A,B}  变量地址描述数组AVALUE：动态记录各变量现行值的存放位置  例：AVALUE[A]={R1, R2, A} |
| 补充说明：  1.因为寄存器的分配是局限于基本块范围之内的，一旦处理完基本块中所有四元式，对现行值在寄存器中的每个变量，如果它在基本块之后是活跃的，则要把它存在寄存器中的值存放到它的主存单元中。  2.要特别强调的是，对形如：A:=B的四元式，如果B的现行值在某寄存器Ri中，则无须生成目标代码，只须在RVALUE(Ri)中增加一个A，(即把Ri同时分配给B和A)，并把AVALUE(A)改为Ri 。 |
| 代码生成算法：  对每个四元式: i: A:=B op C，依次执行：  1. 以四元式: i: A:=B op C 为参数，调用函数过程GETREG(i: A:=B op C)，返回一个寄存器R，用作存放A的寄存器。  2. 利用AVALUE[B]和AVALUE[C]，确定B和C现行值的存放位置B’和C’。如果其现行值在寄存器中，则把寄存器取作B’和C’  3. 如果B’≠R，则生成目标代码：  LD R, B’  op R, C’  否则生成目标代码 op R, C’  如果B’或C’为R，则删除AVALUE[B]或AVALUE[C]中的R。  4. 令AVALUE[A]={R}, RVALUE[R]={A}。  5. 若B或C的现行值在基本块中不再被引用，也不是基本块出口之后的活跃变量，且其现行值在某寄存器Rk中，则删除RVALUE[Rk]中的B或C以及AVALUE[B]或AVALUE[C] 中的Rk ，使得该寄存器不再为B或C占用。 |
| 寄存器分配：GETREG(i: A:=B op C) 返回一个用来存放A的值的寄存器   |  | | --- | | 1 尽可能用B独占的寄存器  2 尽可能用空闲寄存器  3 抢占用非空闲寄存器 |   1 如果B的现行值在某个寄存器Ri中，RVALUE[Ri]中只包含B，此外，或者B与A是同一个标识符，或者B的现行值在执行四元式A:=B op C之后不会再引用，则选取Ri为所需要的寄存器R，并转4；  2 如果有尚未分配的寄存器，则从中选取一个Ri为所需要的寄存器R，并转4；  3 从已分配的寄存器中选取一个Ri为所需要的寄存器R。最好使得Ri满足以下条件：  占用Ri的变量的值也同时存放在该变量的贮存单元中，或者在基本块中要在最远的将来才会引用到或不会引用到。  4.（1）如果V的地址描述数组AVALUE[V]说V还保存在R之外的其他地方，则不需要生成存数指令；  （2）如果V是A，且不是B或C，则不需要生成存数指令；  （3）如果V不会在此之后被使用，则不需要生成存数指令；  （4）否则，生成 目标代码 ST Ri，V |