

第1章 编译引论

1.1 编译引论

- 基本概念：
 - 编译程序/翻译程序 P2
 - 源程序、源语言、目标程序、目标语言、宿主机、宿主语言 P2
 - 程序执行：编译阶段、运行阶段 P2

1.2 编译程序的表示与分类

- 语言分类：
 - 描述方式：命令式、声明式
 - 类型检查时间：静态类型、动态类型
 - 隐式数据转换：强类型、弱类型
 - 实现方式：编译型、解释型
- 编译器表示：函数表示 $T=C(S)$ 、T型图、符号表示 P3

1.3 编译程序的结构与编译过程

- 词法分析器：源程序 \rightarrow 属性字流 P5
- 语法分析：属性字流 \rightarrow 语法树/分析树 P6
- 语义分析与中间代码生成：语法树 \rightarrow 中间代码 P6
- 代码优化：中间代码 \rightarrow 优化后的中间代码 P6
- 目标代码生成：优化后中间代码 \rightarrow 机器语言代码/汇编语言代码 P6
- 分析与综合，前端与后端 P7
- 表格管理 P9
- 出错处理 P9
- 遍，三遍扫描，遍设置的考虑因素 P9

1.5 编译程序结构的实例模型

- 一遍编译程序结构、PRIME机上AHPL语言的两边编译程序、PDP-11机上C语言的三遍编译程序、Tiger编译程序、GCC编译程序结构框架 P9

1.6 编译程序的构造

- 构造要素：源语言、目标语言、编译方法与工具 P14
- 生成方式：编程语言、移植、自编译、自动生成 P15

第2章 形式语言与自动机理论基础

2.1 文法和语言

- 基本概念：
 - 字母表/符号表 Σ ：符号或字符元素的非空有穷集合 P19

- 字母表上的字符串集合 Σ^* P20
- 符号串的长度、符号串的子串、前缀、真前缀、后缀、真后缀 P20
- 符号串的连接、符号串的方幂、符号串集合的乘积、符号串集合的方幂、集合的闭包（正闭包、自反闭包） P20
- 文法 $G = (V_N, V_T, S, P)$, 分别表示非终结符号集、终结符号集、开始符号/公理、产生式集 P22
 - 元语言：用于描述另一种语言的语言，元语言中的符号称为元语言符号 P22
- 语言：字符串的集合 P23
 - 直接推导 \Rightarrow 、多步(>0)推导 \Rightarrow^+ 、直接(≥ 0)推导 \Rightarrow^*
 - 最左/右推导、规范推导（最右推导）、规范规约 P23
 - 句型、句子、规范句型 P23
 - 递归推导、文法递归（左递归、右递归）、直接递归文法（直接左递归文法、直接右递归文法） P23
 - 文法是语言的形式化定义、文法等价 P24
- 文法表示：BNF、扩展BNF、语法图表示 P25
- 语法分析树与二义性、二义文法 P28
- 文法分类：0型文法/短语结构文法-图灵机、1型文法/上下文有关文法-线性有界自动机、2型文法/上下文无关文法-非确定下推自动机、3型文法/正则文法-确定的有限状态自动机 P29

2.2 有限自动机

- 确定的有限自动机DFA: $M = (Q, \Sigma, f, q_0, Z)$, $f: Q \times \Sigma \rightarrow Q$, 表示方法有状态图、状态表、五元组 P31
 - M接受/识别字符串, 拒绝/不识别字符串的定义, $L(M)$ P32
- 非确定的有限自动机NFA: $M = (Q, \Sigma, f, q_0, Z)$, $f: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ P33
 - NFA与DFA区别: NFA状态转换函数值是一个状态子集, 带 ϵ 转换 P33
- NFA与DFA等价性: 任何非确定的有限状态自动机M都存在一个确定的有限状态自动机 M' , 使得 $L(M)=L(M')$
 - ϵ -closure(I): 状态集 I 经过0步转换可以到达的状态集合 P35
 - a 弧转化 I_a : 状态集 I 的 ϵ 闭包, 经过一条 a 弧到达的状态集的 ϵ 闭包 P36
- NFA确定化算法: 子集法 P37
- DFA化简: 消除无关状态, 消除等价状态: 划分法 P38

2.3 正规式与有限自动机

- 正规式与正规集, 闭包 $*$ 、连接、并 $|$ P43
- 正规式RE与自动机FA的等价性 P44
 - 字母表 Σ 上的确定的有限自动机M所接受的语言 $L(M)$ 是 Σ 上的一个正规集
 - 对于 Σ 上的每一个正规式 r , 存在一个 Σ 上的非确定有限自动机M, 使得: $L(M)=L(r)$
- 正规式RE与有限自动机DFA可以等价转换 P49
 - FA转RE: 拓广、按照 $*$ 、 $|$ 顺序替换
 - RE转FA: 拓广与替换、确定化、最小化

第3章 词法分析器

3.1 词法分析与词法分析程序

- 关键问题：定义词法规则、识别输入字符串中的单词
- 词法分析、词法分析程序 P58

3.2 词法分析程序设计与实现

- 词法分析程序输入输出：
 - 单词：语言中具有独立意义的最小语法单位，一般包括关键字、常数、标识符、运算符、界限符 P59
 - 属性字Token：(type, code)，是单词的内部表示 P59
- 源程序输入：对半互补输入缓冲模式 P60
- 源程序预处理：滤掉源程序的注释、剔除源程序的无用字符、宏替换、文件包含嵌入与条件编译嵌入 P61
- 词法分析器设计：
 - 对各类单词构造状态图
 - 状态图合并
 - 初态合并
 - 调整冲突编号
- 词法分析器的实现：
 - 数据中心法：主表（状态，分表地址/子程序入口），分表（输入字符，转换状态） P67
 - 程序中心法 P64

3.3 词法分析程序的自动生成

- Lex/Flex P68

第4章 语法分析-自上而下分析

4.1 语法分析综述

- 语法分析器功能：利用源语言文法，对源程序串经过词法分析器输出的属性字流分析，识别出语法树结构的语法范畴的表示 P80
- 语法分析方法：自上而下分析 P80，自下而上分析 P81

4.2 自上而下分析

- 一般自上而下分析：带回溯的自上而下分析，试探性的推导过程 P81
- 不确定性原因 P82
 - 假匹配导致回溯，可提取左公因子消除回溯
 - FIRST集合：设G是二型文法且文法G不含左递归，则G中的非终结符的每个候选式 α 的终结首符集FIRST(α)为 $\text{FIRST}(\alpha) = \{ a \mid \alpha \Rightarrow a \dots, a \in VT \}$ ，若 $\alpha \Rightarrow \epsilon$ ，则 $\epsilon \in \text{FIRST}(\alpha)$ P85
 - G的左递归导致无止境匹配，可消除G的左递归

4.3 递归下降分析方法和递归下降分析器

- LL(1)分析器：自左向右扫描，最左推导，最多向前看1个输入字符
 - 组成：总控程序、分析栈、LL(1)分析表

- LL(1)分析表的构造：利用FOLLOW集 P94
- LL(1)结论 P95

第5章 语法分析-自下而上分析

5.1 基于移进规约的自下而上分析

- 基本概念：
 - 短语、直接短语、句柄 P101

5.3 LR分析

- LR分析：一类对源程序串进行自左向右扫描并进行规范归约的语法分析方法
- LR(K)：自左向右扫描，最右推导逆序（规范规约），每一步仅根据分析站当前已经已经规约的全部语法符号对\$最多向前看k个输入字符 P119
- LR分析器组成：总控程序、分析栈、LR分析表 P114
 - 分析栈：状态符号 q_i 和文法符号 X_i P114
 - 分析表：LR分析器核心，包括动作表（action表）、状态转换表（goto表） P115
- LR(0)分析器：
 - 基本概念：
 - 前缀：句型的任意首部
 - 活前缀：规范句型不含句柄之后任何符号的前缀，可归前缀 P119
 - 活前缀有效：若文法G的一个LR(0)项目 $[A \rightarrow \beta_1 \cdot \beta_2]$ 对活前缀 $\alpha\beta_1$ 是有效的，当且仅当存在规范推导 $S \Rightarrow \alpha A \omega \Rightarrow \alpha \beta_1 \beta_2 \omega$
 - LR(0)项目 P120，LR(0)项目集规范族 P123
 - 项目分类：规约项目，接受项目，移进项目，待约项目 P120
 - 步骤 P125
 - 拓广文法
 - 构造文法项目集闭包
 - 求状态转换函数 $GO(I, X)$
 - 构造文法G的LR(0)项目集规范族
- 冲突：移进-规约冲突、规约-规约冲突 P129
- SLR(1)分析：对LR(0)冲突项目集，考察移进项、规约符号的FOLLOW集，若彼此不交则可解决冲突 P130
- LR(1)有效项目：搜索当前项目后面可以跟的1个搜索符 P132
- LALR(1)分析：文法G的LR(0)项目集规范族与LR(1)项目集规范族合并同心项目集
 - 同心项目集：项目相同，仅搜索符不同的项目集 P135
 - LALR(1)可能产生规约规约冲突，不会产生移进规约冲突 P136
- 任何一个二义文法不是LR文法，但可以规定优先级、结合性排除二义性

5.5 LR分析的错误处理与恢复

- 编译中的错误种类：词法错误、语法错误、语义错误(静态、动态)、违反环境限制的错误
- 错误处理：
 - 紧急恢复方式：抛弃一个输入符号，直到为同步符号集合

- 短语级恢复：局部校正
- 出错产生式
- 全局纠正

5.6 语法分析程序自动生成器

- yacc: LALR(1)分析器 P142

第6章 语义分析与中间代码生成

6.1 语法制导翻译

- 语义分析：扫描程序，分析程序表达的含义
 - 主要内容：符号的声明与使用（符号表+作用域检查），类型的标记与操作（类型检查）
- 语法制导翻译：对上下文无关文法制导下的语言进行翻译，是语法分析的同时进行语义处理的一种翻译技术 P163
 - 语法制导定义：基于上下文无关文法，较抽象、隐去实现细节的翻译说明，每个文法符号都有综合属性和继承属性，属性值通过语义规则定义 P164
 - 语义规则：用于建立属性之间的依赖关系 P164
- 综合属性：结点的属性值通过子结点的属性值计算得到 P165
- 继承属性：结点的属性值是由该结点的父节点和/或兄弟节点属性定义的 P166
- 依赖图：属性的依赖关系规定了计算顺序，用依赖图表示 P166

6.2 符号表

- 符号表：将声明信息收集放到一个属性表中，用于对标识符的声明和使用进行描述和分析，包括作用域、声明和使用顺序、重复声明
- 作用域检查
 - 作用域：定义了变量的可见范围和存活区间，将变量的使用和声明相关联

6.3 类型检查

- 类型：一个值的集合以及一个可以施加在这些值上的操作集合
 - 分类：基本类型、复合类型
- 类型体系、类型构建、类型转换、类型提升、类型等价
- 类型检查分类：静态类型、动态类型、无类型

6.4 中间语言

- 中间代码：介于源语言和目标代码之间的一种代码
- 常见形式：逆波兰式、N元式（三元式、间接三元式、四元式）、图
 - 逆波兰式：后缀表达式 P183
 - 三元式：NO. (OP, ARG1, [ARG2])
 - 间接三元式：间接码表+三元式
 - 四元式：NO. (OP, [ARG1], [ARG2], RESULT)
 - 图/树

6.5 中间代码生成

- 说明类语句负责定义信息，一般不产生目标代码，只填写符号表，提供语义检查和存储分配的依据 P187
- 常量语句 P187
- 简单说明类语句 P187
- 复合类型说明语句 P188
- 赋值语句 P189
- 算术表达式语句 P189
- 逻辑表达式语句 P189
- 控制流类语句：改变执行顺序
 - 语句标号与拉链-反填技术：解决先引用后定义问题 P191
 - 语句标号分类：定义性出现：L1: S, 使用性出现：goto L1
 - 语句标号表结构：（标号名，是否是定义，addr）
 - 条件语句翻译：if-else P191
 - 循环语句翻译：while P193
 - 多分支语句翻译：switch P194
- 数组的翻译 P196
 - 数组说明：内情向量表（每维大小，维数，体积，数元地址不变部分C，数组首地址） P196
 - 数组元素引用的翻译：地址计算， $T=a-C+V$ P197
- 函数说明和调用语句的翻译 P198

第7章 运行环境

7.1 程序运行时的存储组织与分配

- 基本概念：
 - 过程、活动、生存期、绑定
 - 名字的声明（作用域）与存储空间的对应方式：静态
 - 名字的绑定（生存期）与存储空间的对应方式：动态
 - 环境、状态，名字、存储、值 P203
- 存储区保护对象：生成的目标代码、目标代码运行时的数据空间、记录过程活动的控制栈 P204
- 存储分配的基本单元：过程的活动记录 P204
- 存储分配策略：静态存储分配、栈式存储分配、堆式存储分配 P205

7.2 静态运行时存储分配

- 限制： P208
 - 数据对象的长度和它在内存中的位置的限制必须在编译时知道
 - 不允许递归过程，因为一个过程的所有活动使用同样的局部名字结合
 - 数据结构不能动态建立，因为没有运行时的存储分配机制

7.3 基于栈的运行时环境的动态存储分配

- 数据空间内容：生存期在本过程本次活动中的数据对象 (局部变量、临时变量...)、管理过程活动的记录信息 (过程调用中断时当前机器的状态信息) P208
- 简单的栈式存储分配的实现：不允许嵌套，允许过程递归调用 P208
- 嵌套过程语言的展示存储分配的实现 P210
 - 访问链SL (存取链) P211
 - 嵌套层次显示表 display P211

7.4 基于堆的运行时环境的动态存储分配

- 特点：长度可变，不要求创建、销毁顺序 P213
- 问题：悬空引用 P214，内存碎片、内存泄漏
- 垃圾收集算法：
 - 引用计数算法：多线程一致性问题、循环引用问题
 - 标记清除算法：不可达对象视为垃圾清除，内存耗尽触发
 - 拷贝算法：任何时刻只能使用一半内存
 - 按代垃圾收集：根据存活时间划分新生代、旧生代

第8章 代码优化

8.1 代码优化概述

- 代码优化概念：等价变换、更高效 P221
- 改进程序运行效率的途径：改进算法，在源程序级上等价变换，充分利用系统提供的程序库，编译时优化 (不可替代) P221
- 根据优化所涉及的源程序的范围分类 P222
 - 局部优化：基本块内优化
 - 循环优化：隐式、显式循环体内优化
 - 全局优化：源程序大范围内优化
- 根据优化相对于编译逻辑功能实现的阶段分类 P222
 - 中间代码级：目标代码生成前的优化
 - 目标代码级：目标代码生成后的优化
- 根据优化具体实现角度分类
 - 常量合并与传播：静态表达式代换成常量 P222
 - 公共子表达式删除：提取公共表达式为新的变量，避免重复计算 P222
 - 无用赋值的删除：两次赋值间无变量引用，第一次赋值可删除 P222
 - 死代码删除：永假值代码删除 P223
 - 无用转移语句删除 P223
 - 循环不变量或不变代码外提：避免反复计算不变代码 P223
 - 函数内嵌：简洁的函数可直接内嵌调用函数中 P223
 - 循环转换：运算强度削弱
 - 简单循环内的运算强度削弱：乘法替换为加法 P224

- 动态循环内的运算强度削弱：多个循环控制参数是变量 P224
- 多步循环内的循环归纳变量外提：一个循环内循环变量值更新多次 P225
- 符合变量循环的转换：多个循环控制变量 P226

8.2 局部优化


- 局部优化以基本块为单位 P227
- 基本块的定义：顺序执行的语句序列，只有唯一入口（序列第一条语句）和唯一出口（序列最后一条语句） P227
- 基本块划分：
 - 确定基本块的入口语句：第一条语句/条件转移语句转移到的语句/紧跟在条件转移后面的语句 P228
 - 确定基本块的入口语句，构造其所属的基本块：该入口语句到下一入口语句之前的所有语句/该入口语句到下一个转移语句之间的所有语句/该入口语句到程序的停止或暂停语句之间的所有语句 P228
 - 未包含在基本块的语句删除 P228
- 程序的控制流图/流图：具有唯一首结点的有向图 $G = (N, E, n_0)$ ，表示基本块语句执行的流向关系 P229
- DAG：无环路的有向图 P229
- 基本块的DAG表示：结点上带有下列标记的DAG P230
 - 叶结点：标识符或常量为其唯一标记，叶结点为标识符时，代表名字的初值可加下标0表示初值
 - 内部节点：用运算符标记，表示计算的值
 - 各结点可添加多个标识符，附加在同一结点上的多个标识符具有相同的值
- 常见四元式与DAG结点对应关系 P232
 - 0型：一个结点：赋值语句、无条件转向语句
 - 1型：两个结点：单目运算
 - 2型：三个结点：双目运算、取数组元素的定值语句、条件语句
- 基本块的DAG的构造算法 P232
 - 构造叶结点
 - 捕捉已知量，合并常数（删除原常数结点）
 - 捕捉公共子表达式（删除荣誉的公共子表达式）
 - 捕捉可能的无用赋值

8.3 控制流分析与循环查找

- 基本概念：
 - 结点序列构成循环条件：强连通性+入口唯一 P236
 - 必经结点：流图G中 n_0 到 n_j 必须经过 n_i ，则 n_i 是 n_j 的必经结点，记为 $n_i \text{ DOM } n_j$ ，偏序关系 P237
 - 必经结点集：流图G中结点n的全部必经结点，记为D(n) P237
 - 回边：流图G中一条有向边 $a \rightarrow b$ ，且 $b \text{ DOM } a$ ，则 $a \rightarrow b$ 为回边，记为 $\langle a, b \rangle$ P238
- 利用回边求流图：设 $\langle n, d \rangle$ 为回边，则由d、n以及所有通路到达n而该通路不经过d的所有结点序列构成一个循环L，d是L的唯一入口 P238

8.4 数据流分析

- 基本概念：

- 点：一条中间语言语句在其代码序列中的位置，入口点、出口点、相邻点 P239
- 定值点：变量A获得值的中间代码的位置d，记为 ，定值方式有：赋值语句、输入语句、函数调用的形参与实参的结合
- 引用点：引用变量A的中间代码的位置d，记为<A>，eg：赋值语句、单目运算
- 到达-定值：流图G中，变量A在G中某点d的定值到达另一点p，是指流图中从点d有一通路到达点p且该通路上没有对A的再定值 P239
- 引用-定值链（ud链）：假设在程序中某点P引用了变量A的值，则把G中能到达P的A的定值点的全体，称为A在引用点P的引用一定值链（ud链），是相对于引用点的定值情况 P242
- 活跃变量：在程序中对某变量A和某点P，如果存在一条从P开始的通路，其中引用了A在点P的值，则称A在点P是活跃的，否则称A在点P是死亡的
- 定值-引用链（du链）：假设在程序中某点P对一个变量A定值，则把该定值能到达A的引用点的全体，称为A在定值点P的定值—引用链，是相对于定值点的引用情况（du链） P243
- 数据流方程：
 - 基本元素：
 - 到达定值信息：完成常量合并，删除无用赋值
 - 活跃变量信息：寄存器优化
 - 公共子表达式信息：删除冗余运算
 - 集合定义：in：流入基本块的定值点集，out：流出基本块的定值点集，gen：基本块中定值并到达出口点的信息，kill：基本块中注销（即重新定义）的定值点集 P240
 - 典型数据流方程：
 - 到达-定值数据流方程：用于求解ud链 P240
 - $out[B] = gen[B] \cup (in[B] - kill[B])$
 - $in(B) = \cup out(P) \quad P \in P(B)$
 - 可用表达式数据流方程 P244
 - 表达式x+y在点P可用：如果从初始结点到P的每条路径上都计算x+y，并且在最后一个x+y到P之间未对x或y定值，则表达式x+y在点P可用
 - $out[B] = (in[B] - kill[B]) \cup gen[B]$
 - $in(B) = \cap out[P] \quad (B \text{不是开始块}), \text{设 } in(B1) = \Phi \quad (B1 \text{是开始块})$
 - 活跃变量数据流方程 P242
 - $def_L(B)$ ：在基本块B中定值，且定值之前未曾在B中引用过的变量的集合
 - $use_L(B)$ ：在基本块B中引用的，但在引用前未曾在B中定值的变量集
 - $in_L(B) = (out_L(B) - def_L(B)) \cup use_L(B)$
 - $out_L(B) = \cup in_L(s), s \in S(B), S(B) \text{表示B后继基本块集合}$

8.5 循环优化

- 循环优化准备：循环查找、涉及循环的所有基本块的数据流分析
- 循环前置结点：循环入口点前建立的一个新结点，以循环入口结点为唯一后继
- 代码外提：循环中不变运算提到循环前置结点 P246
- 强度削弱：寻找基本归纳变量 -> 强度削弱 -> 删除归纳变量 P247

第9章 目标代码生成

9.1 代码生成器设计中的要点

- 代码生成器的输入输出：输入为中间表示，输出为机器代码 P253
- 中间代码到目标代码的流程：指令选择、寄存器分配、指令调度
 - 指令选择的复杂度依赖于IR的层次、指令级本身的特点、期望生成的目标代码质量
 - 寄存器分配主要涉及将哪些变量存放到寄存器中、分配哪个寄存器给一个变量，NP完全问题
 - 指令调度：指令的执行顺序，NP完全问题

9.2 简单代码生成器的构造

- 简单的代码生成器：针对每个基本块完成
- 基本过程：遍历基本块中三地址指令，确定需要将哪些操作对象加载到寄存器，生成从内存加载到寄存器的指令，生成计算指令，生成必要的写回指令（从寄存器到内存）
- 翻译过程中的辅助信息：
 - 寄存器描述符：记录每个寄存器当前存储的变量
 - 地址描述符：记录每个程序变量最新值的存储位置
- getReg(I)：输入指令I，输出指令中变量的寄存器分配情况
 - 如果变量x已经在寄存器中，直接返回对应的寄存器
 - 如果y不在寄存器，且有空闲的寄存器，则返回一个空闲寄存器
 - 否则选择一个已经占用的寄存器，根据寄存器描述符将对应的变量写回内存，然后返回对应的寄存器
- 指令选择：对树状IR，每条机器指令可以表示为IR树的一部分：tree pattern
- 寄存器分配：重写中间代码使得每个时刻使用的临时变量比可用寄存器数少，将更多的临时变量分配到同一个寄存器，不改变程序的行为
 - 固定寄存器分配：固定局部变量、堆栈、临时计算的寄存器数量，其他的保留在内存中
 - 局部寄存器分配：以基本块为单位，在基本块开始时加载，结束时写回，基本块内按需分配，根据指令更新寄存器描述符和变量描述符
 - 线性扫描：确定变量的活跃范围（变量活跃的程序点集）和活跃区间（包含活动范围的最小区间），一个变量在程序中某个点是活跃的是指该变量再次被赋值之前会被读
 - 画出每个变量的活跃区间时间图，由此分配寄存器
 - 寄存器不够用时，将变量值spill到内存中
 - 图着色：对程序数据流图，每个基本块内同时活跃的变量间连边，然后做点着色，颜色数即为需要的寄存器数

