编译原理

编译原理

1 编译的步骤
2 文法的分类 ※※※※※
3 词法分析——DFA和NFA
4 语法分析——自顶向下分析和自底向上分析
5 LL分析法和LR分析法
6 符号表管理技术
7 运行时存储管理
8 代码优化，目标代码生成

1 编译的步骤

编译是将高级程序语言（源代码）转换为计算机可执行的低级语言（目标代码）的过程。步骤：

1. **词法分析**：把源代码分解为词法单元（token），如关键字、标识符、运算符、常量等
2. **语法分析**：根据源代码构建抽象语法树，检查源代码是否符合语法规则
3. **语义分析、生成中间代码**：遍历语法树，进行类型检查、语义检查，管理符号表，生成中间代码
4. **代码优化**：提高程序效率，改进代码质量
5. **生成目标程序**：根据机器进行，生成目标机器语言

1-3是前端（与源程序有关），45是后端（与目标机有关）

2 文法的分类 ※※※※※

- 0型文法：**无限制**，一个短语产生另一个短语，可由**图灵机**接受
- 1型文法：**上下文有关**，$P:xUy::=xuy$，在上下文中非终结符->终结符。可由**线性界限自动机**接受
- 2型文法：**上下文无关**，$P:U::=u$，改写时，不必考虑上下文。可由**下推自动机**接受
- 3型文法：**正则文法**，左线性或右线性，$P:U::=t$ 或 $U::=Wt$（左线性，t是终结符），右侧必须推出一个非终结符。可由**有穷自动机**接受

3 词法分析——DFA和NFA

DFA：确定的有穷自动机，在当前状态和输入符号下，只能存在唯一的下一个状态

NFA：不确定的有穷自动机，在当前状态和输入符号下，可能存在多种下一个状态，可以通过空转移或多个转移路径到达下一个状态。NFA都可以转化为DFA

4 语法分析——自顶向下分析和自底向上分析

自顶向下分析：从文法的起始符号开始，逐步向下展开产生式，直到推导出输入串，是预测性的分析。常见递归下降分析和LL(k)。

- **不能处理左递归文法（U->Uv匹配无穷无尽），而且存在回溯问题（非终结符规则右部有多个选择，需要改写文法或超前扫描）**
- **递归子程序法**：常见的语法分析程序
- **LL分析法**：见5

自底向上分析：从输入串开始，通过重复查找当前句型的句柄（最左简单短语），直到文法的起始符号，重点在于识别并构造输入串的局部语法结构。常见LR(k)，主要内容如下：

- **移进规约分析**：建立符号栈，确定下一步动作是移进还是归约。**没真正解决句柄的识别**
- **算符优先分析**：用于上下文无关文法，先确定终结符之间的优先关系矩阵。栈顶优先级大于栈外则归约，否则移进
- **LR分析法**：见5

5 LL分析法和LR分析法

LL分析法：**是自顶向下分析**，自左向右扫描、分析输入串

- 需要构造FIRST（头符号集合，指示该规则可匹配的输入符号）和FOLLOW（后继符号集合，指示推导出ϵ的输入符号）。
- 构造**分析表**（指示用非终结符的哪一条规则匹配输入串，进行一步最左推导）
- 如果**分析表没有多重定义入口**则为LL(1)文法。左递归或二义性文法不是LL(1)

LR分析法：**是自底向上分析**，从左到右扫描，自底向上归约。优先归约当前举行句柄，是规范规约

- 找到规范句型活前缀，构造**GOTO表**，构造了一个识别所有规范句型活前缀的自动机
- 构建分析动作表（**ACTION表**）
- SLR分析表：根据文法构造识别规范句型活前缀的有穷自动机DFA

6 符号表管理技术

符号表：记录名字（变量、函数、过程、数组、标号、参数等）和特性信息（种类、类型、位数、参数个数、目标地址、数值等），多为栈式符号表

7 运行时存储管理

分为静态存储分配和动态存储分配。

- **静态**：开辟数据区，按顺序为模块分配空间，再给模块内变量按顺序分配存储
- **动态**：整个数据区为一个堆栈，进入过程时分配，退出时撤销。
    - 活动记录：局部数据区，参数区（隐式prev abp, ret addr, ret value + 显式参数），display区（存放外层模块活动记录基地址）

8 代码优化，目标代码生成

基本块内优化：DAG图（有向无环图）导出中间代码

全局优化：数据流分析，到达定义分析（out=gen+(in-kill)），活跃变量分析（in=use+(out-def)）

目标代码生成：全局寄存器分配可以使用图着色算法。通过数据流分析构建变量冲突图，进行图着色（k为全局寄存器数量，则需要找临边小于k的点并去掉，否则不分配。随后反向添加，依次分配颜色）