编译原理课程笔记

卢雨轩 19071125

2022 年 3 月 25 日

第一章　绪论

主要内容

- 编译原理及其设计概述　3 学时
- 语言与文法　5 学时
- 词法问题　6 学时
- 语法分析　13 学时　自顶向下（LL）；自底向上（LR）；
- 语义分析　10 学时
- 运行环境　3 学时　过程调用，符号管理
- 优化与代码生成　2 学时
- 总结

1.1　计算机语言的发展

- 机器语言　Machine Language
  - 二进制代码　Binary Code
- 汇编语言　Assemble Language
  - 二进制代码与助记符
  - 接近计算机硬件指令系统
- 高级语言　High Level Language
  - 语句定义数据、描述算法
- 命令语言　Command
  - 功能封装
- 高级语言的分类
  - 命令式语言 Imperative Language
    - Fortran, Basic, Pascal, C, COBOL, ALGOL
  - 函数式语言 Functional Language
    - LISP, ML
  - 逻辑式语言 Logical Language
    - Prolog
  - 面向对象语言

1.2　翻译系统

- 翻译程序 – Translator
  - 将某种语言描述的程序（源程序，Source Code）翻译为等价的另一种语言描述的程序（目标程序，Object Code）的程序。
- 编译程序 – Compiler
  - 将某一种高级语言描述的程序翻译为汇编、机器语言描述的程序
- 解释程序 – Interpreter
- 编译系统 = 编译程序 + 运行系统
- 其他翻译程序
  - 诊断编译程序
  - 优化编译程序
  - 交叉编译程序
  - 可变目标编译程序
  - 并行编译程序
  - 汇编程序、交叉汇编程序、反汇编程序

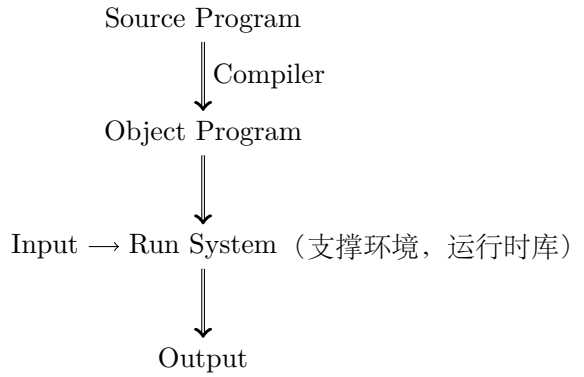


图 1.1: 编译系统 = 编译程序 + 运行系统

1.3　翻译系统的功能分析

- 分析
  - 词法，语法，语义
- 翻译
  - 语句的翻译，代码生成
- 例如：标识符左值与右值的绑定
- 变量：存储单元　　名字：值
- 函数：目标代码序列　名字：入口地址

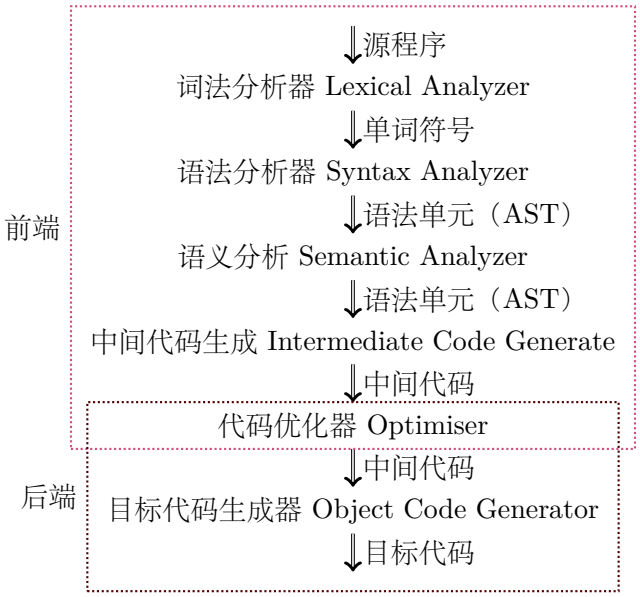
1.4　翻译系统的总体结构

1.4.1　词法分析

词法分析器（Lexical Analyzer）完成词法分析。扫描源程序，并转换为 Token 串，同时查找语法错误，进行标识符登记（管理符号表）。

输入：字符串。

输出：（种类码，属性值）对。



1.4.2　语法分析

语法分析器（Syntax Analyzer）完成语法分析。实现『组词成句』，构造分析树，指出语法错误并指导翻译。

输入：Token 序列

输出：语法成分（抽象语法树）

1.4.3　语义分析

语义分析器（Semantic Analyzer）分析由语法分析器给出的语法单位的语义。

- 获取标识符的属性：类型、作用域等
- 语义检查：运算合法、取值范围
- 子程序的静态绑定：代码的相对地址
- 变量的静态绑定：数据的相对地址

1.4.4　中间代码生成

升层中间代码，如：前、后缀表达式、三地址表示、LLVM IR。

1.4.5　代码优化

对中间代码的优化处理：提高运行速度、节省储存空间

- 与机器无关的优化
  - 常量合并
  - 公共子表达式提取等
- 与机器有关的优化
  - 循环展开
  - 向量化
  - 访存优化
  - 寄存器排布

1.4.6　目标代码生成

将中间代码转换为目标机器上的指令代码或者汇编代码。

1.4.7　表格管理

管理编译过程中的各种符号表，辅助完成语法、语义检查，完成静态绑定，管理编译过程。

1.4.8　错误处理

进行错误的检查、报告以及纠正。

- 词法错误：拼写、定义；
- 语法错误：语句结构、表达式结构；
- 语义错误：表达式类型不匹配；

1.4.9　前端与后端

将编译过程分为前端与后端。其中：

- 前端：与源语言有关，与目标机器无关的部分
  - 词法分析、语法分析、语义分析、中间代码生成、与机器无关的优化
- 后端：与目标机器有关的部分
  - 与机器有关的代码优化、目标代码生成

好处：实现新语言和新机器，只需要实现前端、后端中的某一个。

1.5　编译程序的生成

通过自动化技术来生成编译程序。

1.5.1　T 形图

用 T 形图表示语言翻译：从源语言翻译为目标语言，而翻译器本身是表示语言。

1.5.2　交叉编译（Cross Compiling)、移植

例 1.5.1. *A 机上有一个 C 语言编译器，是否可以利用此编译器实现 B 机器上的 C 语言编译器？*

1. 用 C 语言编制 B 机器的编译程序 P0（C → B）
2. 用 P1 编译 P0，得到 P2
3. 用 P2 编译 P0，得到 P3

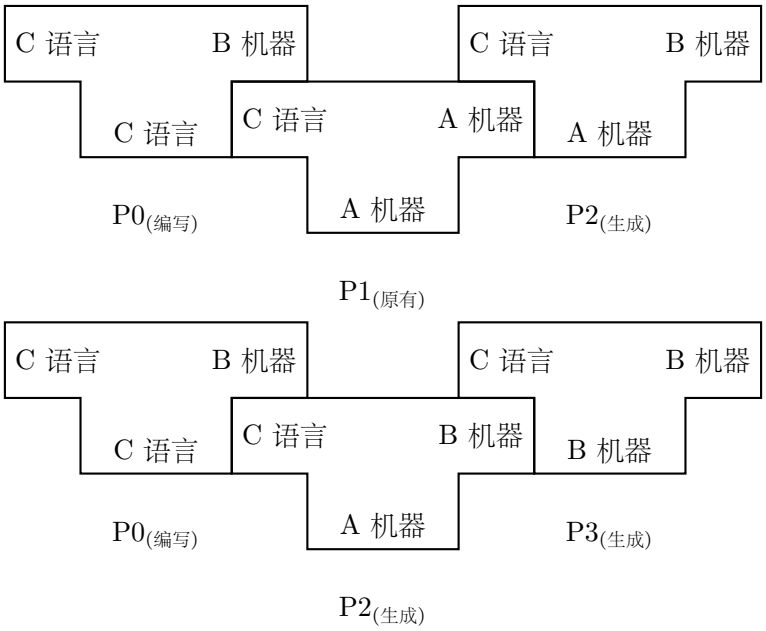


图 1.2: 第 4 页 例 1.5.1　图例

第二章　高级语言及其文法

2.1　语言概述

- 什么是语言
  - 自然语言（Natural Language）
    - 是人与人之间的通讯工具
    - 语义（Semantics）：环境，背景知识，语气，二义性，难以形式化
  - 计算机语言（Computer Language）
    - 计算机系统之间、人机之间的通讯工具
    - 严格的语法（Grammar）、语义（Semantics）
      - 易于形式化，从而实现自动化。
- 语言的描述方法
  - 人类『已有』的语言基础
    - 自然语言：自然、方便——非形式化
    - 数学语言（符号）：严格，准确——形式化
  - 机器要『掌握』规则：形式化描述
    - 高度抽象——表示（representation）
    - 方便计算机表示——有穷描述（finite description）
    - 扎实的理论基础——结构（structure）
- 语言——形式化内容的提取
  - 字符（Character）：语言中允许出现的基本字符
  - 单词（Token）：满足一定规则的字符串
  - 句子（Sentence）：满足一定规则的单词序列
  - 语言（Language）：满足一定规则的句子集合
- 语言是字和组合字的规则——结构性描述
- 程序设计语言——形式化内容的提取
  - 字符（Character）：语言中可出现的所有字符——基本字符集
  - 单词（Token）：满足词法规则的字符串（RL）
  - 语句（Sentence）：满足语法规则的单词序列（CFL）
  - 程序（Program）：满足语法规则的单词序列（CFL）
  - 程序设计语言：组成程序的所有语句的集合（CFL）

- 描述形式——文法
  - 语法——语句
    - 语句的组成规则
    - 描述方法：BNF 范式（Backus Normal Form，Backus-Naur Form）、语法描述图

2.2　基本定义

定义 2.2.1. 字母表（*Alphabet*）Σ 是一个非空有穷集合，字母表中的元素称为字母表的一个字母（*Letter*），也叫字符（*Character*）

定义 2.2.2. 字母表 Σ 上的符号串（*String*）

1. ϵ 是 Σ 上的一个符号串（叫做空串）；
2. 若 x 是 Σ 上的字符串，而 $a \in \Sigma$，则 xa 是 Σ 上的符号串；
3. y 是 Σ 上的符号串，当且仅当它由 *(1)* 和 *(2)* 导出

非形式化描述:由字母表中的符号组成的任何有穷序列被成为在该字母表上的符号串,也称作『字』(*Word*)

定义 2.2.3. 设 Σ_1, Σ_2 是两个字母表，Σ_1 与 Σ_2 的乘积 (Product) $\Sigma_1\Sigma_2 = \{ab | a \in \Sigma_1, b \in \Sigma_2\}$

定义 2.2.4. Σ 的幂（*Power*）:

1. $\Sigma^0 = \{\epsilon\}$
2. $\Sigma^n = \Sigma^{n-1}\Sigma$

定义 2.2.5. Σ 的正闭包（Positive Closure）:

1. $\Sigma^+ = \{x | x$是Σ上的非空字符串$\}$
2. $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

Σ 的克林闭包（Kleene Closure）:
$\Sigma^* = \{\epsilon\} \cup \Sigma^+$
$= \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

定义 2.2.6. 设 Σ 是一个字母表，$\forall L \subseteq \Sigma^*$，$L$ 称为字母表 Σ 的一个语言 (Language)，$\forall x \in L$，x 叫做 L 的一个句子。

定义 2.2.7. 设 s 是字符串:

- 前缀：移去 s 的尾部的若干（含 *0*）个字符
- 后缀：移去 s 的头部的若干（含 *0*）个字符
- 子串：从 s 中删去一个前缀和一个后缀
- 字序列：从 s 中删去 *0* 个或多个符号，不要求连续
- 长度：符号串中符号的数目

定义 2.2.8. 设 x 和 y 是符号串，他们的连接（Concatenation）xy 是把 y 的符号写在 x 的符号之后得到的符号串

定义 2.2.9. 设 x 是符号串，x 的 n 次幂（Power）为:

- $x^0 = \epsilon$
- $x^n = x^{n-1}x$

2.3　文法的定义

如何实现语言结构的形式化描述?

定义 2.3.1. 文法（*Grammar*）G 是一个四元组

$$G = (V, T, P, S)$$

其中，

V ——变量（Variable）的非空有穷集。$\forall A \in V$，A 叫做语法变量（*syntactic variable*），也叫非终极符号（*nonterminal*）。

T ——终极符（Terminal）的非空有穷集。$\forall a \in T$，a 叫做终极符。$V \cup T = \emptyset$。

P ——产生式（Production）的非空有穷集。对于 $a \to b$，a 是左部，b 是右部。

S ——$S \in V$，文法 G 的开始符号（Start symbol）。

约定:

- 只写产生式，第一个产生式的左部为开始符号
- 对一组有相同左部的产生式
  $\alpha \to \beta_1, \alpha \to \beta_2, \alpha \to \beta_3, \ldots$ 可以记为 $\alpha \to \beta_1|\beta_2|\beta_3 \ldots$ $\beta_1, \beta_2, \beta_3$ 称为候选式（Candidate）
- 形如 $\alpha \to \epsilon$ 的产生式叫做空产生式，也可叫做 ϵ 产生式
- 符号
  - 英文大写字母为语法变量
  - 英文小写字母为终结符号
  - 英文较后的大写字母为语法变量或者终极符号
  - 英文较后的大写字母为终极符号行
  - 希腊字母表示语法变量和终极符号组成的行

定义 2.3.2. 对于文法 $G = (V, T, P, S)$:

语法范畴 A $L(A) = \left\{w | w \in T^* \text{且} A \overset{*}{\Rightarrow} w\right\}$

语言（*Language*）$L(G) = \left\{w | w \in T^* \text{且} S \overset{*}{\Rightarrow} w\right\}$

句子（*Sentence*）$\forall w \in L(G)$

句型（*Sentential Form*）$\forall \alpha \in (V \cup T)^*$，如果 $S \overset{*}{\Rightarrow} \alpha$，则称 α 是 G 产生的一个句型。

例 2.3.1. 定义字母表 $\Sigma = \{0, 1\}$ 上的字符串:

- ϵ 是 Σ 的一个字符串（叫做空串）
- 若 x 是 Σ 上的字符串，而 a 是 Σ 的元素，则 xa 是 Σ 上的字符串。

$S \to \epsilon$

$S \to S0|S1$

2.4　文法的乔姆斯体系

定义 2.4.1. 对于文法 $G = (V, T, P, S)$:

G 叫做 *0* 型文法，*Type 0 Grammar*，也叫短语结构文法（*PSG*，*Phrase Structure Grammar*）

$L(G)$ 是 *0* 型语言，也叫短结构语言，可递归枚举集。

定义 2.4.2. 对于 *0* 型文法文法 $G=(V,T,P,S)$:
如果对于 $\forall\alpha\to\beta\in P$，均有 $|\beta|\geq|\alpha|$，则 G 是 *1* 型文法，或上下文有关文法。

定义 2.4.3. 对于 *1* 型文法文法 $G=(V,T,P,S)$:
如果对于 $\forall\alpha\to\beta\in P$，均有 $|\beta|\geq|\alpha|$，并且 $\alpha\in V$ 则 G 是 *2* 型文法，或上下文无关文法。

定义 2.4.4. 对于 *2* 型文法文法 $G=(V,T,P,S)$:
如果对于 $\forall\alpha\to\beta\in P$:
如果形如 $A\to wB$ 和 $A\to w$，其中 $A,B\in V,w\in T^+$: G 是右线性文法。
如果形如 $A\to Bw$ 和 $A\to w$，其中 $A,B\in V,w\in T^+$: G 是左线性文法。
则 G 是 *3* 型文法，或正则文法。

2.5　BNF 范式

定义 2.5.1. *Backus-Normal*:

- $\alpha\to\beta$ 表示为 $\alpha::=\beta$
- 非终极符: 用 <> 扩起来
- 终极符: 基本符号集
- 其他:
  - $\beta(\alpha_1|\alpha_2|\ldots|\alpha_n)=\beta\alpha_1|\beta\alpha_2|\ldots|\beta\alpha_n$
  - $\{\alpha_1|\alpha_2|\ldots|\alpha_n\}_l^u$: 出现 l 次到 u 次
  - $\{\alpha_1|\alpha_2|\ldots|\alpha_n\}m$: $l=0,u=m$
  - $[\alpha]=\alpha|\epsilon$

2.6　CFG 的语法树

定义 2.6.1. *CFG* $G=(V,T,P,S)$ 的语法树为满足如下条件的树:

1. 每个节点的标记 $x\in V\cup T\cup\{\epsilon\}$
2. 如果节点 V 的标记为 A，V 从左到右的子节点 $V_1,V_2,\ldots,V_k$ 的标记依次为 $y_1,y_2,\ldots,y_k$，则 $A\to y_1y_2\ldots y_k\in P$
3. 根节点标记为 S
4. 中间节点的标记为变量 $x\in V$
5. 从左到右的叶子节点 $v_1,\ldots,v_n$ 的标记 $x_1,\ldots,x_n$ 组成的串 $x_1x_2\ldots x_n$ 为该树的结果。
6. 如果 v 的标记为 ϵ，则它没有兄弟。

例 2.6.1. $X\to(S)|S|\epsilon$，请画出句型 $((S))$ 的语法树。

定义 2.6.2. 满足语法树定义中除第三条外条件的树，称作 A-子树。

定理 2.6.1. 有一颗结果为 α 的语法树 $\iff S\overset{*}{\Rightarrow}\alpha$

定义 2.6.3. 如果 $S\overset{*}{\Rightarrow}\alpha A\beta$，且 $A\overset{+}{\Rightarrow}\gamma$，则称 γ 是句型 $\alpha\gamma\beta$ 相对于变量 A 的短语。
如果 $S\overset{*}{\Rightarrow}\alpha A\beta$，且 $A\Rightarrow\gamma$，则称 γ 是句型 $\alpha\gamma\beta$ 相对于变量 A 的直接短语。
最左直接短语称为『句柄』。
用树解释短语:

- 短语：子树的结果是相对于子树根的短语
- 直接短语：只有父子两代的子树的结果
- 句柄：一个句型的分析树中最左的、只有父子两代的子树的结果
- 一个句型的短语数量等于非独子的中间结点数量

第三章　语法分析

3.1　语法分析的任务

- 输入：Token 序列
- 输出：
  - 语法成分以及组成
  - 表现形式：语法树
  - 错误报告
- 出错处理：
  - 定位、续编译
- 自顶向下和自底向上
  - 自顶向下：预测分析法（LL（1））
  - 自底向上：算符优先分析法（LR（0），SLR（1），LR（1），LALR（1））

3.2　自顶向下的问题与 CFG 改造

- 自顶向下的分析
  - 从文法开始符号出发，寻找每个输入符号串的最左推导
- 重要问题：虚假匹配
  - 发现不匹配需要回退
- 二义性及其消除
  - dangling else 问题
  - 改造文法，要求最近/最长匹配
- 消除左递归
  - 将 $A \to A\alpha|\beta$ 转换为 $A \to \beta A'; A' \to \alpha A'|\epsilon$

3.3　First 和 Follow 集

- 希望寻找一类文法，可以根据当前输入的符号选择产生式。
- $A \to \alpha_1|\alpha_2|\ldots|\alpha_n$：
  - 对于 $\forall\alpha \in (V \cup T)^*$：

    * FIRST(α) = $\{a|\alpha \overset{*}{\Rightarrow} a\ldots, \alpha \in T\}$
    * 当 $\alpha \overset{*}{\Rightarrow} \epsilon$ 时，$\epsilon \in$ FIRST(α)
  - 对于 $\forall A \in V$，A 的后随符号集
    * $FOLLOW(A) = \{a|S \overset{*}{\Rightarrow} \ldots Aa\ldots, a \in T\}$

- 求 FIRST(X) 的算法

Algorithm 1 求 FIRST(X)

procedure FIRST
  for $X \in V \cup T$ **do**
    if $X \in T$ **then**
      return X
    else
      FIRST(X) $\leftarrow \begin{cases} \{a|X \to a\cdots \in P\} & X \to \epsilon \notin P \\ \{a|X \to a\cdots \in P\} \cup \epsilon & X \to \epsilon \in P \end{cases}$
    end if
  end for
  while $X \in V$, and FIRST(X) keeps changes **do**
    if $X \to Y\cdots \in P$, $Y \in V$ **then**
      FIRST(X) $\leftarrow$ FIRST(X) $\cup$ (FIRST(Y) $-$ $\{\epsilon\}$)
    end if
    if $X \to Y_1Y_2\ldots Y_n \in P$, $Y_1Y_2\ldots Y_{i-1} \overset{*}{\Rightarrow} \epsilon$ **then**
      for $X \in [1, i]$ **do**
        FIRST(X) $\leftarrow$ FIRST(X) $\cup$ (FIRST(Y_i) $-$ $\{\epsilon\}$)
      end for
    end if
    if $X \to Y_1Y_2\ldots Y_n \in P$, $Y_1Y_2\ldots Y_n \overset{*}{\Rightarrow} \epsilon$ **then**
      FIRST(X) $\leftarrow$ FIRST(X) $\cup$ $\{\epsilon\}$
    end if
  end while
end procedure

- 求 FIRST(α) 的算法
  - 设 $\alpha = X_1X_2X_3\ldots X_n$

Algorithm 2 求 FIRST(α)

procedure FIRST
  FIRST(α) $\leftarrow$ FIRST(X_1) $-$ $\{\epsilon\}$
  $k \leftarrow 1$
  while $\epsilon \in$ FIRST(X_k), $k < n$ **do**
    FIRST(α) $\leftarrow$ FIRST(α) $\cup$ (FIRST(X_{k+1}) $-$ $\{\epsilon\}$)
    $k \leftarrow k + 1$
  end while
  if $k = n$, $\epsilon \in$ FIRST(X_n) **then**
    FIRST(α) $\leftarrow$ FIRST(α) $\cup$ $\{\epsilon\}$
  end if
end procedure