```yaml
title: 大学课程 | 编译原理知识点
tags:
  - 编译原理
  - 大学课程
categories:
  - 学习笔记
abbrlink: 50753
reward: true
copyright: true
date: 2019-10-25 22:22:39
cover: https://npm.elemecdn.com/justlovesmile-img/001243-1573661563c583.jpg
top_img: https://npm.elemecdn.com/justlovesmile-img/001243-1573661563c583.jpg
```

作者博客：Justlovesmile's BLOG
大三编译原理复习知识点

问题?

- 什么是解释器？什么是编辑器？什么是前端后端？分析和综合？遍？翻译过程的输入输出？T型图的意义描述？
- 词法分析：什么是正则表达式？什么是有穷自动机？DFA？NFA？区别，特点？基本概念？正则表达式到NFA到DFA，再最小化。构建方法。扫描器功能的输入输出？什么是字母表，元符号，正则表达式的三种基本操作
- 0/1/2/3型文法？什么是最左推导？最右推导？什么是终结符，非终结符？什么是产生式？如何识别二义性，消除方法？语言到文法？
- 递归下降？LL(1)判断是不是？消除左递归，提取左公因子，First集follow集，构造分析表，对一个句子分析。LL(1)三种基本动作：生成（最左推导），匹配，接受。
- 自底向上？
- 语义分析：什么是属性？什么是属性文法？什么是联编？联编的时间？静态语义和动态语义？常见的静态语义？什么是符号表？作用,内容?描述-->属性文法？综合属性，基本属性
- 了解几种运行环境的特点：Fortran77 完全静态，不允许递归调用。基于栈的C，C++，Pascal。LISP完全动态
- 中间代码：种类，三元式，四元式，控制表达式，逆波兰，波兰。

第一章 概论

什么是编译器?

（1）

- 编译器是将一种语言翻译为另一种语言的计算机程序。
- 编译器将编写的程序作为输入，而产生用目标语言编写的等价程序
- 源程序→{编译器}→目标程序
  （2）
- 编译器是将便于人编写，阅读，维护的高阶计算机语言翻译为计算机能解读，运行的低阶机器语言的程序。
- 编译器将原始程序作为输入，翻译产生使用目标的等价程序。源代码一般为高阶语言，而目标语言则是汇编语言或目标机器的目标代码，有时也称作机器代码。

编译器分类结构

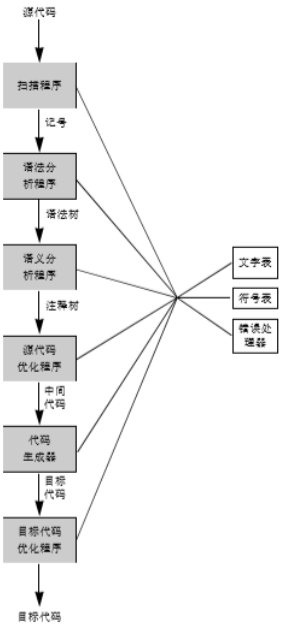
- 根据**语言文法的难易程度**以及**识别它们所需要的算法**分类：如**乔姆斯基分类结构**：
- 4类：分为0型，1型，2型，3型文法
  - 0型文法为：无限制文法
  - 1型文法为：上下文有关文法
  - 2型文法为：上下文无关文法
  - 3型文法为：正则文法
- 4个文法的定义是逐渐增加限制的

与编译器相关的程序

- （1）解释程序
- 如同编译器的一种语言翻译程序。与编译器的不同在于：它立即执行源程序而不是在翻译完成之后才执行目标代码。
- （2）汇编程序
- 用于特定计算机上的汇编语言的翻译程序
- （3）连接程序
- 将分别在不同的目标文件中编译或汇编的代码收集到一个可直接执行的文件中
- （4）装入程序
- 可处理所有与指定的基地址或起始地址有关的可重定位的地址
- （5）预处理器
- 在真正的翻译开始之前由编译器调用的独立程序
- （6）编辑器
- 编译器通常接受由任何生成标准文件（例如 ASCII 文件）的编辑器编写的源程序。编译器如今常与编辑器和其他程序捆绑进一个交互的开发环境--IDE中。
- （7）调试程序
- 可在被编译了的程序中判定执行错误的程序。编译器必须为调试程序提供恰当的符号信息。
- （8）描述器
- 在执行中搜集目标程序行为统计的程序
- （9）项目管理程序

翻译步骤

- 编译器内部包括了许多步骤或称为阶段。

- 

- （1）扫描程序：编译器阅读源程序。扫描程序会执行词法分析，将字符序列收集到称作**记号**的单元中。
- （2）语法分析程序：从扫描程序获取记号形式的源代码，并完成定义程序结构的语法分析。通常将语法分析的结果表示为**分析树**或者**语法树**。
- （3）语义分析程序：程序的语义确定程序的运行。但是大多数的程序设计语言都具有在执行前被确定而不易有语法表示和由分析程序分析的特征。这些特征被称为**静态语义**。而语义分析程序的任务就是分析这些语义。由语义分析程序计算的额外信息被称为属性，输出结果为**注释树**
- （4）源代码优化程序：源代码优化程序可能通过将其输出称为**中间代码**来使用三元式代码。
- （5）代码生成器：代码生成器得到中间代码，生成**目标机器的代码**。
- （6）目标代码优化程序：优化目标代码

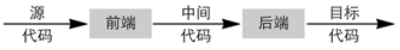
编译器中的主要数据结构

- 记号
- 语法树
- 符号表
- 常数表
- 中间代码
- 临时文件

什么是遍?

- 编译过程的几个阶段仅仅是逻辑功能上的一种划分，具体实现时，受不同源语言，设计要求，使用对象和计算机条件（如主存容量）的限制，往往将编译程序组织为若干遍。所谓“遍”就是对源程序或源程序的中间结果从头到尾扫描一次，并作有关的加工处理，生成新的中间结果或目标程序。
- 通常，每遍的工作由从外存上获得的前一遍的中间结果开始，完成它所含的有关工作之后，再把结果记录于外存，既可以将几个不同阶段合为一遍，也可以把一个阶段的工作分为若干遍。当一遍中包含若干阶段时，各阶段的工作是穿插进行的。

什么是前端后端?

- 通常认为，只依赖于源语言的的操作为前端，只依赖于目标语言的操作为后端。
- 如：扫描程序，分析程序，语义分析程序为前端。代码生成器为后端。
- 便于编译器的可移植性
- 

什么是分析与综合?

- 分析：分析源程序以计算其特性的编译器操作
- 综合：生成翻译代码时所涉及到的操作

什么是扫描器?

- 扫描器就是词法分析程序
- 其主要功能是依据词法规则，分析由字符组成的源程序，把它分割为一个一个具有独立意义的最小语法单位，即单词。

汇编语言的优缺点

- 优点：汇编语言大大提高了编程的速度和准确度
- 缺点：编写起来也不容易，阅读和理解很难；而且汇编语言的编写严格依赖于特定的机器，所以为一台计算机编写的代码在应用于另一台计算机时必须完全重写。

什么是静态语义

- 程序的语义确定程序的运行，但是大多数的程序设计语言都具有在执行之前被确定而不易由语法表示和由分析程序分析的特征。这些特征被称作**静态语义**。
- 一般的程序设计语言的典型静态语义包括**声明**和**类型检查**。由语义分析程序计算的额外信息（诸如数据类型）被称为属性，它们通常是作为注释或“装饰”增加到树中（还可将属性添加到符号表中）。

编译器中第一个考虑目标机的物理特性的模块是： 代码生成器

T 型图中S,T,H 分别代表什么?

| S T|
| H |

- 语言H(代表宿主语言) 编写的编译器将语言S(代表源语言) 翻译为语言T(代表目标语言)

T 型图描述自举及移植的过程

-

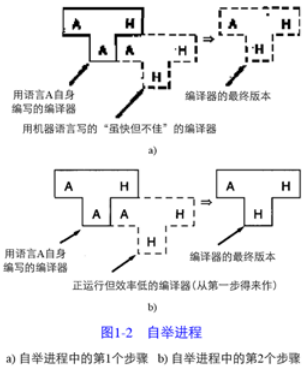


图1-2 自举进程

a) 自举进程中的第1个步骤 b) 自举进程中的第2个步骤

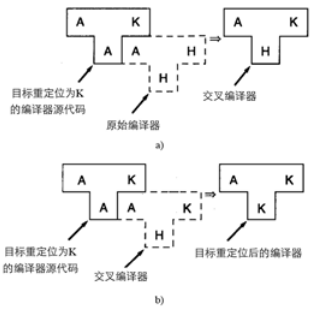


图1-3 移植一个在其自身源代码中编写的编译器

a) 步骤1 b) 步骤2

第二章 词法分析

什么是词法分析

- 将源程序读作字符文件并将其分为若干记号

记号分类

- 关键字：如if，while
- 标识符：用户定义的串
- 特殊符号：算术符号，一些多字符符号等

正则表达式

- 是一种表示字符串的格式
- 三种基本操作：选择，连结，重复（闭包）
- 元字符/元符号：正则表达式中有特殊含义的字符

什么是有穷自动机

- 是描述特定类型算法的数学方法。
- 圆圈表示状态，带有箭头的线表示记录一个状态向另一个状态的转换。

DFA（确定性有穷自动机）

-

定义：DFA（确定性有穷自动机）M由字母表$\sum$、状态集合S、转换函数T：$S\times\sum\rightarrow S$、初始状态$s_0\in S$以及接受状态的集合$A\subset S$组成。由M接受的且写作$L(M)$被定义为字符$c_1c_2\ldots c_n$串的集合，其中每个$c_i\in\sum$，存在状态$s_1=T(s_0,c_1),s_2=T(s_1,c_2),\ldots,s_n=T(s_{n-1},c_n)$，其中$s_n$是$A$（即一个接受状态）的一个元素。

- 给出一个状态和字符，通常肯定会有一个指向单个新状态的唯一转换

NFA（非确定性有穷自动机）

-

定义：NFA（nondeterministic finite automaton）M由字母表$\sum$、状态的集合S、转换函数T：$S\times(\sum\cup\{\varepsilon\})\rightarrow\wp(S)$、$S$的初始状态$s_0$，以及$S$的接受状态$A$的集合组成。由$M$接受的语言写作$L(M)$，它被定义为字符$c_1c_2\ldots c_n$，其中每一个$c_i$都属于$\sum\cup\{\varepsilon\}$，且存在关系：$s_1$在$T(s_0,c_1)$中、$s_2$在$T(s_1,c_2)$中、…、$s_n$在$T(s_{n-1},c_n)$中，$s_n$是$A$中的元素。

第三章 上下文无关文法

上下文无关文法与正则表达式的主要区别：

- 上下文无关文法的规则是递归的

终结符和非终结符

- 非终结符：在推导中必须被进一步替换的结构名
- 终结符：终结推导的字母表中的符号

什么是推导

- 推导是在文法规则的右边进行选择的一个结构名字的替换序列。推导以一个结构名字开始并以记号符号串结束。
- 最左推导：它的每一步中最左的非终结符都要被替换的推导。
- 最右推导：它的每一步中最右的非终结符都要被替换的推导。
- 最左推导和与其相关的分析树的内部节点的前序编号相对应；而最右推导则和后序编号相对应。

产生式

- 文法规则也被称为产生式。

二义性文法

- 可生成两个不同分析树的串的文法
- 解决方法：一，设置规则，即消除二义性规则。二，将文法改变成一个强制正确分析树构造的格式

语法分析器的作用

- 编译过程中，语法分析器的任务是
  (1) 分析单词串是如何构成语句和说明的
  (2) 分析语句和说明是如何构成程序的
  (3) 分析程序的结构

第四章 自顶向下的分析

自顶向下的分析

- 两类程序：回溯分析程序；预测分析程序
- 两类算法：递归下降分析；LL(1)分析

LL(1)文法

- LL(1)分析：第一个`L`指由左向右处理输入，第二个`L`为输入串描绘出一个最左推导，`1`是指先行一个符号
- 使用显示栈来完成分析
- 是非二义性的文法
- 对于文法G，其相关的LL(1)分析表的每个项目中至多只有一个产生式，则该文法就是LL(1)文法。
- LL(1)三种基本动作：生成（最左推导），匹配，接受

将BNF写为LL(1)分析算法

- 消除左递归：

$$A \rightarrow A\alpha \mid \beta$$
$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \varepsilon$$

- 提取左公因子：

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$
$$A \rightarrow \alpha A'$$
$$A' \rightarrow \beta \mid \gamma$$

FIRST集 定义：

令 X 为一个文法符号（一个终结符或非终结符）或 ε，则集合 First (X) 由终结符组成，此外可能还有 ε，它的定义如下：

1. 若 X 是终结符或 ε，则 First (X) = {X}。
2. 若 X 是非终结符，则对于每个产生式 X → X1 X2 . . . Xn， First (X) 都包含了 First (X1) - { ε } 。若对于某个 i < n，所有的集合 First (X1), . . . , First (Xi) 都包括了 ε，则 First (X) 也包括了 First (X i + 1) - { ε } 。若所有集合 First (X1), . . . , First (Xn) 包括了 ε，则 First (X) 也包括 ε 。

FOLLOW集 定义：

给出一个非终结符 A，那么集合 Follow (A) 则是由终结符组成，此外可能还有 $ 。
集合 Follo w (A) 的定义如下：

1. 若 A 是开始符号，则 $ 就在 Follo w (A) 中。
2. 若存在产生式 B → α A γ，则 First (γ) - { ε } 在 Follo w (A) 中。
3. 若存在产生式 B → α A γ，且在 Firs t (γ) 中，则 F ollo w (A) 包括 Follow (B) 。

SELECT集

1. 定义：
   给定上下文无关文法的产生式A→α, A∈VN,α∈V*, 若α不能推导出ε,则SELECT(A→α)=FIRST(α);如果α能推导出ε则：SELECT(A→α)= （FIRST(α)–{ε}）∪FOLLOW(A)。需要注意的是，SELECT集是针对**产生式**而言的。
2. LL(1)文法：
   一个上下文无关文法是LL(1)文法的充分必要条件是：对每个非终结符A的两个不同产生式，A→α, A→β,满足SELECT(A→α)∩SELECT(A→β)=空集 其中α，β不同时能推导出ε。

LL(1) 证明定理

1. 在每个产生式 A → α 1 | α 2 | . . . | α n 中，对于所有的 i 和 j： 1 ≤ i， j ≤ n， i ≠ j， First (α i) ∩ First (α j) 为空。
2. 若对于每个非终结符 A 都有 First (A) 包含了 ε，那么 First (A) ∩ Follow (A) 为空。

构造LL(1)预测分析表

1. 对于文法G的每一个产生式A→α执行第2，3步
2. 对每个终结符a∈FIRST(α)，把A→α加到M[A,a]中
3. 若ε∈FIRST(α)，则对任何b∈FOLLOW(A)把A→α加入[A,b]中
4. 其余无定义为出错

-

当非终结符A位于分析栈的顶部时，根据当前的输入记号（先行），必须使用刚刚描述过的分析办法做出一个决定：当替换栈中的A时应为A选择哪一个文法规则，相反地，当记号位于栈顶部时，就无需做出这样的决定，这是因为无论它是当前的输入记号（由此就发生一个匹配），还是不是输入记号（从而就发生一个错误），两者都是相同的。

通过构造一个**LL(1)**分析表(LL(1) parsing table)就可以表达出可能的选择。这样的表格基本上是一个由非终结符和终结符索引的二维数组，其中非终结符和终结符包括了要在恰当的分析步骤（包括代表输入结束的$）中使用的产生式选择。这个表被称为$M[N, T]$，这里的$N$是文法的非终结符的集合，$T$是终结符或记号的集合（为了简便，禁止将$加到T上），M可被认为是“运动的”表。我们假设表$M[N, T]$在开始时，它的所有项目均为空。任何在构造之后继续为空的项目都代表了在分析中可能发生的潜在错误。

根据以下规则在这个表中添加产生式：

1) 如果$A \rightarrow \alpha$是一个产生式选择，且有推导$\alpha \Rightarrow *a\beta$成立，其中$a$是一个记号，则将$A \rightarrow \alpha$添加到表项目$M[A, a]$中。
2) 如果$A \rightarrow \alpha$是一个产生式选择，且有推导$\alpha \Rightarrow *\varepsilon$和$S\$ \Rightarrow *\beta Aa\gamma$成立，其中$S$是开始符号，$a$是一个记号（或$），则将$A \rightarrow \alpha$添加到表项目$M[A, a]$中。

第五章 自底向上的分析

LR(1)

- LR(1)分析：L表示由左向右处理输入，R表示生成了最右推导，数字1表示先行一个符号
- 移进规约分析程序：主要任务是判断分析中的下一个句柄

第六章 语义分析

什么是语义分析

- 语义分析也可称为**静态语义分析**
- 语义分析包括：构造符号表，记录声明中建立的名字的含义，在表达式和语句中进行类型推断和类型检查，以及在语言的类型规则作用域内判断正确性
- 语义分析分为：程序的分析；由编译程序执行的分析

什么是属性

- 属性： 属性是编程语言结构的任意特性。属性在其包含的信息和复杂性等方面变化很大，特别是当它们能确定时翻译 / 执行过程的时间。属性的典型例子有：
  •变量的数据类型。
  •表达式的值。
  •存储器中变量的位置。
  •程序的目标代码。
  •数的有效位数。

什么是属性文法

- 确定语言实体的属性或特性，它们必须进行计算并写成属性等式或语义规则，并描述这些属性的计算如何与语言的文法规则相关。这样的一组属性和等式称作**属性文法**。

什么是联编

- 联编： 属性的计算及将计算值与正在讨论的语言结构联系的过程称作属性的联编。
- 联编时间： 联编属性发生时编译 / 执行过程的时间称作联编时间 。
- 执行之前联编的属性是**静态**的，
- 执行期间联编的属性是**动态**的。

静态动态

- 在如 C 或 Pascal 这样的静态类型的语言中，变量或表达式的数据类型是一个重要的编译时属性。
- FORTRAN7 7 中所有的变量都是静态分配。
- 程序的目标代码无疑是一个静态属性。
- 表达式的值通常是动态的，编译程序要在执行时生成代码来计算这些值。
- 变量的分配可以是静态的也可以是动态的，这依赖于语言和变量自身的特性
- LIS P 中所有的变量是动态分配的。
- C 和 Pasca l 语言混合了静态和动态的两种变量分配。
- 数 的有效位数在编译期间是一个不被明确探讨的属性。

符号表

- 是一种目录数据结构
- 符号表的主要操作：插入，查找，删除。
- 符号表的功能：
  （1） 建立存储信息
  （2） 类型检查
  （3） 数据地址

第七章 运行时的环境

运行环境

- 完全静态环境：FORTRAN77，所有数据都是静态的，执行期间保持固定。这样的环境可用来实现没有指针或动态分配，且过程不可递归调用的语言。
- 基于栈的环境：C，C++，Pascal，Ada。在允许递归调用以及每一个调用中都重新分配局部变量的语言中，不能静态地分配活动记录。相反地，必须以一个基于栈的风格来分配活动记录，即当进行一个新的过程调用（活动记录的压入时，每个新的活动记录都分配在栈的顶部，而当调用退出时则再次解除分配 。
- 完全动态环境：LISP。因为活动记录仅在对它们所有的引用都消失了才再重新分配，而且这又要求活动记录在执行时可动态地释放任意次，所以称这个环境为完全动态的 。

第八章 代码生成

中间代码

- 两种形式：三地址码，P代码
- 中间代码应具备的特性
  1）便于语法制导翻译
  2）既与机器指令的结构相近,又与具体机器无关.

控制语句

- 控制语句的分类：①无条件转移、②条件转移、③循环语句、④分支语句

代码优化

- 代码优化：对程序进行各种等价变换，使得从变换后的程序出发，能产生更有效的目标代码。
- 目的：产生高效的目标代码。
- 级别：局部优化、循环优化、全局优化。

补充

- 活前缀：右句型的前缀，而且其右端不会超过该句型的最右边句柄的末端。

直接短语

书中的定义：

> 令 G 是一个文法，S 是文法的开始符号，假定 αβδ 是文法 G 的一个句型，如果有
>
> $$S \overset{*}{\Rightarrow} \alpha A \delta \text{ 且 } A \overset{+}{\Rightarrow} \beta$$
>
> 则称 β 是句型 αβδ 相对于非终结符 A 的短语。特别是，如果有
>
> $$A \Rightarrow \beta$$
>
> 则称 β 是句型 αβδ 相对于规则 A→β 的直接短语，一个句型的最左直接短语称为该句型的

书中的意思总结来说，指的是如果子树中不再包含其他的子树，即A只能推导出b，而b不能再推出其他的式子，则b为此句型的直接短语

句柄

先来看一下书中的定义：

> 令 G 是一个文法，S 是文法的开始符号，假定 αβδ 是文法 G 的一个句型，如果有
>
> $$S \overset{*}{\Rightarrow} \alpha A \delta \text{ 且 } A \overset{+}{\Rightarrow} \beta$$
>
> 则称 β 是句型 αβδ 相对于非终结符 A 的短语。特别是，如果有
>
> $$A \Rightarrow \beta$$
>
> 则称 β 是句型 αβδ 相对于规则 A→β 的直接短语，一个句型的最左直接短语称为该句型的句柄。

书中的意思就是：直接短语中的最左直接短语为该句型的句柄。

短语

书上的定义如下:

令 G 是一个文法，S 是文法的开始符号，假定 αβδ 是文法 G 的一个句型，如果有

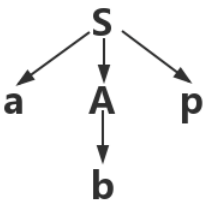
$$S \overset{+}{\Rightarrow} \alpha A \delta \text{ 且 } A \overset{+}{\Rightarrow} \beta$$

则称 β 是句型 αβδ 相对于非终结符 A 的短语。特别是，如果有

书上写的比较抽象，我这里简单解释一下，有两个文法，分别是:

```
S=*=>aAp（由于部分字符难以输入，在此用a,b,p代替）
A=+=>b
```

我们由此可以画出他的抽象语法树，如下:



那么，abp为此句型的短语

总结来说：**一个句型的语法树中任一子树叶结点所组成的符号串都是该句型的短语**，由这概念，那么我们自然可以想到，b也应该是该句型的一个短语。