语言翻译程序把一种语言（称作**源语言**）书写的程序翻译成另一种语言（称作**目标语言**）的等价程序。

编译程序：源语言是像...那样的高级语言，目标语言是像...那样的低级语言，这种翻译程序。

程序的处理过程

|  |  |
| --- | --- |
| 预处理程序的任务 | 一个源程序有时可能分成几个模块存放在不同的文件里，将这些源程序汇集在一起。 |
| 编译程序 |  |
| 汇编程序 | 可再装配（可重定位） |
| 装配/连接编辑程序、... |  |

编译程序的基本任务是将**源语言程序**翻译成等价的**目标语言程序**。

编译过程

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 词法分析的任务 | 从左到右一个字符一个字符的读入源程序，对构成源程序的字符流进行扫描和分解，从而识别出一个个**单词**。  单词（**单词符号**、**符号**）：逻辑上紧密相连的一组字符，这些字符具有集体含义。   |  | | --- | | 标识符 | | 保留字（关键字、基本字） | | 算符 | | 界符 | | ... | |
| 语法分析的任务 | 在词法分析的基础上将单词序列分解成各类**语法短语**。  这种语法短语也称为**语法单位**，可表示成**语法树**。   |  | | --- | | 程序 | | 语句 | | 表达式 | | ... |   语法分析所依据的是语言的**语法规则**，即描述程序结构的规则。 |
| 语义分析的任务 | 审查源程序有无语义错误，为代码生成阶段收集**类型信息**。 |
| 中间代码生成 | **中间语言**（**中间代码**）：在进行了上述的语法分析和语义分析阶段的工作之后，有的编译程序将源程序变成一种内部表示形式，这种内部表示形式。 |
| 代码优化的任务 | 对前一阶段产生的中间代码进行变换或进行改造，目的是使生成的目标代码更**高效**，即省时间和省空间。 |
| 目标代码生成的任务 | 把中间代码变换成特定机器上的**绝对指令代码**或**可重定位的指令代码**或**汇编指令代码**。 |

编译程序

|  |  |  |
| --- | --- | --- |
| 表格管理程序 | 词法分析程序 | 出错处理程序 |
| 语法分析程序 |
| 语义分析程序 |
| 中间代码生成程序 |
| 代码优化程序 |
| 目标代码生成程序 |

编译阶段

前端（front end）：依赖于源语言而与**目标机**无关的工作。

后端（back end）：那些依赖于目标机而一般不依赖于源语言，只与中间代码有关的那些阶段的工作。

编译程序是一个语言处理程序，它把一个高级语言程序翻译成某个机器的汇编语言程序或二进制代码程序，这个二进制代码程序在机器上运行以生成结果。

解释程序接受某个语言的程序并立即运行这个源程序。

扩展巴克斯范式（EBNF）

|  |  |
| --- | --- |
|  | 用尖括号括起来的中文字表示**语法构造成分**，或称**语法单元**；而用尖括号括起来的英文字表示一类**词法单元**。 |
|  | 表示左部的语法单位由右部定义，可读作“定义为”。 |
|  | 表示“或”，即多选项。 |
|  | 用花括号括起来的成分可以重复0次到任意多次。 |
|  | 用方括号括起来的成分为任选项，即出现一次或不出现。 |
| '**字符**' |  |

仅出现在右边的符号均为**词法单元**，即**单词**或**单词符号**。

# 文法和语言

（语言的）语法：一组规则，用它可以形成和产生一个合适的程序。

（语言的）语义

　　静态语义：一系列限定规则，并确定哪些合乎语法的程序是合适的。

　　动态语义（运行语义、执行语义）：表明程序要做些什么，要计算什么。

　　形式语义学

规则

元语言

文法

：使用一条规则，代替其左端的某个符号，产生其右端的符号串。

|  |  |  |
| --- | --- | --- |
| 字母表（符号集）：元素的非空有穷集合。 | | |
|  | （字母表上的）符号串集合：集合中的一切元素都是某字母表上的符号串，。 | |
| 符号：字母表中的元素。 | | |
| 符号串：由字母表中的符号组成的任何有穷序列。 | | |
|  | 空符号串（||）：不包含任何符号的符号串。 | |
|  | （符号串的）长度（||）：某符号串中有个符号，。 | |
|  | （符号串的）头：是一符号串，。 | |
|  |  | （符号串的）固有头：是一符号串，非空，。 |
|  | （符号串的）尾：是一符号串，。 | |
|  |  | （符号串的）固有尾：是一符号串，是非空的，。 |
| 运算 | | |
|  | （符号串和符号串的）连接（||）：设和是符号串，把的符号写在的符号之后得到的符号串。 | |
|  |  | （符号串的）方幂（||）：设是符号串，把自身连接次得到符号串，即。 |
|  | （符号串集合和符号串集合的）乘积：，即是满足属于，属于的所有符号串所组成的集合。 | |
|  |  | （字母表的）方幂。 |
|  |  | （字母表的）闭包：（||）：上的所有有穷长的串的集合。。 |
|  |  | （字母表的）正闭包（||）：。 |

|  |  |
| --- | --- |
| 规则（重写规则、产生式、生成式）：形如或的有序对。 | |
|  | 规则的左部：。 |
|  | 规则的右部：。 |
|  | （）读作“定义为”。 |
|  | （关于的）规则：。 |
|  | 规则：的产生式。  规则：上下文无关文法中某些规则可具有形式，其中，这种规则。 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 文法（||）：四元组。为非终结符集；为终结符集；为规则（）的集合，且至少包含一个非终结符，；、和是非空有穷集；是一个非终结符，至少要在一条规则中作为左部出现；和不含公共的元素，即。 | | | | | | |
|  | 非终结符（语法实体、变量）  使用尖括号括起非终结符。 | | | | | |
|  | （文法的）字母表（字汇表||）：。 | | | | | |
|  | 有时，为书写简洁，常把相同左部的产生式，形如缩写为。  这里的元符号读作“或”。 | | | | | |
|  | 有害规则：形为的产生式。 | | | | | |
|  | 多余规则：文法中那些连一个句子的推导都用不到的规则。 | | | | | |
|  |  | 非终结符是不可到达的：文法中的某些非终结符不在任何规则的右部出现，所以任何句子的推导中都不可能用到它们，这种非终结符。 | | | | |
|  |  | 非终结符是不可终止的：在文法中有这样的非终结符：不能够从它推导出终结符号串来，这种非终结符。 | | | | |
|  | 识别符（开始符）：。 | | | | | |
|  | 分类 | | | | | |
|  |  | 0型文法（短语文法）：设，它的每一个产生式是这样一种结构：且至少含有一个非终结符，而，。 | | | | |
|  |  |  | 1型文法（上下文有关文法、文法是1型的、文法是上下文有关的（context-sensitive））：设为一个文法，中的每一个产生式均满足，仅仅除外，。  产生式的形式是，其中、和都在中，，在中。 | | | |
|  |  |  |  | 2型文法（上下文无关文法、文法是2型的、文法是上下文无关的（context-free））：设，中的每一个产生式满足：是一个非终结符，，。  产生式的形式是，其中。 | | |
|  |  |  |  |  | 3型文法（正规文法）：设，中的每一个产生式的形式都是或，其中和都是非终结符，，。 | |
|  | 中的符号串之间的关系 | | | | | |
|  |  | 直接推导（||）  是的直接推导（（应用规则）直接产生、直接归约到||）：设是文法的规则（或说是中的一个产生式），和是中的任意符号串，有符号串、满足，。 | | | | |
|  |  | 长度为（）的推导（||）  推导出（产生）（推导长度为）（规约到||）：存在直接推导的序列（）。 | | | | |
|  |  | 长度为（）的推导（||）  （||）：有，或。 | | | | |
|  | （文法的）句型：设是一个文法，符号串是从识别符号推导出来的，即有，。 | | | | | |
|  |  | （文法的）句子：仅由终结符号组成，即，。 | | | | |
|  | （文法所产生的）语言（||）：集合，其中为文法识别符号，且。（文法描述的语言是该文法一切句子的集合。） | | | | | |
|  |  | 分类 | | | | |
|  |  |  | 0型语言：0型文法产生的语言。 | | | |
|  |  |  |  | 上下文有关语言：上下文有关文法产生的语言。 | | |
|  |  |  |  |  | 上下文无关语言：上下文无关文法产生的语言。 | |
|  |  |  |  |  |  | 正规语言：正规文法产生的语言。 |
|  | 关系 | | | | | |
|  |  | （文法和文法是）等价的：。 | | | | |
|  | 性质 | | | | | |
|  |  | 0型文法 | | | | |
|  |  |  | 0型文法的能力相当于图灵机（Turing Machine）。（任何0型语言都是递归可枚举的；反之，递归可枚举集必定是一个0型语言。） | | | |
|  |  | 上下文有关文法 | | | | |
|  |  |  | 如果是一个上下文有关文法，则存在另一个上下文有关文法，它所产生的语言与产生的相同，其中的开始符号不出现在的任何产生式的右边。 | | | |
|  |  | 上下文无关文法 | | | | |
|  |  |  | 若是由文法产生的语言，中的每一个产生式的形式均为，其中，（即可能为），则能由这样的一种文法产生，即每一个产生式或者为形式，其中为一个非终结符，即，；或者为形式，且不出现在任何产生式的右边。 | | | |
|  |  |  | 如果是一个上下文无关文法，则存在另一个上下文无关文法，它所产生的语言与产生的相同，其中的开始符号不出现在的任何产生式的右边。 | | | |
|  |  | 正规文法 | | | | |
|  |  |  | 如果是一个正规文法，则存在另一个正规文法，它所产生的语言与产生的相同，其中的开始符号不出现在的任何产生式的右边。 | | | |

（上下文无关文法有足够的能力描述现今程序设计语言的语法结构。）

|  |
| --- |
| 语法树（推导树、语法分析树、分析树）：给定文法，对于的任何句型都能构造与之关联的语法树（推导树）。这棵树满足下列4个条件：  （1）每个结点都有一个标记，此标记是的一个符号。  （2）根的标记是。  （3）若一个结点至少有一个它自己除外的子孙，并且有标记，则肯定在中。  （4）如果结点的直接子孙从左到右的次序是结点，其标记分别为，那么一定是中的一个产生式。 |

最左推导：在推导的任何一步，其中、是句型，都是对中的最左非终结符进行替换，这种推导。

最右推导（规范推导）：在推导的任何一步，其中、是句型，都是对中的最右非终结符进行替换，这种推导。

右句型（规范句型）：由规范推导所得的句型。

文法是二义的：一个文法存在某个句子对应两棵不同的语法树。（一个文法中存在某个句子，它有两个不同的最左推导。）（一个文法中存在某个句子，它有两个不同的最右推导。）

上下文无关语言是先天二义的：产生上下文无关语言的每一个文法都是二义的。

要判定任给的一个上下文无关文法是否为二义的，或它是否产生一个先天二义的上下文无关语言，这两个问题是递归不可解的。（不存在一个算法，它能在有限步骤内确切判定任给的一个文法是否为二义的。）

句型分析问题：如何知道所给定的符号串是文法的句型。

句型分析：一个识别输入符号串是否为语法上正确的程序的过程。

分析程序（识别程序）：在语言的编译实现中，完成句型分析的程序。

|  |  |  |  |
| --- | --- | --- | --- |
| 分析算法（识别算法） | | | |
|  | 分类 | | |
|  |  | 按识别符号串方向（？） | |
|  |  |  | 从左到右的分析算法：总是从左到右地识别输入符号串，首先识别符号传中的最左符号，进而识别右边的一个符号。 |
|  |  |  | 从右向左的分析算法 |
|  |  | 按建立语法树方向（？） | |
|  |  |  | 自顶向下分析法：从文法的开始符号出发，反复使用各种产生式，寻找“匹配”于输入符号串的推导。 |
|  |  |  | 自底向上的方法：从输入符号串开始，逐步进行“规约”，直至规约到文法的开始符号。 |

# 词法分析

词法分析程序（扫描程序）：执行词法分析的程序。

单词符号分类：关键字（保留字）、标识符、常数、运算符、界符。

用于词法规则描述的工具：**状态转换图**、**扩展巴克斯范式**（EBNF）、有限状态自动机、正规表达式、正规文法等。

**有限状态自动机**、**正规表达式**、**正规文法**是适合于正规语言的描述及处理的形式模型。

（在现实程序设计语言中，几乎任何一种有意义的单词种别对应的单词集合都是正规语言。）（程序设计语言中的单词都能用正规式来定义。）

## 正规文法

|  |  |
| --- | --- |
| 正规文法（3型文法||），其的每一条规则都有下述形式：或，其中，。 | |
|  | （上的）正规集：正规文法所描述的语言。 |

## 正规式

|  |
| --- |
| （上的）正规式（正则表达式）、它所表示的正规集（上的正规集）：  （1）和都是上的正规式，它们所表示的正规集分别为和。  （2）任何，是上的一个正规式，它们所表示的正规集为。  （3）假定和都是上的正规式分别为和，那么，、、和也都是正规式，它们所表示的正规集分别为、、和。  （4）仅由有限次使用上述3个步骤而定义的表达式才是上的正规式。仅由这些正规式所表示的符号串的集合才是上的正规集。 |
| “”读为“或”（也有使用“”代替“”的）； |
| “”读为“连接”，连接符一般可省略不写； |
| “”读为“闭包”（即任意有限次的自重复连接）。 |

在不致混淆时，括号可省去。

算符的优先顺序：“”“”“”。

算符的结合性：“”“”“”是左结合的。

代数规律

|  |  |
| --- | --- |
| “连接”的恒等元素（幺元） |  |
| “连接”的可结合律 |  |
| “或”的抽取律（幂等律） |  |
| “或”的交换律 |  |
| “或”的可结合律 |  |
| “连接”对“或”的分配律 |  |

## 有穷自动机

有穷自动机（有限自动机）

|  |  |  |
| --- | --- | --- |
| 确定的有穷自动机（Deterministic Finite Automata）：一个确定的有穷自动机是一个五元组：，其中：  （1）是一个有穷集。  （2）是一个有穷字母表。  （3）是转换函数，是上的映像。  （4）。  （5），是一个终态集。 | | |
|  | 状态：的每个元素。 | |
|  | 输入符号表：。 | |
|  |  | 输入符号：的每个元素。 |
|  | 将转换函数扩充。设，函数，即，如果输入符号是空串，则仍停留在原来的状态上。符号串（将它表示成的形式，其中，）在DFA上运行：。 | |
|  | 初态：。 | |
|  | 终态（可接受状态、结束状态） | |
|  | 符号串可为DFA所接受：对于中的任何符号串，存在一条从初态结点到某一终态结点的道路，且这条路上所有弧的标记符连接成的符号串等于。空字可为所识别（接受）：的初态结点同时又是终态结点。  符号串可为DFA所接受（识别）：，。 | |
|  | DFA所能接受的符号串的全体（字的全体）（||） | |
|  | 上的一个符号串集是正规的，当且仅当存在一个上的确定有穷自动机，使得。 | |

状态图（状态转换图）

有穷自动机的无用状态：从该自动机的开始状态出发，任何输入串也不能到达的那个状态，或者从这个状态没有通路到达终态。

有穷自动机中状态和状态不等价（状态和状态是可区别的）：

（1）一个是可接受状态，另一个是不可接受状态，或

（2）和不等价。

有穷自动机中状态和状态等价：状态和状态不等价。

|  |  |
| --- | --- |
| 不确定的有穷自动机（Nondeterministic Finite Automata）：一个不确定的有穷自动机是一个五元组：，其中：  （1）是一个有穷集。  （2）是一个有穷字母表。  （3）是一个从到的全体子集的映像，即，其中表示的幂集。  （4），是一个非空初态集。  （5），是一个终态集。 | |
|  | 状态：的每个元素。 |
|  | 输入符号：的每个元素。 |
|  | 符号串在NFA上运行：… |
|  | 符号串可为NFA所识别（读出、接受）：对于中的任何一个串，存在一条从某一初态结点到某一终态结点的道路，且这条道路上所有弧的标记字依序连接成的串（不理睬那些标记为的弧）等于。空字可为所接受：的某些结点既是初态结点又是终态结点，或者存在一条从某个初态结点到某个终态结点的道路。 |
|  | 状态集合的闭包（||）：状态集中的任何状态经任意条弧而能到达的状态的集合。 |
|  | 状态集合的弧转换（||）：所有那些可从中的某一状态经过一条弧而到达的状态的全体。 |

设为一个由不确定的有穷自动机接受的集合，则存在一个接受的确定的有穷自动机。（对于每个NFA，存在一个DFA，使得。）

正规式产生式：将上的一个正规式转换成文法。选择一个非终结符生成类似产生式的形式：，并将定为的识别符号。，的右部含有“”“”“”等正规式符号，不是中的符号。

对于上的NFA，可以构造一个上的正规式，使得。

对于上的每个正规式，可以构造一个上的NFA，使得。

# 语法分析

## 自顶向下

自顶向下分析方法（面向目标的分析方法）

### 确定

确定的自顶向下分析方法：从文法的开始符号出发，考虑如何根据当前的输入符号（单词符号）唯一地确定选用哪个产生式替换相应非终结符以往下推导，或如何构造一棵相应的语法树。

空产生式

文法符号

输入串的结束符（输入串括号||）

|  |
| --- |
| （符号串的）开始符号集（首符号集）（||）：设是上下文无关文法。。若，则规定。 |
| （文法符号的）后跟符号的集合（||）：设是上下文无关文法，，是开始符号。。若，，则。（。若有，则规定。） |
| （产生式的）选择符号集（||）：给定上下文无关文法的产生式，，。若，则；若，则。 |

文法含有左递归的规则（直接左递归）：，，。

文法中含有左递归（间接左递归）：，，，。

文法是左递归的：文法中含有直接左递归或间接左递归。

|  |  |
| --- | --- |
| LL（k）文法：需向前查看k个符号才可确定选用哪个产生式。 | |
| LL（1）：第1个L表明自顶向下分析是从左向右扫描输入串，第2个L表明分析过程中将用最左推导，1表明只需向右看一个符号便可决定如何推导，即选择哪个产生式（规则）进行推导。 | |
|  | 一个上下文无关文法是LL（1）文法的充分必要条件是，对每个非终结符的两个不同产生式，，，满足，其中、不同时能。 |
|  | 文法中不含左公共因子是LL（1）文法的必要条件。 |
|  | 文法中不含左递归是LL（1）文法的必要条件。 |

文法中不含回路：无的推导。

文法中不包含形如的有害规则和的空产生式是文法中不含回路的充分条件。

### 不确定

回溯：在自顶向下分析方法中，假定要被代换的最左非终结符是，且有条规则：，确定用哪个右部去代替。从各种可能的选择中挑选一种，并希望它是正确的。如果以后发现它是错误的，必须退回去，再试另外的选择，这种方式。

带回溯的自顶向下分析：一个试探过程，当分析不成功时则推翻分析，退回到适当位置，再重新试探其余候选可能的推导，这样需要记录已选过的产生式，直到把所有可能的推导序列都试完仍不成功，才能确认输入串不是该文法的句子。

## 自底向上

可规约串：在自底向上的分析方法中，在分析程序工作的每一步，都是从当前串中选择一个子串，将它规约到某个非终结符，这个子串。

（句型相对于非终结符的）短语：令是一个文法，是文法的开始符号，是文法的一个句型。有且，。

（句型相对于规则的）直接短语（简单短语）：特别地，有，。

（句型的）句柄：一个右句型的直接短语。

如果所考虑的文法是无二义的，那么每个右句型有唯一的最右推导，因而其句柄是唯一的；对于二义文法，右句型就可能有多个句柄。

对于无二义文法，一个右句型的唯一句柄是其所有直接短语中最左边的哪一个，对于这种情形，该句型的最左直接短语即是它的句柄。

规范规约：自左向右按句柄规约的过程。（自顶向下最右推导的逆过程。）

### 确定

简单优先分析法

句子括号（||）

|  |  |  |
| --- | --- | --- |
| 简单优先分析法：按照文法符号（终结符和非终结符）的优先关系确定句柄。 | | |
|  | 文法符号的优先性关系（？） | |
|  |  | （文法符号和文法符号的）优先性相等（||）：中存在产生式规则。 |
|  |  | 文法符号的优先性比文法符号优先性大（||）：中存在产生式规则，且。 |
|  |  | 文法符号的优先性比文法符号优先性小（||）：中存在产生式规则，且和。 |
|  | 优先关系矩阵：把文法符号之间的关系用矩阵表示。 | |
| 简单优先文法：  （1）在文法符号集中，任意两个符号之间最多只有一种优先关系成立。  （2）在文法中，任意两个产生式没有相同的右部。 | | |

的优先性小于所有符号，所有符号的优先性大于，当然这里只是对与号有相邻关系的文法符号而言。

直观算符优先分析法

人为地规定算符的优先顺序。

|  |
| --- |
| 算符的优先性低于算符（||） |
| 算符的优先性等于于算符（算符的优先性与算符相同||） |
| 算符的优先性高于于算符（||） |

算符优先分析法

产生式

|  |  |  |  |
| --- | --- | --- | --- |
| 算符文法（operator grammar）（OG文法）：设有文法，中没有形如的产生式，其中和为非终结符，。 | | | |
|  | 性质 | | |
|  |  | 在算符文法中任何句型都不包含两个相邻的非终结符。 | |
|  |  | 如果出现在算符文法的句型中，其中，，则中任何含此的短语必含有。  如果出现在算符文法的句型中，其中，，则中任何含此的短语必含有。 | |
|  | 设是一个不含产生式的算符文法。 | | |
|  |  | ：中含有形如或的产生式。 | |
|  |  | ：中含有形如的产生式，且或。 | |
|  |  | ：中含有形如的产生式，且或。 | |
|  |  | 算符优先文法（operator precedence grammar）（OPG文法）：设有一个不含产生式的算符文法，任意终结符对之间至多只有、和3种关系中的一中成立，。 | |
|  |  |  | ，其中表示中的符号串。 |
|  |  |  | 。 |

素短语：设有文法，其句型的素短语是一个短语，它至少包含一个终结符，并除自身外不包含其它素短语。

最左素短语：最左边的素短语。

LR分析

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |