山东大学
SHANDONG UNIVERSITY

编译原理

第二章 高级语言及其语法描述

授课教师：余仲星
手机：15866821709（微信同号）
邮箱：zhongxing.yu@sdu.edu.cn

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

2.1 程序语言的定义

- 程序语言
  - 语法
  - 语义
  - 语用：主要是有关程序设计技术和语言成份的使用方法，它使语言的基本概念与语言的外界（如数学概念或计算机的对象和操作）联系起来。

2.1.1 语法

- 字符集：又称字母表，是一个有限符号的集合，如大小写英文字母、数字、空白、+-*/、>=<、…。
- 程序语言：可以看做一定字符集上的一个字符串（有限序列）。
- 语法：是一组规则，用它可以形成和产生一个合适的程序。
  - 词法规则：规定了字母表中哪样的字符串是一个单词符号，如0.5是一个实型常数，:=是赋值符号等等；正规式和有限自动机理论是描述词法结构和进行词法分析的有效工具。
  - 语法规则：又称产生式规则，如0.5+x*y代表一个算术式；上下文无关文法是一种描述语法规则的有效工具（但不完备）。

2.1.2 语义

- 语义规则：定义一个程序的意义，如算术表达式是左结合还是右结合等。
- 一个程序语言的基本功能是描述数据和对数据的运算，程序本质上是描述一定数据的处理过程。
  - 逻辑：从数学上考虑每个组成成份。
  - 实现：注重在计算机内的表示和实现的可能性与效率。

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

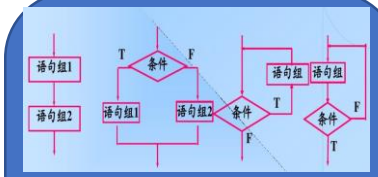
2.2.1 程序语言的发展

1801年，法国人Joseph Jacquard发明使用打孔卡片，控制织布机上的编织图样 。

1946年2月14日，美国宾夕法尼亚大学研制出第一台通用计算机ENIAC，用插线板表示程序。

1956年，IBM推出704计算机，其上的符号汇编程序(SAP)是汇编发展中的一个重要里程碑。

针对汇编语言的缺点，IBM于1954年发布高级程序设计语言Fortran，1956年正式使用。



20世纪60~70年代：结构化设计方法，如Pascal、C等；后用软件工程方法开发更大规模程序。

20世纪80年代：面向对象的程序设计语言Smalltalk问世，OOP立意于模拟现实世界。

20世纪90年代中期：基于可视化和面向对象的编程语言，如VB、VC、Delphi等。

进一步发展：二进制级别软件复用、软件标准件的生产、组态平台（低代码平台）等。

2.2.2 高级语言的分类

- 一、强制式语言(Imperative Language)：又称过程式语言
  - 特点是命令驱动，面向语句。
  - 一个强制式语言由一系列语句组成，每个语句的执行引起若干存储单元中值的改变。
  - 举例：Fortran、C、Pascal、Ada。

- 形式：

语句1；

语句2；

……

语句n；

2.2.2 高级语言的分类

二、应用式语言(Applicative Language)

- 更注重程序所表示的功能，而不是一个接一个语句的执行。
- 程序的开发过程是从前面已有的函数出发构造出更复杂的函数，对初始数据集进行操作直至最终的函数可以用于从初始数据计算出最终的结果。
- 举例：LISP、ML。

形式：

函数n(… 函数2(函数1(数据))…)

2.2.2 高级语言的分类

三、基于规则的语言(Rule-based Language)

- 检查一定的条件，当它满足值，则执行适当的动作。
- 举例：Prolog。

形式：

条件1→动作1

条件2→动作2

……

条件n→动作n

2.2.2 高级语言的分类

- 四、面向对象的语言(Object-Oriented Language)
  - 主要特征是封装性、继承性、多态性。
  - 举例：目前最流行、最重要的语言。

2.2.3 程序结构

一、Fortran

- 一个Fortran程序由一个程序段和若干个辅程序段组成。
- 辅程序段可以是子程序、函数段或数据库。
- 每个程序段由一系列说明语句和执行句组成。

```
PROGRAM MAIN
…
END
SUBROUTINE SUB1
…
END
FUNCTION  FUN1
…
END
```

2.2.3 程序结构

二、Pascal

- 嵌套结构中允许同一个标识符在不同子程序中表示不同的名字。
- 一个子程序B1中说明的名字X只在B1中有效（局部于B1）。
- 如果B2是B1的一个内层子程序，如果B2中对X没有新的说明，则X在B2中依然有效；如果B2对X重新做了说明，则B2中对X的引用都是指重新说明过的X。

```
program main
   …
   procedure  P1;
      …
      procedure P11;
         …
         begin
         …
         end;
      begin
      …
      end;
      procedure P2;
      …
      begin
      …
      end;
  begin
  …
  end.
```

2.2.4 数据类型与操作

数据类型三要素

- 用于区别这种类型的数据对象的属性；
- 这种类型的数据对象可以具有的值；
- 可以作用于这种类型的数据对象的操作。

初等数据类型

- 数值类型：如整数、实数、复数以及这些类型的双长（或多倍长）精度数，对它们施行算术运算（+, -, *, /等）。
- 逻辑数据：多数语言有逻辑型（布尔型）数据，有些甚至有位串数据，对它们可以施行逻辑运算（and, or, not等）。
- 字符数据：有些语言容许有字符型或字符串型数据，这对于符号处理是必须的
- 指针类型：其值指向另一些数据。

2.2.4 数据类型与操作

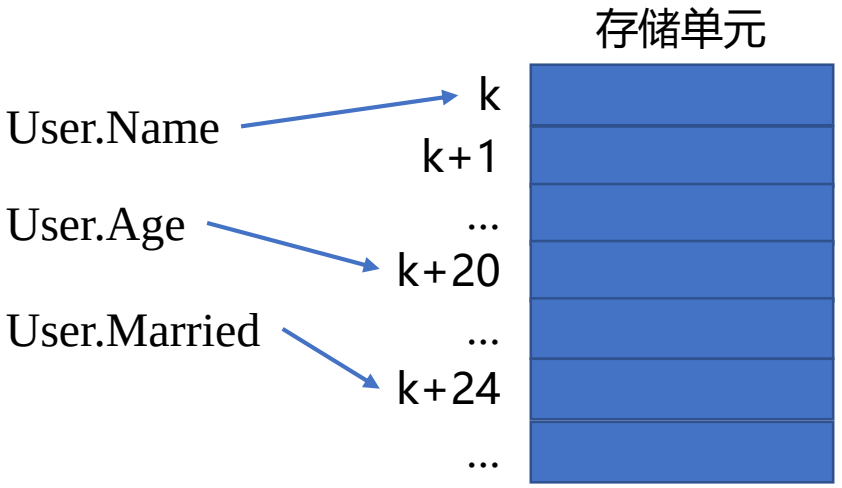
- 数组
  - 是由同一类型数据组成的某种n维矩形结构；
  - 沿着每一维的距离称为一个下标，每维下标只在该维上下限间变动；
  - 数组的每个元素是矩形结构中的一个点，它的位置可以通过每维的下标确定。
- 数组空间
  - 确定数组：存储空间在编译时就可以确定。
  - 可变数组：存储空间在运行时才能确定。
- 数组存储组织
  - 行优先
  - 列优先

2.2.4 数据类型与操作

记录

- 是由已知类型的数据组合起来的一种结构；
- 一个记录结构通常含有若干分量，每个分量称为记录的一个域(field)；
- 每个分量都是一个确定类型的数据，不同分量的数据类型可以不同。

```
struct User
{
    char Name[20];
    int Age;
    bool Married;
};
```



2.2.4 数据类型与操作

- 字符串、表格、栈、队列

- 抽象数据类型（程序包或类）
  - 是数据对象的一个集合；
  - 作用于这些数据对象的抽象运算的集合；
  - 这种类型对象的封装，即除了使用类型中定义的运算外，用户不能对这些对象进行操作。

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

一、表达式

- 表达式：由运算量（又称操作数，即数据引用或函数调用）和算符组成。
  - 一元算符通常写在它的运算量前面，如-X、!b，称为前缀形；也有写后面的，如P↑，称为后缀形。
  - 二元算符一般写在两个运算量中间，如X+Y，称为中缀形式；也有后缀形式如XY+；理论上也有前缀形式+XY。
- 对于多数程序语言来说，表达式的形成规则可概括为
  - 变量（包括下标变量）、常数是表达式；
  - 若E_1、E_2为表达式，θ是一个二元算符，则$E_1\theta E_2$是一个表达式；
  - 若E是表达式，θ为一元算符，则θE（或Eθ）是表达式；
  - 若E是表达式，则(E)是表达式。

一、表达式

- 多数语言的运算顺序和结合性
  - $X + Y * Z \Leftrightarrow X + (Y * Z)$，*优先于+
  - $X - Y - Z \Leftrightarrow (X - Y) - Z$，同级左结合优先
  - $X - Y + Z \Leftrightarrow (X - Y) + Z$，同级左结合优先
  - $X\text{\^{}}Y\text{\^{}}Z \Leftrightarrow X\text{\^{}}(Y\text{\^{}}Z)$，同级右结合优先

一、表达式

- 多数语言的算术算符和逻辑算符的优先级
  - 乘幂（^ 或 ** 或 ↑）
  - 一元负（- 或 @）
  - 乘、除（*, /, ÷）
  - 加、减（+, -）
  - 关系符（<, =, >, <=, <>, >=）
  - 非（! 或 ¬ 或not）
  - 与（&& 或 ∧ 或 and）
  - 或（|| 或 ∨ 或 or）
  - 蕴含（⊃ 或 imp）
  - 等值（值+类型相同，≡ 或 ~ 或 equi）

一、表达式

- 算符的代数性质（交换律、结合律、分配律）常常可用来优化目标程序的质量，但应注意：
  - 交换律一般在计算机上是成立的；
  - 结合律和分配律至少在有效数位上有差别，如：(A+B)+C≠A+(B+C)。

二、语句

- 赋值语句：A:=B
  - 每个名字（A和B）一方面代表其存储单元，另一方面代表单元内容的值；
  - 把一个名字代表的那个单元（地址）称为该名字的左值，一个名字代表的值称为该名字的右值；
  - 变量一般既持有左值又持有右值，常数和带有算符的表达式一般认为只持有右值；
  - 赋值号左边的变量必须左值，出现在赋值号右边的表达式只需持有右值。

二、语句

控制语句

- 无条件转移语句：goto L
- 条件语句：if B then S

  if B then S_1 else S_2
- 循环语句：while B do S

  repeat S until B

  for i := E_1 step E_2 until E_3 do S
- 过程调用语句：call P(X_1, X_2, …, X_n)
- 返回语句：return (E)

二、语句

- 说明语句
  - 旨在定义名字的性质，如：int i
- 简单句和复合句
  - 简单句指那些不包含其它语句成份的基本句，如赋值语句、goto语句等。
  - 复合句指那些句中有句的语句，如条件语句、循环语句等。

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

基本概念

- 设Σ是一个有穷字母表，它的每个元素称为一个符号。
- Σ上的一个符号串是指由Σ中的符号所构成的一个有穷序列。
- 不包含任何符号的序列称为空字（空串），记为ε（/'epsɪlɒn/）。
- 用Σ^*表示Σ上所有符号串的全体，空字ε也包括在其中，称为Σ 的闭包。
  - $\Sigma = \{a, b\}$，则$\Sigma^* = \{\varepsilon, a, b, aa, ab, bb, aaa, \ldots\}$。
- 用ϕ表示不含任何元素的空集{}。
  - 注意ε、{}、{ε}的区别。

基本概念

- Σ^*的子集U和V的连接（积）定义为：$UV = \{\alpha\beta | \alpha \in U, \beta \in V\}$(逗号表示与)
  - 一般而言，$UV \neq VU$，但$(UV)W = U(VW)$。
- V自身的n次连接（积）记为：$V^n = \underbrace{VV \ldots V}_{n}$。
  - 规定：$V^0 = \{\varepsilon\}$。
- V的闭包：$V^* = V^0 \cup V^1 \cup V^2 \cup \cdots$。
  - 闭包V^*中的每个符号串都是由V中的符号串经过有限次连接而成的，即闭包中每个字符串长度有限。
- V的正则闭包（正闭包）：$V^+ = VV^*$。

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

2.3.1 上下文无关文法

- 上下文无关文法：它所定义的文法范畴（或语法单位）是完全独立于这种范畴可能出现的环境的。
  - 当碰到一个算术表达式，我们完全可以对它“就事论事”的进行处理，而不必考虑它的上下文。
  - 在自然语言中，一个句子、一个词乃至一个字，它的语法性质和所处的上下文往往都有密切关系。

2.3.1 上下文无关文法

【例2.1】He gave me a book.

- <句子> → <主语> <谓语> <间接宾语> <直接宾语>.
- <主语> → <代词>
- <谓语> → <动词>
- <间接宾语> → <代词>
- <直接宾语> → <冠词> <名词>
- <代词> → He
- <代词> → me
- <冠词> → a
- <动词> → gave
- <名词> → book

语法树

- 反复利用规则，将→左边的乘法替换成右边，推导出句子：

<句子>⇒ <主语> <谓语> <间接宾语> <直接宾语>.

⇒ <代词> <谓语> <间接宾语> <直接宾语>.

⇒ He <谓语> <间接宾语> <直接宾语>.

⇒ He <动词> <间接宾语> <直接宾语>.

⇒ He gave <间接宾语> <直接宾语>.

⇒ He gave <代词> <直接宾语>.

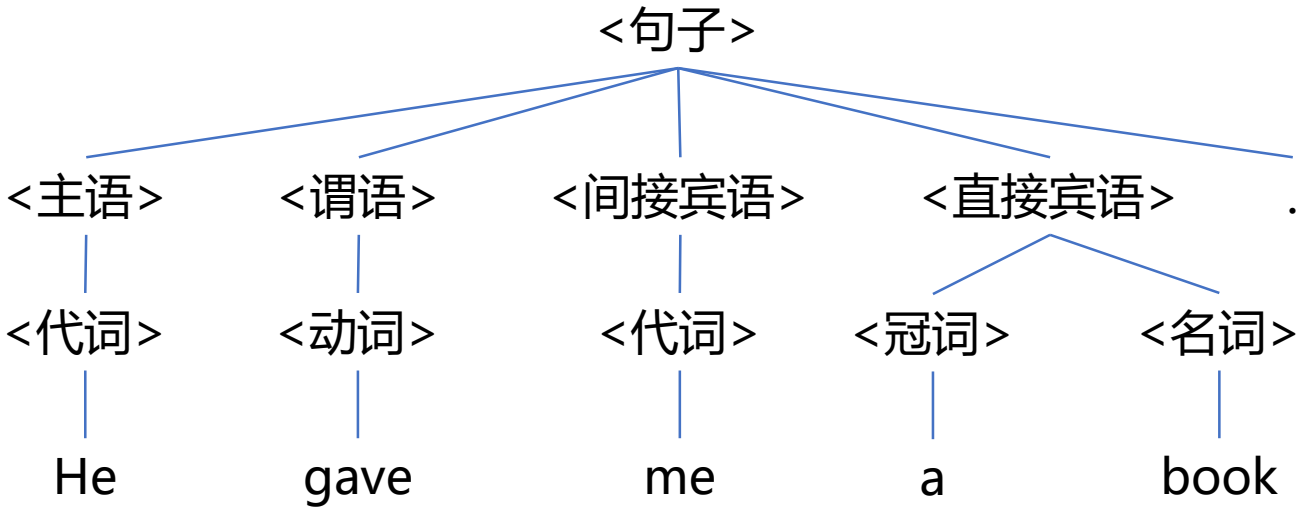
⇒ He gave me <直接宾语>.

⇒ He gave me <冠词> <名词>.

⇒ He gave me a <名词>.

⇒ He gave me a book.

推导



- 上下文无关文法要素
  - 终结符号：是组成语言的基本符号，是语言的一个不可再分的单位。
  - 非终结符号：也称语法变量，用来代表语法范畴，是一个类的记号，而不是个体记号。
  - 开始符号：是一个特殊的非终结符号，代表所定义的语言中我们最感兴趣的语法范畴。
  - 产生式：也称产生规则或简称规则，是定义语法范畴的一种书写规则。

产生式

- 产生式：$A \to \alpha$
  - 左边的A是一个非终结符号，称为产生式的左部符号。
  - 右边的α是由终结符号和/或非终结符号组成的一个符号串，称为产生式的右部
  - 有时也说$A \to \alpha$是关于A的一条产生规则。
  - 有的书上，→也用::=表示，这种表示方法也称为巴科斯范式 (BNF)。

【例2.2】递归产生式定义加乘算术表达式

$E \to i$

$E \to E + E$

$E \to E * E$

$E \to (E)$

形式化定义

- 形式化定义：一个上下文无关文法G是一个四元式$(V_T, V_N, S, \mathcal{P})$
  - V_T是一个非空有限集合，它的每个元素称为终结符号；
  - V_N是一个非空有限集合，它的每个元素称为非终结符号，$V_T \cap V_N = \phi$；
  - S是一个非终结符号，称为开始符号；
  - $\mathcal{P}$是一个产生式集合（有限），每个产生式的形式是$P \to \alpha$，其中$P \in V_N, \alpha \in (V_T \cup V_N)^*$，开始符号$S$至少在某个产生式左部出现一次。
- 为书写方便，经常采用如下简写，每个α_i也称为是P的一个候选式
  - $\left.\begin{array}{l} P \to \alpha_1 \\ P \to \alpha_2 \\ \quad \ldots\ldots \\ P \to \alpha_n \end{array}\right\} \Leftrightarrow P \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$
- 箭头 → 读为“定义为”，直竖| 读为“或”，它们是元语言符号。

形式化定义

约定

- 用大写字母A、B、C…，或带尖括号的词组如<算术表达式>，代表非终结符号；
- 用小写字母a、b、c…代表终结符号；
- 用希腊字母α、β、γ等代表由终结符号和非终结符号组成的字符串。
- 为简便起见，当引用具体文法的例子时，仅列出产生式和指出开始符号。

【例2.3】开始符号是E

$E \to i \mid EAE$

$A \to + \mid *$

推导

- 【例2.4】有如下文法：$E \to E+E \mid E*E \mid (E) \mid i$
  - 推导：$E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (i+E) \Rightarrow (i+i)$
  - 这个推导提供了一个证明，证明$(i+i)$是这个文法所定义的一个算术表达式。
- 我们称$\alpha A\beta$直接推导出$\alpha\gamma\beta$，即$\alpha A\beta \Rightarrow \alpha\gamma\beta$，当且仅当$A \to \gamma$是一个产生式，且$\alpha$、$\beta \in (V_T \cup V_N)^*$
  - 如果$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$，则称这个序列是从$\alpha_1$到$\alpha_n$的一个推导；
  - 若存在一个从α_1到α_n的推导，则称α_1可推导出α_n，其逆过程称为归约；
  - 用$\alpha_1 \overset{+}{\Rightarrow} \alpha_n$表示从$\alpha_1$出发，经一步或若干步，可推导出$\alpha_n$；
  - 用$\alpha_1 \overset{*}{\Rightarrow} \alpha_n$表示从$\alpha_1$出发，经0步或若干步，可推导出$\alpha_n$，即$\alpha_1 = \alpha_n$或$\alpha_1 \overset{+}{\Rightarrow} \alpha_n$

语言

- 假定G是一个文法，S是它的开始符号，如果$S \overset{*}{\Rightarrow} \alpha$，则称$\alpha$是一个句型。
- 如果一个句型中只包含终结符号，则称其为一个句子。
- 文法G所产生的句子的全体是一个语言，记为：$L(G) = \{\alpha | S \overset{+}{\Rightarrow} \alpha, \alpha \in V_T^*\}$。
- 【例2.5】有文法G：$E \to E + E \mid E * E \mid (E) \mid i$
  - $(i * i + i)$是该文法的一个句子，因为有推导：$E \Rightarrow (E) \Rightarrow (E + E) \Rightarrow (E * E + E) \Rightarrow (i * E + E) \Rightarrow (i * i + E) \Rightarrow (i * i + i)$；
  - $E, (E), (E + E), (i * E + E), \ldots, (i * i + i), i * i + i$都是这个文法的句型。

语言

- 一个句型到另一个句型的推导过程往往不唯一：$E \to E+E \mid E*E \mid (E) \mid i$
  - $E \Rightarrow E+E \Rightarrow i+E \Rightarrow i+i$
  - $E \Rightarrow E+E \Rightarrow E+i \Rightarrow i+i$
- 约束
  - 若推导过程中，总是最先替换最右(左)的非终结符，则称为最右(左)推导；
  - 若归约过程中，总是最先归约最右(左)的非终结符，则称为最右(左)归约。
- 规范
  - 句型的最右推导称为规范推导，其逆过程最左规约称为规范归约；
  - $i*i+i$的规范推导：$E \Rightarrow E*E \Rightarrow E*E+E \Rightarrow E*E+i \Rightarrow E*i+i \Rightarrow i*i+i$

语言

- 【例2.6】有文法$G_1 = (V_T, V_N, S, \mathcal{P})$，其中：

$V_N = \{<\text{数字串}>, <\text{数字}>\}; V_T = \{0,1,2,3,4,5,6,7,8,9\};$

$\mathcal{P} = \{<\text{数字串}> \rightarrow <\text{数字串}><\text{数字}> \mid <\text{数字}>,$

$<\text{数字}> \rightarrow 0|1|2|3|4|5|6|7|8|9\};$

$S = \{<\text{数字串}>\}$

确定G_1对应的语言。

  - 【分析】由<数字串>→<数字>，可得数字串为0~9的任意数字

    每次用<数字串>→<数字串><数字>推导，末尾就增加一个0~9的数字，直到使用<数字串>→<数字>推导为止。

  - 【结论】$L(G_1)$表示十进制非负整数。

语言

- 【例2.7】有文法$G_2[S]: S \to bA, A \to aA|a$，确定$G_2$对应的语言。
  - $S \Rightarrow bA \Rightarrow ba$
  - $S \Rightarrow bA \Rightarrow baA \Rightarrow baa$
  - $S \Rightarrow bA \Rightarrow baA \Rightarrow baaA \Rightarrow baaa$
  - …
  - $S \Rightarrow bA \Rightarrow baA \Rightarrow \cdots \Rightarrow ba \ldots a$
  - 归纳得：$L(G_2) = \{ba^n | n \geq 1\}$

语言

- 【例2.8】有文法$G_3[S]: S \to AB, A \to aA|a, B \to bB|b$，确定$G_3$对应的语言。
  - $S \Rightarrow AB \Rightarrow ab$
  - $S \Rightarrow AB \Rightarrow aAB \Rightarrow aaB \Rightarrow aab$
  - …
  - $S \Rightarrow AB \Rightarrow AB \Rightarrow aB \Rightarrow abB \Rightarrow abb$
  - $S \Rightarrow AB \Rightarrow AB \Rightarrow aB \Rightarrow abB \Rightarrow abbB \Rightarrow abbb$
  - …
  - $S \Rightarrow AB \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow a \ldots aB \Rightarrow a \ldots abB \Rightarrow a \ldots ab \ldots b$
  - 归纳得：$L(G_3) = \{a^m b^n | m \geq 1, n \geq 1\}$

语言

- 【例2.9】构造文法：$L(G_4)=\{a^nb^n|n\geq 1\}$。
  - ab
  - $aabb$
  - $aaabbb$
  - …
  - 归纳得：$G_4[S]: S\rightarrow aSb|ab$

语言

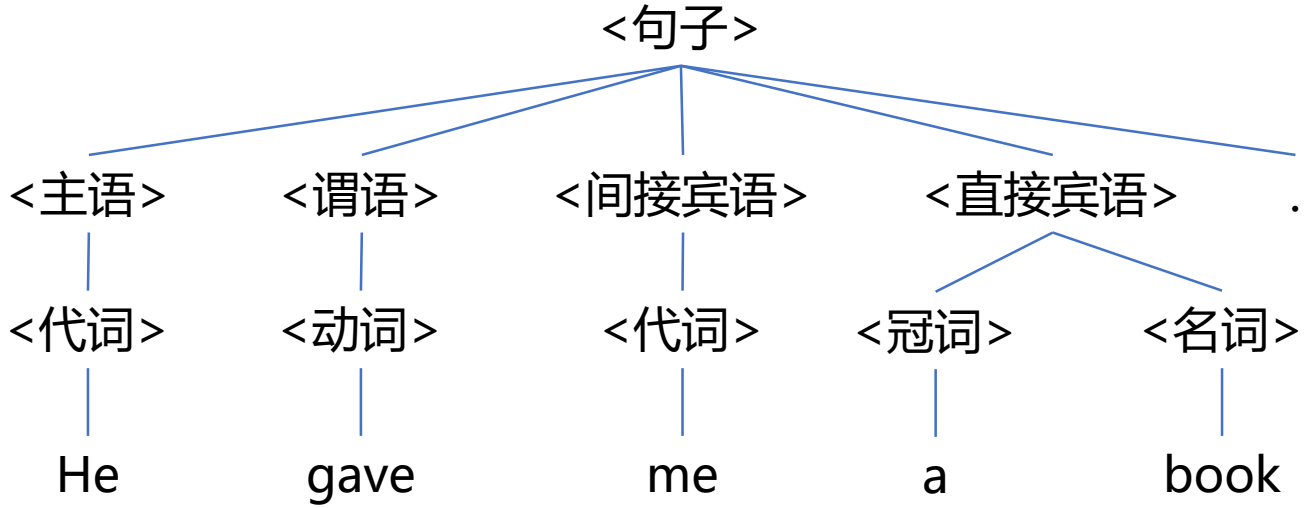
- 【例2.10】构造文法G_5，使其描述的语言为正奇数集合。
- 【分析】正奇数要求，要么是一位奇数数字，要么是以奇数数字结尾的十进制数字。
- 【解】令$G_5 = (V_T, V_N, <\text{正奇数}>, \mathcal{P})$; $V_T = \{0,1,2,3,4,5,6,7,8,9\}$
  - $\mathcal{P}$: $<\text{一位奇数}> \rightarrow 1|3|5|7|9$; $<\text{一位数字}> \rightarrow <\text{一位奇数}> |0|2|4|6|8$
  - $<\text{正奇数}> \rightarrow <\text{一位奇数}> | <\text{数字串}><\text{一位奇数}>$
  - $<\text{数字串}> \rightarrow <\text{一位数字}> | <\text{数字串}><\text{一位数字}>$
  - $V_N = \{<\text{正奇数}>, <\text{数字串}>, <\text{一位数字}>, <\text{一位奇数}>\}$

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

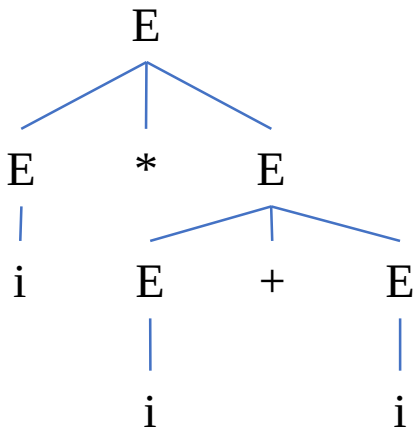
语法分析树

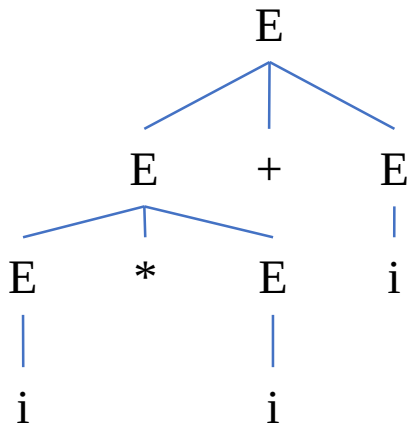
- 语法分析树，简称语法树，即用树形图表示一个句型的推导过程。
  - 一棵语法树表示了句型的种种可能的不同推导过程（但未必是全部），包括最左（最右）推导，即一棵语法树是不同推导过程的共性抽象。
  - 如果坚持使用最左（最右）推导，那么一棵语法树就完全等价于一个最左（最右）推导。



二义文法

- 二义文法：如果一个文法的某个句子对应两棵不同的语法树，即其最左（最右）推导不唯一，称该文法为二义文法。
- 【例2.11】$E \to E+E \mid E*E \mid (E) \mid i$，关于句子i*i+i的最右推导：
  - $E \Rightarrow E*E \Rightarrow E*E+E \Rightarrow E*E+i \Rightarrow E*i+i \Rightarrow i*i+i$
  - $E \Rightarrow E+E \Rightarrow E+i \Rightarrow E*E+i \Rightarrow E*i+i \Rightarrow i*i+i$





二义文法

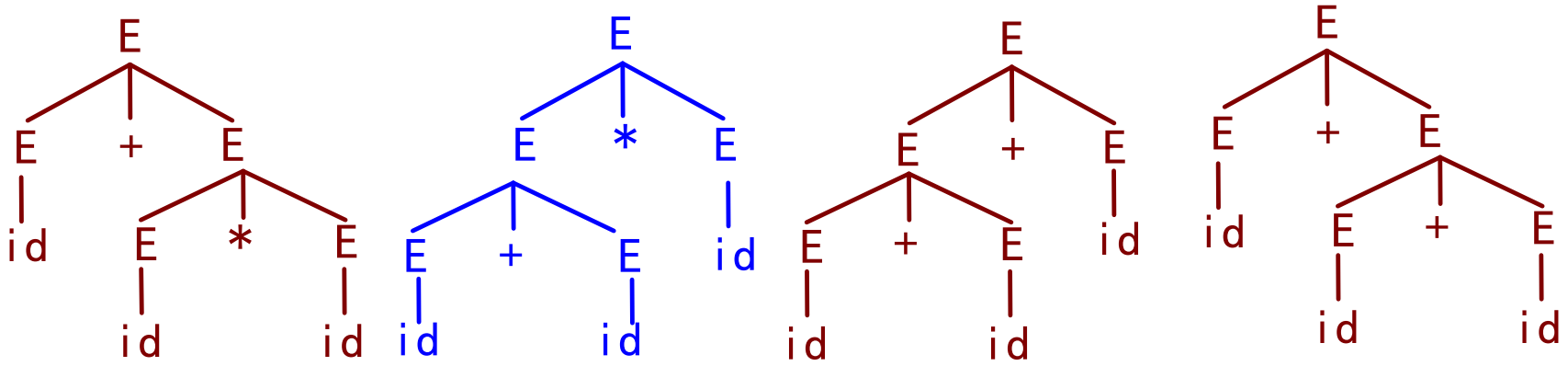
- 文法的二义性和语言的二义性是不同的概念
  - 可能有两个不同的文法G和G'，其中一个是二义的而另一个是无二义的，但是有$L(G) = L(G')$；
  - 对程序设计语言来说，常常希望它的文法是无二义的，因为我们希望对它每个语句的分析是唯一的；
  - 但是，只要能控制和驾驭文法的二义性，有时候存在二义性并不一定是坏事；
  - 目前已经证明，二义性问题是不可判定的，即不存在一个算法，它能在有限步骤内确切的判定一个文法是否为二义性的。

二义文法

句子id+id*id和id+id+id可能的语法树：

E→ E + E | E * E | (E) | -E | id

(1) (2) (3) (4) (5)



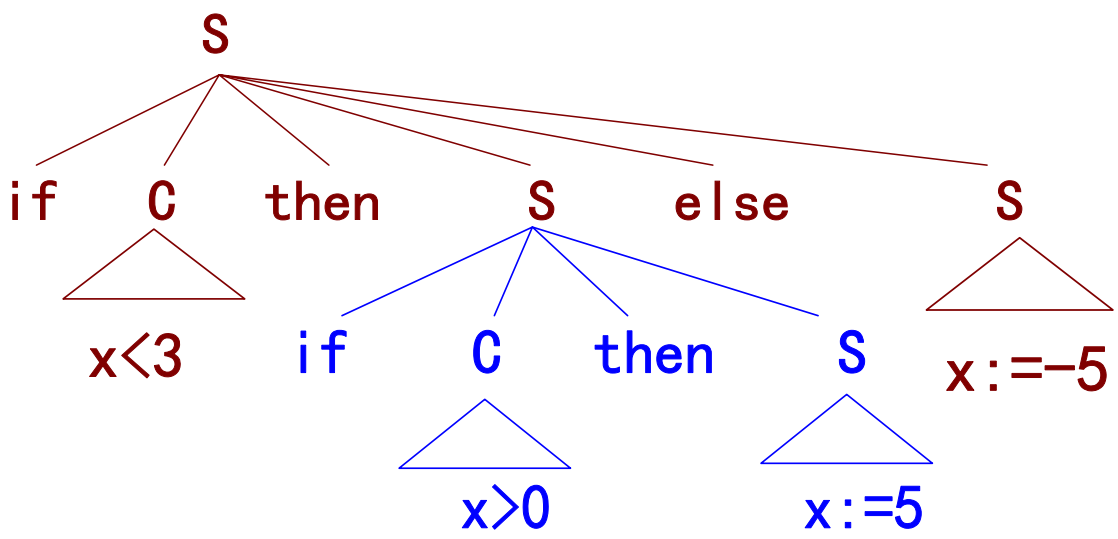
*优先级高　　+优先级高　　+左结合　　+右结合

原因：文法中缺少对文法符号优先级和结合性的规定，在产生句子的过程中某些推导有多于一种选择

“悬空（dangling）else”问题

S → if C then S　　(1)
　| if C then S else S　　(2)
　| id := E　　(3)
C → E = E | E < E | E > E　　(4)...(6)
E → E + E | - E | id | n　　(7)...(10)

例 条件语句 if x<3 then if x>0 then x:=5 else x:=-5



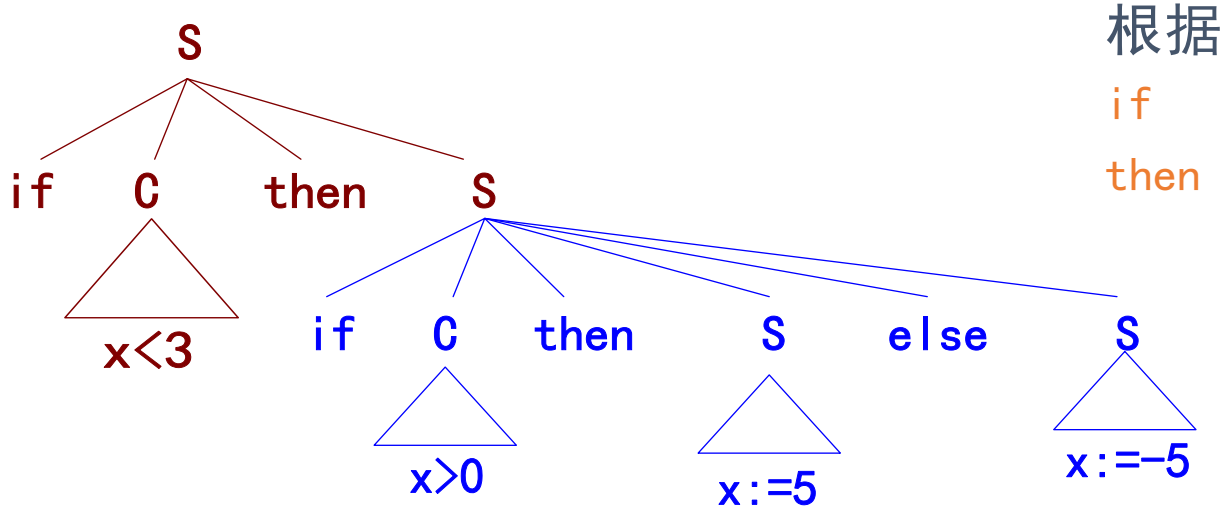
根据产生式(2)，有
if　x<3
then　if x>0 then x:=5
else　x:=-5

else与离它远的then匹配

“悬空（dangling）else”问题

S → if C then S　　(1)
　| if C then S else S　　(2)
　| id := E　　(3)　　(G3.3)
C → E = E | E < E | E > E　　(4)...(6)
E → E + E | - E | id | n　　(7)...(10)

例 条件语句 if x<3 then if x>0 then x:=5 else x:=-5



根据产生式(1)，有

if　x<3
then　if　x>0
　　then　x:=5
　　else　x:=-5

else与离它近的then匹配

二义性的消除

改写二义文法为非二义文法

等价的非二义文法：

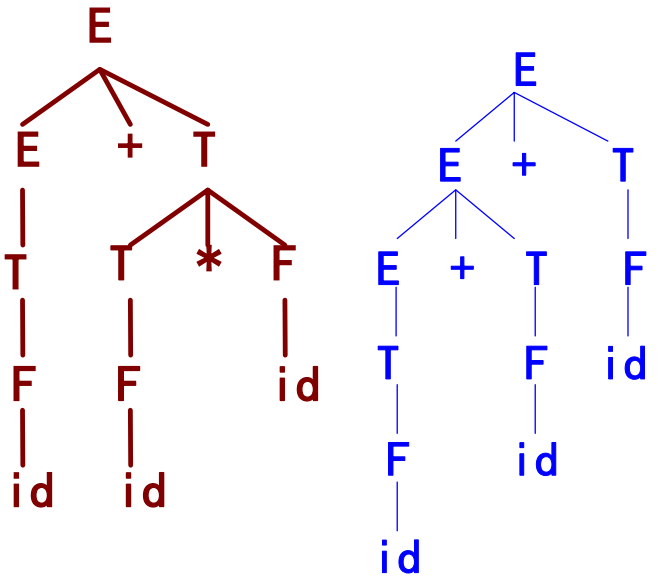
$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid -F \mid id$

$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid id$

再推导id+id*id和id+id+id：

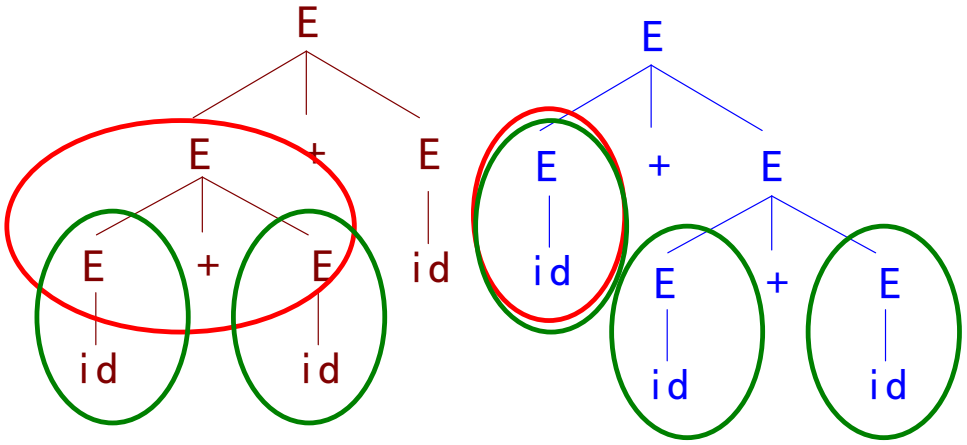


问题：如何将二义文法改写为非二义文法？

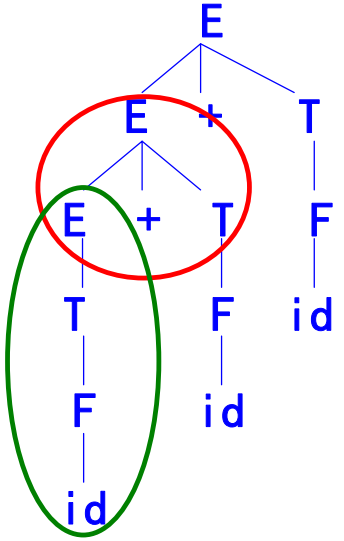
可以看出新文法的特点

1. 新引入的非终结符，限制了每一步直接推导均有唯一选择；
2. 最终语法树的形状，仅与文法有关，而与推导方法无关；
3. 非终结符的引入，增加了推导步骤（语法树增高）；

句子 id+id+id 的语法树



二义文法的语法树



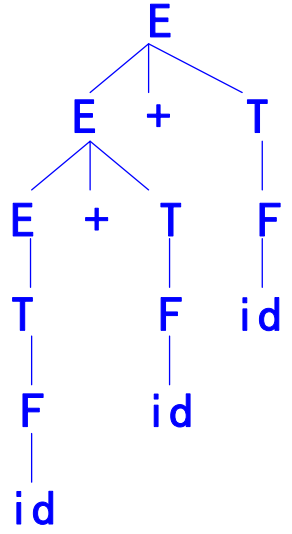
新文法的语法树

可以看出新文法的特点

1. 新引入的非终结符，限制了每一步直接推导均有唯一选择；
2. 最终语法树的形状，仅与文法有关，而与推导方法无关；
3. 非终结符的引入，增加了推导步骤（语法树增高）；
4. 越接近S的文法符号的优先级越低；
5. 对于A→αAβ，其右部中，若A在终结符a左边出现（即β中包含a），则终结符a具有左结合性质。

等价的非二义文法：

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid -F \mid id$$



文法 的语法树

可以看出 G3.4 的特点

1. 新引入的非终结符，限制了每一步直接推导均有唯一选择；
2. 最终语法树的形状，仅与文法有关，而与推导方法无关；
3. 非终结符的引入，增加了推导步骤（语法树增高）；
4. 越接近S的文法符号的优先级越低；
5. 对于A→αAβ，其右部中，若A在终结符a左边出现（即β中包含a），则终结符a具有左结合性质。

等价的非二义文法：

E → E + T | T

T → T * F | F

F → (E) | -F | id

改写二义文法的关键步骤：

1. 引入一个新的非终结符，增加一个子结构并提高一级优先级；
2. 递归非终结符在终结符左边，使该终结符具有左结合性，否则具有右结合性。

改写二义文法

① 引入一个新的非终结符，增加一个子结构并提高一级优先级；

② 递归非终结符在终结符左边，运算具有左结合性，否则具有右结合性。

1. 优先级：{+}　{*}　{( ), −, id }
2. 结合性：左结合：+, *
   右结合：−
   无结合：id ( )
3. 非终结符与运算：

E: +　　(E产生式，左递归)
T: *　　(T产生式，左递归)
F: −, ( ), id　(F产生式，右递归)

E→ E + E
| E * E
| (E)　(
| − E
| id

E → E + T | T
T → T * F | F
F → −F | (E) | id

再讨论“悬空else”问题

if-then-else和if-then：在一个复合if语句中，可能then多于else，使得else不知与哪个then结合。

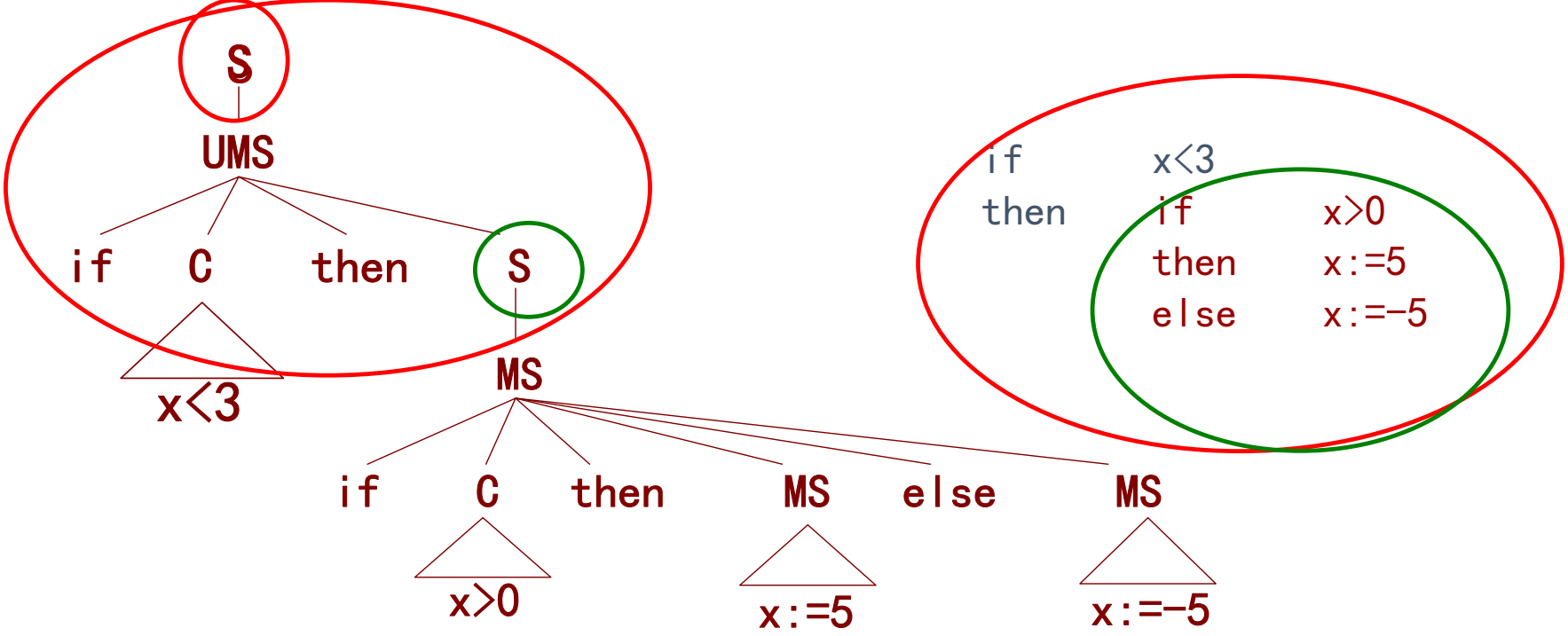
一般原则：else与其左边最靠近的then结合，即右结合。

改写文法的关键：是将S分为完全匹配（MS）和不完全匹配（UMS）两类，并且在UMS中规定else右结合。

```
S   → MS                                  (1)
    | UMS                                 (2)
MS  → if C then MS else MS    (3)
    | id := E                 (4)
UMS → if C then S                         (5)
    | if C then MS else UMS   (6)
```

```
S → if C then S
  | if C then S else S
  | id := E
C → E=E | E<E | E>E
E → E+E | -E | id | n
```

```
C  → E = E | E < E | E > E  (7)...(9)
E  → E + T | T              (10)...(11)
T  → -T | id | n                  (12)...(14)
```



S → MS (1)
| UMS (2)

MS → if C then MS else MS (3)
| id := E (4)

UMS→ if C then S (5)
| if C then MS else UMS (6)

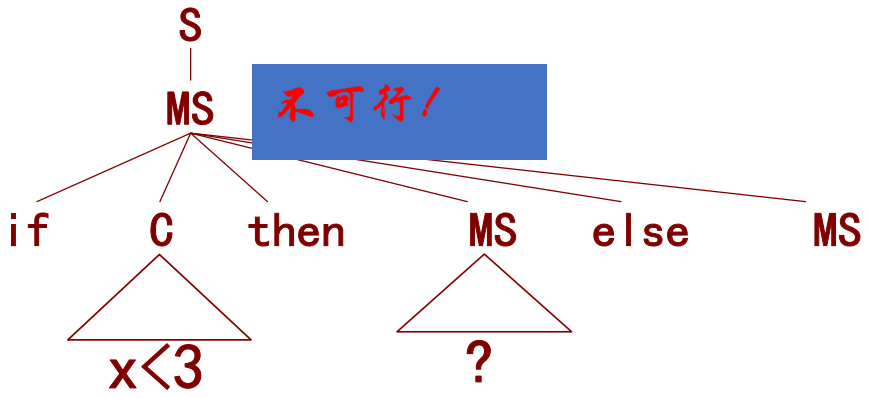
C → E = E | E < E | E > E (7)...(9)

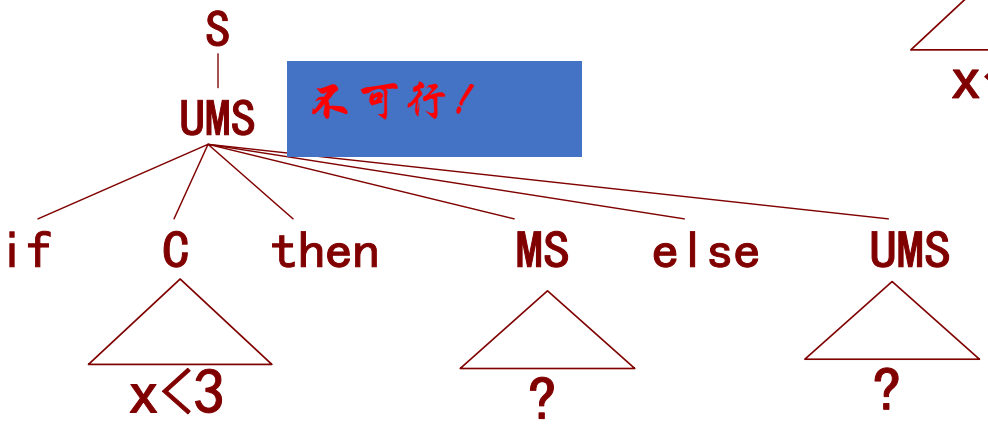
E → E + T | T (10)...(11)

T → -T | id | n (12)...(14)

```
if      x<3
then    if x>0 then x:=5
else    x:=-5
```

不可能！





S	→ MS	(1)
	| UMS	(2)
MS	→ if C then MS else MS	(3)
	| id := E	(4)
UMS	→ if C then S	(5)
	| if C then MS else UMS	(6)

约定

- 对上下文无关文法，对其施加以下限制，满足这两个条件的文法也称化简了的文法。
  - 文法不含产生式$P \to P$，因为这种产生除了引起二义性外没有任何用处。
  - 每个非终结符P必须都有用处，这意味着必须存在推导：$(1) S \overset{*}{\Rightarrow} \alpha P \beta$，以及$(2)\ P \overset{+}{\Rightarrow} \gamma, \gamma \in V_T^*$。

第二章 高级语言及其语法描述

- 2.1 程序语言的定义
  - 2.1.1 语法
  - 2.1.2 语义
- 2.2 高级语言的一般特性
  - 2.2.1 程序语言的发展
  - 2.2.2 高级语言的分类
  - 2.2.3 程序结构
  - 2.2.4 数据类型与操作
  - 2.2.5 语句与控制结构
- 2.3 程序语言的语法描述
  - 2.3.1 上下文无关文法
  - 2.3.2 语法分析树与二义性
  - 2.3.3 形式语言鸟瞰

2.3.3 形式语言鸟瞰

- Chomsky于1956年建立了形式语言的描述，并将文法划分为4种类型。
- 0型文法：我们说文法$G = (V_T, V_N, S, \mathcal{P})$是一个0型文法，如果它的每个产生式$\alpha \to \beta$满足：$\alpha \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$，$\beta \in (V_N \cup V_T)^*$。
  - 0型文法也称短语文法。
  - 0型文法的能力相当于图灵(Turing)机，或者说任何0型语言都是递归可枚举的；反之，递归可枚举必定是一个0型语言。
- 图灵机相关知识参考
  - 对一个句子，可以做出某语言接受或拒绝该句子的判断，这个语言称为图灵可判断语言。
  - 除了接受、拒绝，如果还存在不停机可能，称为图灵可识别语言。
  - 图灵可识别语言等价于递归可枚举语言，同时也被认为是半可判定的，它是可以被图灵机识别的。

2.3.3 形式语言鸟瞰

- 0型文法：我们说文法$G=(V_T,V_N,S,\mathcal{P})$是一个0型文法，如果它的每个产生式$\alpha\to\beta$满足：$\alpha\in(V_N\cup V_T)^*V_N(V_N\cup V_T)^*$，$\beta\in(V_N\cup V_T)^*$。
- 1型文法：在满足0型文法基础上，除$S\to\varepsilon$外，每个产生式$\alpha\to\beta$满足$|\alpha|\leq|\beta|$，且S不能出现在任何产生式的右部。
  - 1型文法也称上下文有关文法，即对非终结符号进行替换时必须考虑上下文，并且一般不允许替换成空串ε。
  - 例如，假如$\alpha A\beta\to\alpha\gamma\beta$是1型文法的一个产生式，$\alpha$和$\beta$均不空，则非终结符号$A$只有在$\alpha$和$\beta$这个上下文环境中才能替换为$\gamma$。

2.3.3 形式语言鸟瞰

- 0型文法：我们说文法$G = (V_T, V_N, S, \mathcal{P})$是一个0型文法，如果它的每个产生式$\alpha \to \beta$满足：$\alpha \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$，$\beta \in (V_N \cup V_T)^*$。
- 2型文法：在满足0型文法基础上，每个产生式满足：$A \to \beta, A \in V_N, \beta \in (V_N \cup V_T)^*$。
  - 2型文法也称上下文无关文法。
  - 上下文无关文法对应下推自动机，使用下推表（先进后出栈）的有限自动机是分析上下文无关文法的基本手段。

2.3.3 形式语言鸟瞰

- 0型文法：我们说文法$G=(V_T,V_N,S,\mathcal{P})$是一个0型文法，如果它的每个产生式$\alpha\to\beta$满足：$\alpha\in(V_N\cup V_T)^*V_N(V_N\cup V_T)^*$，$\beta\in(V_N\cup V_T)^*$。
- 3型文法：在满足0型文法基础上，每个产生式满足：$A\to\alpha B$或$A\to\alpha$，其中，$\alpha\in V_T^*,A\in V_N,B\in V_N$。
  - 3型文法也称右线性文法。
  - 3型文法还有另外一种形式，称为左线性文法，如果产生式形式为：$A\to B\alpha$或$A\to\alpha$，其中，$\alpha\in V_T^*,A\in V_N,B\in V_N$。
  - 3型文法等价于正规式，所以也成为正规文法。
  - 正规式就是正则表达式，英文为Regular Expression；正规文法也称为正则文法，英文为Regular Grammar。

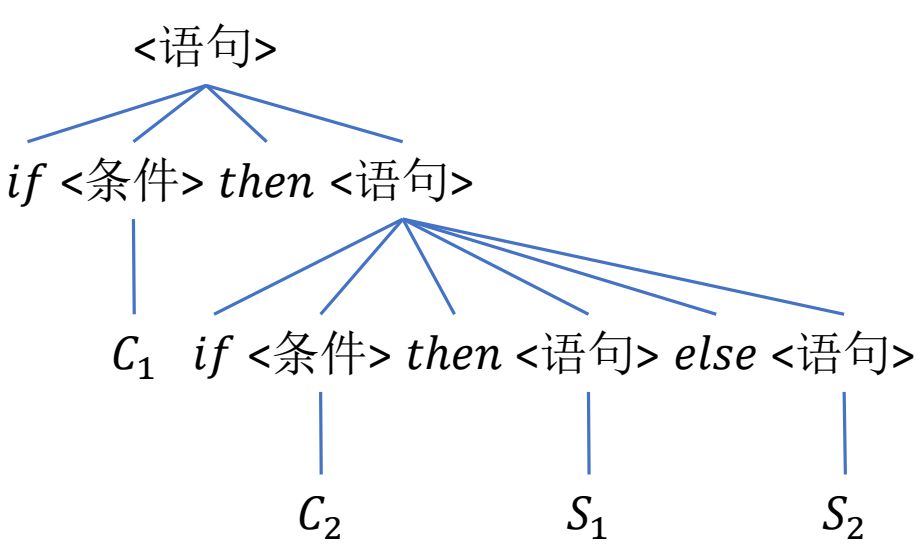
几个有趣的结论

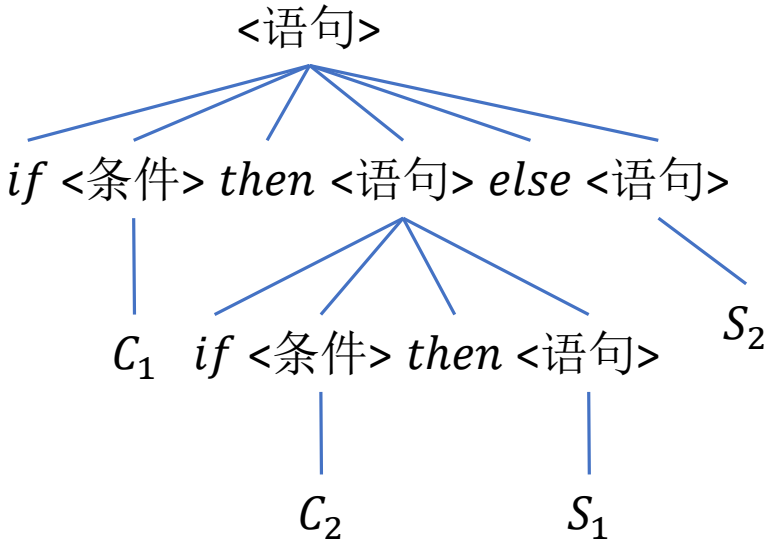
- 正规文法(3)不能产生语言$L(G)=\{a^n b^n | n \geq 1\}$，上下文无关文法(2)则可以：$S \rightarrow aSb|ab$。
- $L(G)=\{a^n b^n c^i | i \geq 1, n \geq 1\}$是一个上下文无关语言：$S \rightarrow AB, A \rightarrow aAb|ab,\ B \rightarrow Bc|c$。
- 语言$L(G)=\{a^n b^n c^n | n \geq 1\}$只能用上下文有关文法(1)产生：
  - $S \rightarrow aSBA|abB,$
  - $BA \rightarrow BA',\quad BA' \rightarrow AA',\quad AA' \rightarrow AB,$
  - $bA \rightarrow bb,\quad bB \rightarrow bc,\quad cB \rightarrow cc$
- 语言$L(G)=\{\alpha c \alpha | \alpha \in (a|b)^*\}$只能用0型文法产生。

几个有趣的结论

- 上下文无关文法表示条件语句：
  - $< 语句 > \rightarrow if < 条件 > then < 语句 >$

    $| if < 条件 > then < 语句 > else < 语句 >$
  - 这是个二义文法，如句子：$if\ C_1\ then\ if\ C_2\ then\ S_1\ else\ S_2$





几个有趣的结论

- 上下文无关文法表示条件语句：
  - $<\text{语句}>\rightarrow if<\text{条件}>then<\text{语句}>$
    $|\, if<\text{条件}>then<\text{语句}>else<\text{语句}>$
  - 这是个二义文法，如句子：$if\ C_1\ then\ if\ C_2\ then\ S_1\ else\ S_2$
- 一般语言都规定：else必须匹配最后那个未得到匹配的then，即就近匹配
  - $<\text{语句}>\rightarrow<\text{匹配句}>|<\text{非匹配句}>$
  - $<\text{匹配句}>\rightarrow if<\text{条件}>then<\text{匹配句}>else<\text{匹配句}>$
    $|<\text{其它语句}>$
  - $<\text{非匹配句}>\rightarrow if<\text{条件}>then<\text{语句}>$
    $|\, if<\text{条件}>then<\text{匹配句}>else<\text{非匹配句}>$

第二章作业

【作业2-1】令文法为

$$E \to T|E+T|E-T$$

$$T \to F|T*F|T/F$$

$$F \to (E)|i$$

(1) 给出$i+i*i$、$i*(i+i)$的最左推导和最右推导。

(2) 给出$i+i+i$、$i+i*i$、$i-i-i$的语法树。

【作业2-2】证明下面的文法是二义的：$S \to iSeS|iS|i$

【作业2-3】把下面的文法改为无二义的：$S \to SS|(S)|()$

【作业2-4】给出下面语言的相应文法

$L_1 = \{a^n b^n c^i | n \geq 1, i \geq 1\}$

$L_2 = \{a^i b^n c^n | n \geq 1, i \geq 0\}$

$L_3 = \{a^n b^n a^m b^m | m \geq 0, n \geq 0\}$

$L_4 = \{1^n 0^m 1^m 0^m | m \geq 0, n \geq 0\}$