编译原理

第3讲 高级程序设计语言的语法描述

编译原理

高级程序设计语言的语法描述
——文法

高级语言及其语法描述

- 程序语言的定义
- 高级语言的一般特性
- 程序语言的语法描述

文法

- 文法： 描述语言的语法结构的形式规则
- 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
<名词> → book
<冠词> → a
<动词> → gave

文法

1. <句子> → <主语><谓语><间接宾语><直接宾语>
2. <主语> → <代词>
3. <谓语> → <动词>
4. <间接宾语> → <代词>
5. <直接宾语> → <冠词> <名词>
6. <代词> → He
7. <代词> → me
8. <名词> → book
9. <冠词> → a
10. <动词> → gave

<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

编译原理

高级程序设计语言的语法描述
——语法描述的几个基本概念

语法描述的几个基本概念

- 字母表：一个有穷字符集，记为∑
- 字母表中每个元素称为字符
- ∑上的字(也叫字符串）是指由∑中的字符所构成的一个有穷序列
- 不包含任何字符的序列称为空字，记为ε
- 用∑*表示∑上的所有字的全体，包含空字ε
- 例如: 设 ∑={a， b}，则

$$\sum^{*}=\{\varepsilon,a,b,aa,ab,ba,bb,aaa,...\}$$

语法描述的几个基本概念

- $\sum^*$的子集U和V的连接（积）定义为

$$UV = \{ \alpha\beta \mid \alpha \in U \ \& \ \beta \in V \}$$

- 示例：设

$U = \{ a, aa \}$

$V = \{ b, bb \}$

$UV = \{ ab, abb, aab, aabb \}$

语法描述的几个基本概念

- $\sum^*$的子集U和V的连接（积）定义为

$$UV = \{ \alpha\beta \mid \alpha \in U \ \& \ \beta \in V \}$$

- V自身的 n次积记为

$$V^n = \underbrace{V\ V \ldots V}_{n\text{个}}$$

- $V^0 = \{\varepsilon\}$
- V^*是V的闭包：$V^* = V^0 \cup V^1 \cup V^2 \cup V^3 \cup \ldots$
- V^+是V的正规闭包：$V^+ = V\ V^*$

语法描述的几个基本概念

- 设:

  $U = \{ a, aa \}$

- 那么:

  $U^* = \{ \varepsilon, a, aa, aaa, aaaa, \ldots \}$

  $U^+ = \{ a, aa, aaa, aaaa, \ldots \}$

编译原理

高级程序设计语言的语法描述
——上下文无关文法

上下文无关

<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
<名词> → book
<冠词> → a
<动词> → gave

- 上下文无关文法G是一个四元组

  $G=(V_T, V_N, S, P)$，其中
  - V_T：终结符(Terminal)集合(非空)
  - V_N：非终结符(Noterminal)集合(非空)，且$V_T \cap V_N=\varnothing$
  - S：文法的开始符号，$S \in V_N$
  - P：产生式集合(有限)，每个产生式形式为
    - $P \rightarrow \alpha$， $P \in V_N$， $\alpha \in (V_T \cup V_N)^*$
  - 开始符S至少必须在某个产生式的左部出现一次

上下文无关文法

- 例，定义只含+，*的算术表达式的文法

  G=< {i，+，*，(，)}，{E}，E，P >，其中，P由下列产生式组成：

  $E \rightarrow i$

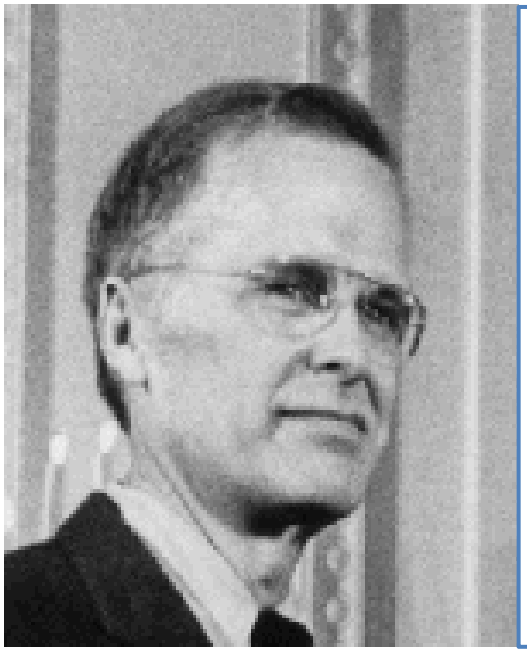
  $E \rightarrow E+E$

  $E \rightarrow E*E$

  $E \rightarrow (E)$

上下文无关文法

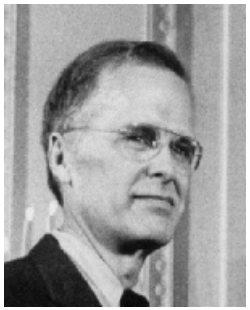
- 巴科斯范式(BNF)
  - “→”用“::="表示



John W. Backus

For profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for seminal publication of formal procedures for the specification of programming languages.

巴科斯范式(BNF)



John W. Backus

首次在ALGOL 58中使用这种记号系统描述语法

Peter Naur

在ALGOL 60中发展并简化
命名Backus Normal Form

Donald E. Knuth

主张称为巴斯科-诺尔范式(Backus–Naur Form)
认为它不算是一种正规形式(Normal Form)

上下文无关文法

- 约定

$$P \to \alpha_1$$
$$P \to \alpha_2$$
$$\cdots$$
$$P \to \alpha_n$$

可缩写为 $\Rightarrow$ $P \to \alpha_1|\alpha_2|\ldots|\alpha_n$

  - 其中，“|”读成“或”，称α_i为P的一个候选式
  - 表示一个文法时，通常只给出开始符号和产生式

文法G=<{i，+，*，(，)}，{E}，E，P>，P定义如下：

$E \to i$

$E \to E+E$

$E \to E*E$

$E \to (E)$

G(E)：$E \to i \mid E+E \mid E*E \mid (E)$

编译原理

高级程序设计语言的语法描述
——文法生成语言

推导

<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

- 定义：称$\alpha A\beta$直接推出$\alpha\gamma\beta$，即

$$\alpha A\beta \Rightarrow \alpha\gamma\beta$$

仅当$A \to \gamma$是一个产生式，且α， $\beta \in (V_T \cup V_N)^*$ 。

- 如果$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$，则我们称这个序列是从$\alpha_1$到$\alpha_n$的一个推导。若存在一个从$\alpha_1$到$\alpha_n$的推导，则称$\alpha_1$可以推导出$\alpha_n$ 。

- 对文法G(E)： $E \to i \mid E+E \mid E*E \mid (E)$

$$E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (i+E) \Rightarrow (i+i)$$

推导

<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

$\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ 从α_1出发，经过0步或若干步推出α_n

$\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ 从α_1出发，经过1步或若干步推出α_n

$\alpha \overset{*}{\Rightarrow} \beta$ 即 $\alpha = \beta$或$\alpha \overset{+}{\Rightarrow} \beta$

<句子>$\overset{*}{\Rightarrow}$He gave me a book

<句子>$\overset{+}{\Rightarrow}$He gave me a book

He gave <间接宾语><直接宾语>$\overset{+}{\Rightarrow}$He gave me <冠词><名词>

句型、句子和语言

<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

- 假定G是一个文法，S 是它的开始符号。
- 如果$S \overset{*}{\Rightarrow} \alpha$，则称α是一个句型。
- 仅含终结符号的句型是一个句子。
- 文法G所产生的句子的全体是一个语言，记为 L(G):

$$L(G) = \left\{ \alpha \middle| S \overset{+}{\Rightarrow} \alpha,\ \alpha \in \mathrm{V_T^*} \right\}$$

编译原理

高级程序设计语言的语法描述
——句型和句子练习

句型、句子和语言

- 请证明 (i*i+i)是文法
  G(E)： E → i | E+E | E*E | (E)
  的一个句子。

$$S \overset{+}{\Rightarrow} \alpha, \alpha \in V_T^*$$

句型、句子和语言

- 请证明 (i*i+i)是文法

  G(E)： $E \rightarrow i \mid E+E \mid E*E \mid (E)$

  的一个句子。

证明：

$E \Rightarrow (E)$

$\Rightarrow (E+E)$

$\Rightarrow (E*E+E)$

$\Rightarrow (i*E+E)$

$\Rightarrow (i*i+E)$

$\Rightarrow (i*i+i)$

(i*i+i)是文法G的句子

E，(E)，(E*E+E)，…，(i*i+i)是句型。

编译原理

高级程序设计语言的语法描述
——从文法到语言

句型、句子和语言

- 设文法 $G_1(A)$:

  $A \rightarrow c \mid Ab$

  $G_1(A)$产生的语言是什么?

- 以c开头，后继若干个b
- $L(G_1)$={c，cb，cbb，…}

$A \Rightarrow c$

$A \Rightarrow Ab$

$\Rightarrow cb$

$A \Rightarrow Ab$

$\Rightarrow Abb$

$\Rightarrow Abbb$

$\Rightarrow \ldots$

$\Rightarrow Ab\ldots b$

$\Rightarrow cb\ldots b$

句型、句子和语言

- 设文法$G_2(S)$:
  $S \rightarrow AB$
  $A \rightarrow aA|a$
  $B \rightarrow bB|b$

$G_2(S)$产生的语言是什么?

$L(G_2)=\{a^m b^n | m, n>0\}$

$S \Rightarrow AB$
$A \Rightarrow a$
$A \Rightarrow aA$
$\Rightarrow aaA$
$\Rightarrow aaaA$
$\Rightarrow \ldots$
$\Rightarrow a\ldots aA$
$\Rightarrow a\ldots aa$
$B \Rightarrow b$
$B \Rightarrow bB$
$\Rightarrow bbB$
$\Rightarrow bbbB$
$\Rightarrow \ldots$
$\Rightarrow b\ldots bB$
$\Rightarrow b\ldots bb$

编译原理

高级程序设计语言的语法描述
——从语言到文法

句型、句子和语言

- 请给出产生语言为$\{a^n b^n | n \geq 1\}$的文法

  $G_3(S)$:

  $S \rightarrow aSb$

  $S \rightarrow ab$

- 计算思维的典型方法--递归
  - 问题的解决又依赖于类似问题的解决，只不过后者的复杂程度或规模较原来的问题更小
  - 一旦将问题的复杂程度和规模化简到足够小时，问题的解法其实非常简单

上下文无关文法示例

- 请给出产生语言为$\{a^m b^n | 1 \le n \le m \le 2n\}$的文法 $G_4(S)$:

$S \rightarrow ab \mid aab$

$S \rightarrow aSb \mid aaSb$

编译原理

高级程序设计语言的语法描述
——推导与语法树

最左推导和最右推导

- 从一个句型到另一个句型的推导往往不唯一

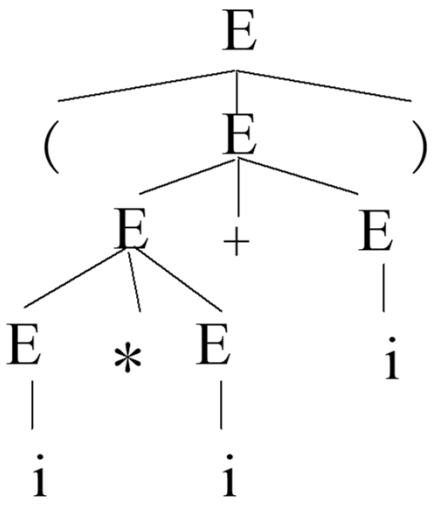
$$E+E \Rightarrow i+E \Rightarrow i+I$$

$$E+E \Rightarrow E+i \Rightarrow i+i$$

- 最左推导：任何一步$\alpha \Rightarrow \beta$都是对α中的最左非终结符进行替换
- 最右推导：任何一步$\alpha \Rightarrow \beta$都是对α中的最右非终结符进行替换

语法树

- 用一张图表示一个句型的推导,称为语法树
- 一棵语法树是不同推导过程的共性抽象



G(E): $E \rightarrow i \mid E+E \mid E*E \mid (E)$

(i*i+i)

$E \Rightarrow (E)$
$\Rightarrow (E+E)$
$\Rightarrow (E*E+E)$
$\Rightarrow (i*E+E)$
$\Rightarrow (i*i+E)$
$\Rightarrow (i*i+i)$

$E \Rightarrow (E)$
$\Rightarrow (E+E)$
$\Rightarrow (E+i)$
$\Rightarrow (E*E+i)$
$\Rightarrow (E*i+i)$
$\Rightarrow (i*i+i)$

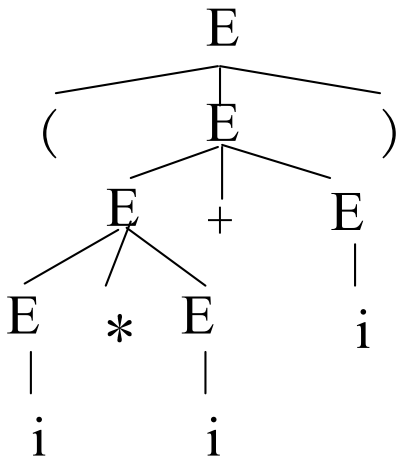
编译原理

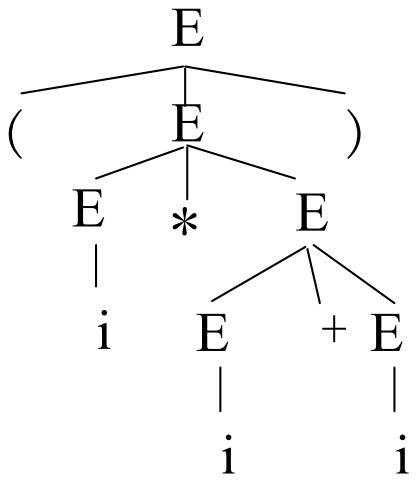
高级程序设计语言的语法描述
——语法树与二义性

语法树与二义性(ambiguity)

- 一个句型是否只对应唯一一棵语法树?
- (i*i+i)

G(E): $E \rightarrow i|E+E|E*E|(E)$
是二义的(ambiguous)





语法树与二义性(ambiguity)

- 文法的二义性：如果一个文法存在某个句子对应两棵不同的语法树，则说这个文法是二义的

  G(E)： $E \rightarrow i|E+E|E*E|(E)$ 是二义文法

- 语言的二义性：一个语言是二义的，如果对它不存在无二义的文法
  - 对于语言L，可能存在G和G’，使得L(G)=L(G’)=L，有可能其中一个文法为二义的，另一个为无二义的

语言的二义性

John saw Mary in a boat.

语言的二义性

二义文法G(E)：

$E \rightarrow i \mid E+E \mid E*E \mid (E)$

无二义文法G'(E)：

$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

$F \rightarrow (E) \mid i$

表达式 → 项 | 表达式 + 项

项 → 因子 | 项 * 因子

因子 → (表达式) | i

语言的二义性

无二义文法G'(E):

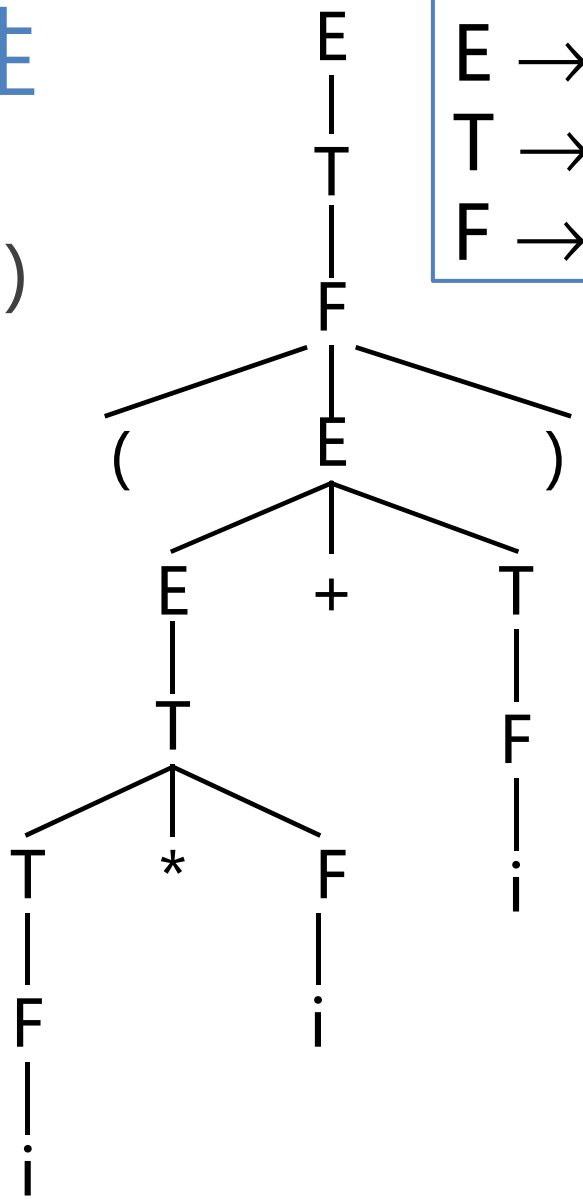
$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

$F \rightarrow (E) \mid i$

- 考虑句子(i*i+i)

$E \Rightarrow T$
$\Rightarrow F$
$\Rightarrow (E)$
$\Rightarrow (E+T)$
$\Rightarrow (T+T)$
$\Rightarrow (T*F+T)$
$\Rightarrow (F*F+T)$
$\Rightarrow (i*F+T)$
$\Rightarrow (i*i+T)$
$\Rightarrow (i*i+F)$
$\Rightarrow (i*i+i)$



语法树与二义性(ambiguity)

- 文法的二义性：如果一个文法存在某个句子对应两棵不同的语法树，则说这个文法是二义的

  G(E)： $E \rightarrow i|E+E|E*E|(E)$ 是二义文法
- 语言的二义性：一个语言是二义的，如果对它不存在无二义的文法
  - 对于语言L，可能存在G和G’，使得L(G)=L(G’)=L，有可能其中一个文法为二义的，另一个为无二义的
- 二义性问题是不可判定问题，即不存在一个算法，它能在有限步骤内，确切地判定一个文法是否是二义的
- 可以找到一组无二义文法的充分条件

编译原理

高级程序设计语言的语法描述
——形式语言鸟瞰

形式语言鸟瞰

- 乔姆斯基(Chomsky)是美国当代有重大影响的语言学家
- www.chomsky.info

w3

形式语言鸟瞰

- 乔姆斯基于1956年建立形式语言体系，他把文法分成四种类型：0，1，2，3型
- 与上下文无关文法一样，它们都由四部分组成，但对产生式的限制有所不同
  - $G=(V_T，V_N，S，P)$
    - V_T：终结符(Terminal)集合(非空)
    - V_N：非终结符(Noterminal)集合(非空)，且$V_T \cap V_N=\varnothing$
    - S：文法的开始符号，$S \in V_N$
    - P：产生式集合(有限)

w3 Chomsky图片

wang, 2009/8/27

形式语言鸟瞰

$P \to \alpha$， $P \in V_N$， $\alpha \in (V_T \cup V_N)^*$

- 0型(短语文法，图灵机)
  - 产生式形如： $\alpha \to \beta$
  - 其中：$\alpha \in (V_T \cup V_N)^*$且至少含有一个非终结符；$\beta \in (V_T \cup V_N)^*$
- 1型(上下文有关文法，线性界限自动机)
  - 产生式形如： $\alpha \to \beta$
  - 其中：$|\alpha| \leq |\beta|$，仅 $S \to \varepsilon$ 例外

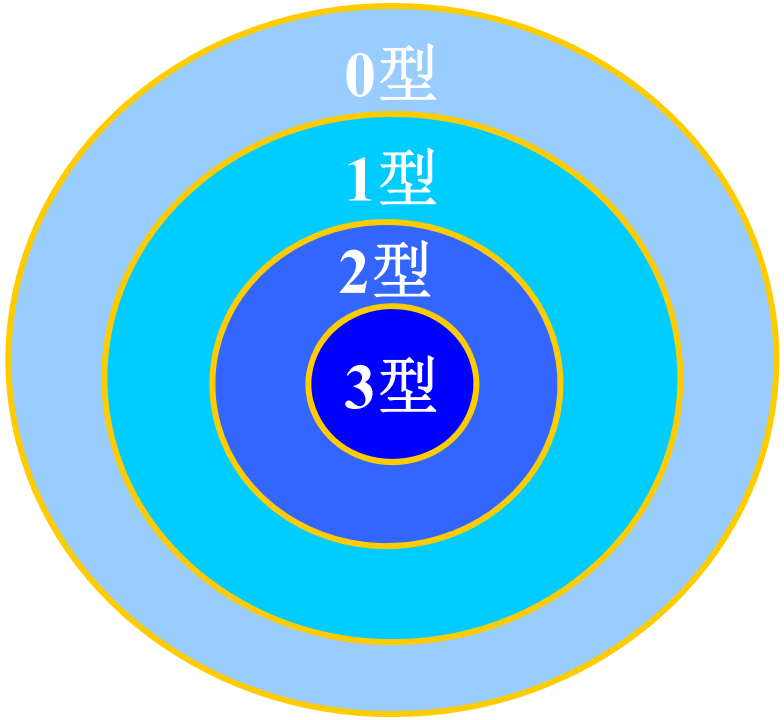
形式语言鸟瞰

- 2型(上下文无关文法，非确定下推自动机)
  - 产生式形如： $A \to \beta$
  - 其中：$A \in V_N$；$\beta \in (V_T \cup V_N)^*$
- 3型(正规文法，有限自动机)
  - 产生式形如： $A \to \alpha B$ 或 $A \to \alpha$
  - 其中： $\alpha \in V_T^*$； A，$B \in V_N$
  - 产生式形如： $A \to B\alpha$ 或 $A \to \alpha$
  - 其中： $\alpha \in V_T^*$； A，$B \in V_N$

右线性文法

左线性文法

四种类型文法描述能力比较



上下文无关文法 与上下文有关文法

- $L_5=\{a^nb^n|n\geq 1\}$ 不能由正规文法产生，但可由上下文无关文法产生

  $G_5(S)$:

  $S \rightarrow aSb|\ ab$

上下文无关文法 与上下文有关文法

- $L_6=\{a^nb^nc^n|n\geq 1\}$不能由上下文无关文法产生，但可由上下文有关文法产生

$G_6(S)$: $S \rightarrow aSBC|\ aBC$

$CB \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

S
⇒aSBC
⇒ aaSBCBC
⇒ aaaBCBCBC
⇒ aaaBBCCBC
⇒ aaaBBCBCC
⇒ aaaBBBCCC
⇒ aaabBBCCC
⇒ aaabbBCCC
⇒ aaabbbCCC
⇒ aaabbbcCC
⇒ aaabbbccC
⇒ aaabbbccc

四种类型描述能力比较

- 计算思维的典型方法
  - 理论可实现 vs. 实际可实现
  - 理论研究重在探寻问题求解的方法，对于理论成果的研究运用又需要在能力和运用中作出**权衡**

- 程序设计语言不是上下文无关
  上下文有关语言
- $L_7=\{\alpha c\alpha \mid \alpha\in\{a,b\}^*\}$不能由上下文无关文法产生，甚至连上下文有关文法也不能产生，只能由0型文法产生
  - 标识符引用
  - 过程调用过程中，“形-实参数的对应性”(如个数，顺序和类型一致性)
- 对于现今程序设计语言，在编译程序中，仍然采用上下文无关文法来描述其语言结构

编译原理

高级程序设计语言的语法描述
——小结

小结

- 文法、推导
  - 文法⇔语言
  - 最左推导、最右推导
- 语法树
- 二义性
  - 文法的二义性、语言的二义性
- 乔姆斯基形式语言体系