编译原理

第2章 文法和语言的基本知识

高级语言及其语法描述

问题：如何描述语言?

- 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

例：如何判断英文句子是正确的？

“HUST celebrates the 70th anniversary.”

HUST celebrates the 70th anniversary

- <句子> → <主语><谓语><宾语>
- <主语> → <名词>
- <谓语> → <动词>
- <宾语> → <定语><名词>
- <定语> → <冠词><序数词>
- <名词> → **HUST**
- <名词> → **anniversary**
- <动词> → **celebrates**
- <冠词> → **the**
- <序数词> →**70th**

HUST celebrates the 70th anniversary

<句子> → <主语><谓语><宾语>
<主语> → <名词>
<谓语> → <动词>
<宾语> → <定语><名词>
<定语> → <冠词><序数词>
<名词> → HUST
<名词> → anniversary
<动词> → celebrates
<冠词> → the
<序数词> →70th

- <句子>
- ⇒<主语><谓语><宾语>
- ⇒<名词><谓语><宾语>
- ⇒ **HUST** <谓语><宾语>
- ⇒ **HUST** <动词><宾语>
- ⇒ **HUST celebrates** <宾语>
- ⇒ **HUST celebrates** <定语><名词>
- ⇒ **HUST celebrates** <冠词><序数词><名词>
- ⇒ **HUST celebrates the** <序数词><名词>
- ⇒ **HUST celebrates the 70th** <名词>
- ⇒ **HUST celebrates the 70th anniversary**

编译原理

第2章 文法和语言的基本知识
—字母表和符号串的基本概念

语法描述的几个基本概念

- 字母表：元素的非空有穷集合。
- 符号（字符）：字母表中每个元素称为字符。
- 字符串（字）：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符号串，记为ε，长度|ε|=0
- 例子: 设字母表∑={a,b,c}， 则有字符串 a,b,c,aa,ab,ac,ba,bb,bc,…,aaa,…

符号串的运算

- 连接：x和y是符号串，则串xy称为它们的连结。
  - 例子：设x=ab,y=11,则xy=ab11
- 集合的乘积：A和B是符号串的集合，则A和B的乘积 $AB=\{ xy \mid x\in A,y\in B \}$
  - 例子：设A＝{ a, aa } B＝ { b, bb }
  - AB= { *ab, abb, aab, aabb*}

符号串的运算

- X是符号串，x的幂运算定义为：

  $x^0 = \varepsilon$　　$x^n = \underbrace{xx\ldots x}_{n} = x\,x^{n-1} (n>0)$

  $x^1 = x$

- A自身的n次积记为　$A^n = \underbrace{A\,A\ldots A}_{n个}$
- $A^0 = \{\varepsilon\}$
- A^*是A的闭包：　$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \ldots$
- A^+是A的正闭包：$A^+ = A\,A^*$

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

语法描述的几个基本概念

- 例子：

  设$U=\{a, aa\}$

  - 那么：

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

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

编译原理

第2章 文法和语言的基本知识
—文法和语言的形式定义

上下文无关文法

<句子> → <主语><谓语><宾语>
<主语> → <名词>
<谓语> → <动词>
<宾语> → <定语><名词>
<定语> → <冠词><序数词>
<名词> → HUST
<名词> → anniversary
<动词> → celebrates
<冠词> → the
<序数词> →70th

- 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
    - $A \to \alpha$，$A \in V_N$，$\alpha \in (V_T \cup V_N)^*$
  - 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 >，

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
首次在ALGOL 58中使用这种记号系统描述语法

上下文无关文法

- 约定

$$P \to \alpha_1$$
$$P \to \alpha_2$$
$$\dots$$
$$P \to \alpha_n$$

可缩写为 $P \to \alpha_1|\alpha_2|\dots|\alpha_n$

  - 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：

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

缩写

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
—语言的形式定义

推导

<句子> → <主语><谓语><宾语>
<主语> → <名词>
<谓语> → <动词>
<宾语> → <序数词><名词>
<名词> → HUST
<名词> → anniversary
<动词> → celebrates
<序数词> →70th

- 直接推导：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，$\beta \in (V_T \cup V_N)^*$。

- 推导：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$\alpha_n$。

- 例子：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><宾语>
<主语> → <名词>
<谓语> → <动词>
<宾语> → <定语><名词>
<定语> → <冠词><序数词>
<名词> → HUST
<名词> → anniversary
<动词> → celebrates
<冠词> → the
<序数词> →70th

广义推导：$\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ 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$\alpha \overset{+}{\Rightarrow} \beta$

例子：<句子>$\overset{*}{\Rightarrow}$HUST celebrates the 70th anniversary

<句子>$\overset{+}{\Rightarrow}$ HUST celebrates the 70th anniversary

HUST celebrates <序数词><名词>

$\overset{+}{\Rightarrow}$ HUST celebrates the 70th <名词>

句型、句子和语言

<句子> → <主语><谓语><宾语>
<主语> → <名词>
<谓语> → <动词>
<宾语> → <定语><名词>
<定语> → <冠词><序数词>
<名词> → HUST
<名词> → anniversary
<动词> → celebrates
<冠词> → the
<序数词> →70th

- 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。
- 语言:文法**G**所产生的句子的全体是一个语言，记为 **L(G):**

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

例子（从文法到语言）

- 请证明 (i*i+i)是文法

  G(E)：　E → i | E+E | E*E | (E)的一个句子。

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

例子（从文法到语言）

- 请证明 (i*i+i)是文法

  G(E)： E → i | E+E | E*E | (E)的一个句子。

证明：

E ⇒ (E)
⇒ (E+E)
⇒ (E*E+E)
⇒ (i*E+E)
⇒ (i*i+E)
⇒ (i*i+i)

(i*i+i)是文法G的句子

E，(E)，(E*E+E)，…，(i*i+i)是句型。

例子（从文法到语言）

- 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，…}

$A \Rightarrow c$

$A \Rightarrow Ab$

$\Rightarrow cb$

$A \Rightarrow Ab$

$\Rightarrow Abb$

$\Rightarrow Abbb$

$\Rightarrow \ldots$

$\Rightarrow Ab\ldots b$

$\Rightarrow cb\ldots b$

例子（从文法到语言）

- 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?

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

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

例子（从语言到文法）

- 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

- 请给出产生语言为$\{a^nb^n|n\geq 1\}$的文法?

  $G_3(S)$:

  $S \rightarrow aSb$

  $S \rightarrow ab$

例子（从语言到文法）

- 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
——规范推导和规范规约

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

规约

推导的逆过程

若用$\dot{\Rightarrow}$表示归约，设A→α是文法G中的一个规则，则我们有：

$$xAy \Rightarrow x\alpha y \qquad x\alpha y \dot{\Rightarrow} xAy$$

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

$$(i+i) \dot{\Rightarrow} (E+i) \dot{\Rightarrow} (E+E) \dot{\Rightarrow} (E) \dot{\Rightarrow} E$$

短语

- G[s]是一个文法,假定αβδ是文法G的一个句型，如果有

$$S \overset{*}{\Rightarrow} \alpha A \delta \qquad A \overset{+}{\Rightarrow} \beta$$

则称 β是相对于非终结符A的，句型αβδ的**短语**。

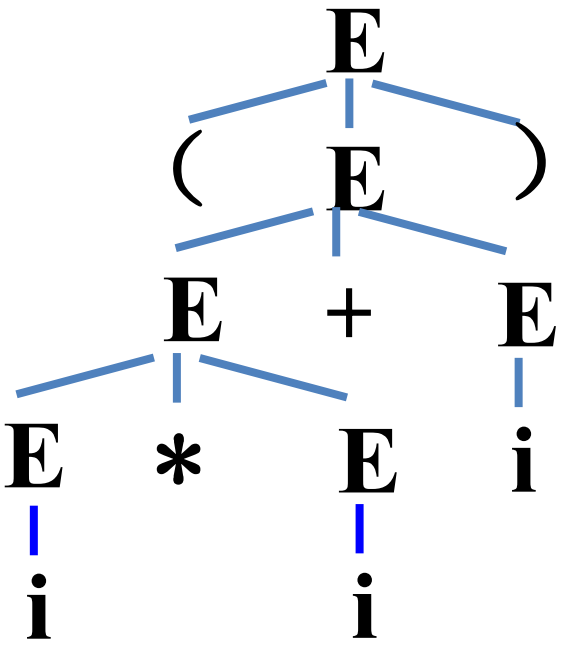
$$S \overset{*}{\Rightarrow} \alpha A \delta \qquad A \Rightarrow \beta$$

则称 β是相对于非终结符A的，句型αβδ的**直接短语**。

一个句型的最左直接短语称为该句型的**句柄**

语法树

- 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)

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

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

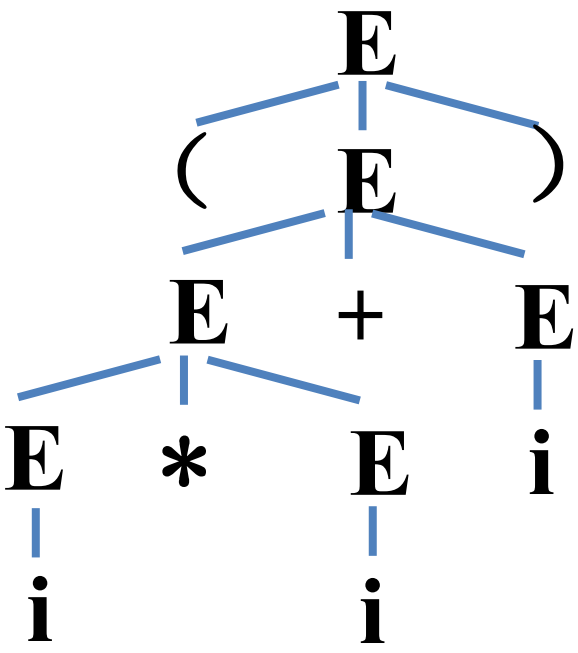
编译原理

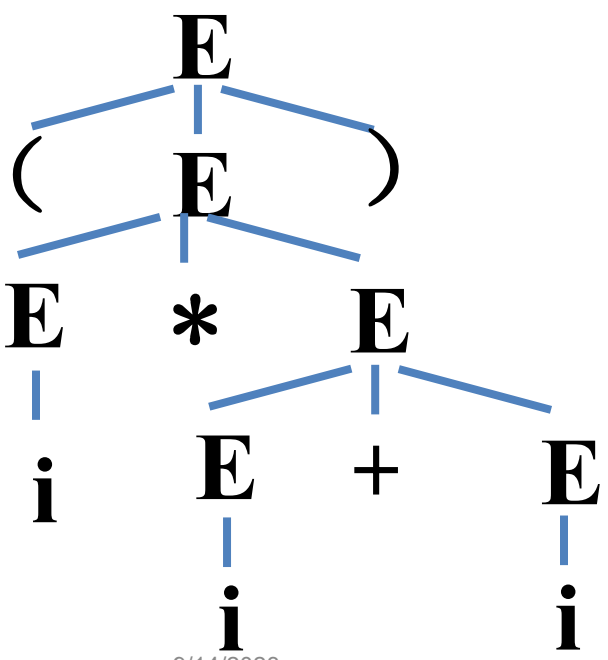
高级程序设计语言的语法描述
——语法树与二义性

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 → i|E+E|E*E|(E)
G(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 → 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

武汉的天气：
冬天能穿多少穿多少，夏天能穿多少穿多少

中国的煤都是（　）

John saw Mary in a boat.

i = i + i * i　　i = （i + i ）* i

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

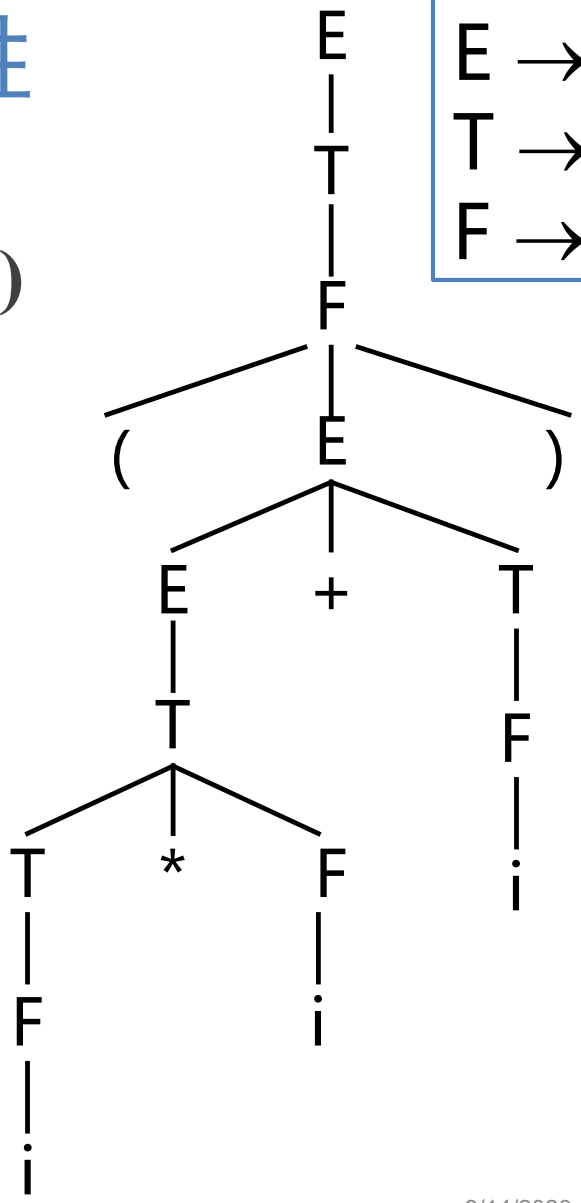
项 → 因子 | 项 * 因子

因子 → (表达式) | i

语言的二义性

- 考虑句子(i*i+i)

$$
\begin{aligned}
E &\Rightarrow T \\
&\Rightarrow F \\
&\Rightarrow (E) \\
&\Rightarrow (E+T) \\
&\Rightarrow (T+T) \\
&\Rightarrow (T*F+T) \\
&\Rightarrow (F*F+T) \\
&\Rightarrow (i*F+T) \\
&\Rightarrow (i*i+T) \\
&\Rightarrow (i*i+F) \\
&\Rightarrow (i*i+i)
\end{aligned}
$$



无二义文法G'(E):

$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

$F \rightarrow (E) \mid i$

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 → 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
- 不可判定问题: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

二义性证明与消除

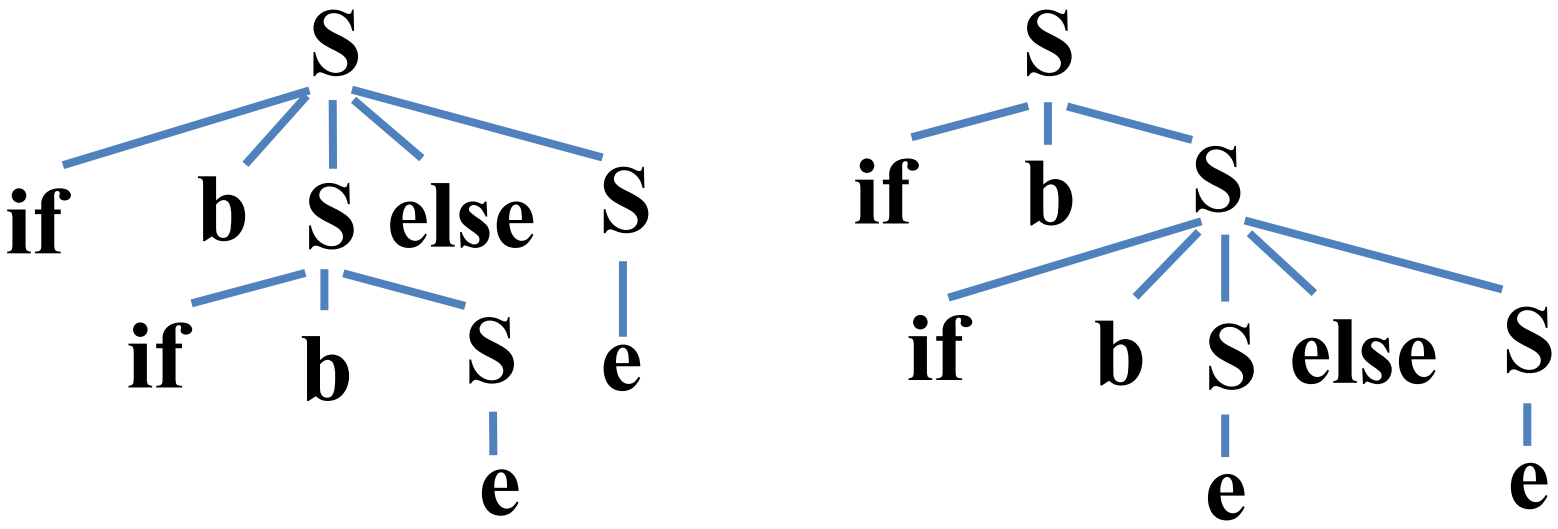
例 定义某程序语言条件语句的文法G为:

$$S \rightarrow \text{if } b \ S$$
$$\mid \text{if } b \ S \ \text{else } S$$
$$\mid e$$

试证明该文法是二义性的

S→if b S | if b S else S | e

存在句子 if b if b e else e 有两个不同的语法树，如下：



所以该文法是二义的。

二义性证明与消除

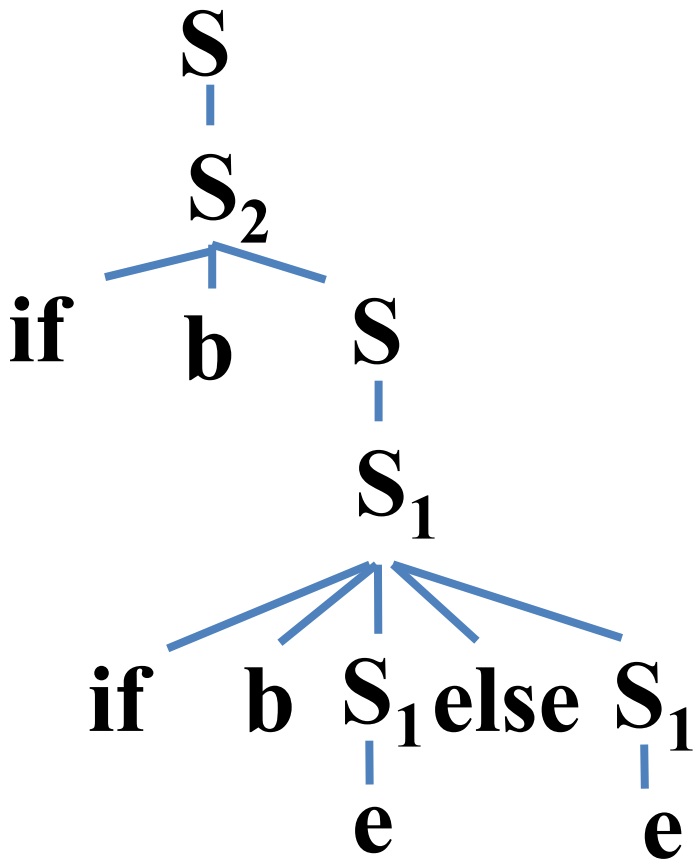
改写文法G为G'

G: S→if b S
| if b S else S
| e

G': S→ S_1 | S_2

S_1→ if b S_1 else S_1 | e

S_2→ if b S | if b S_1 else S_2



L={ a^i b^j c^k | i=j 或 j=k 且 i, j, k≥1}

编译原理

高级程序设计语言的语法描述
——文法和语言的分类

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**

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(Nonterminal)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)

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： $\alpha A \beta \to \alpha \mu \beta$,
  - $A \in V_N$, $\alpha, \beta \in (V_N \cup V_T)^*$ $\mu \in (V_N \cup V_T)^+$
  - 或
  - 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： $A \rightarrow \beta$
  - 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： $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$
  - 其中： $\alpha \in V_T^*$； A， $B \in V_N$
  - 产生式形如： $A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$
  - 其中： $\alpha \in V_T^*$； A， $B \in V_N$

右线性文法

左线性文法

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

$G_6(S)$： $S \rightarrow aSBC|\ aBC$

$CB \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

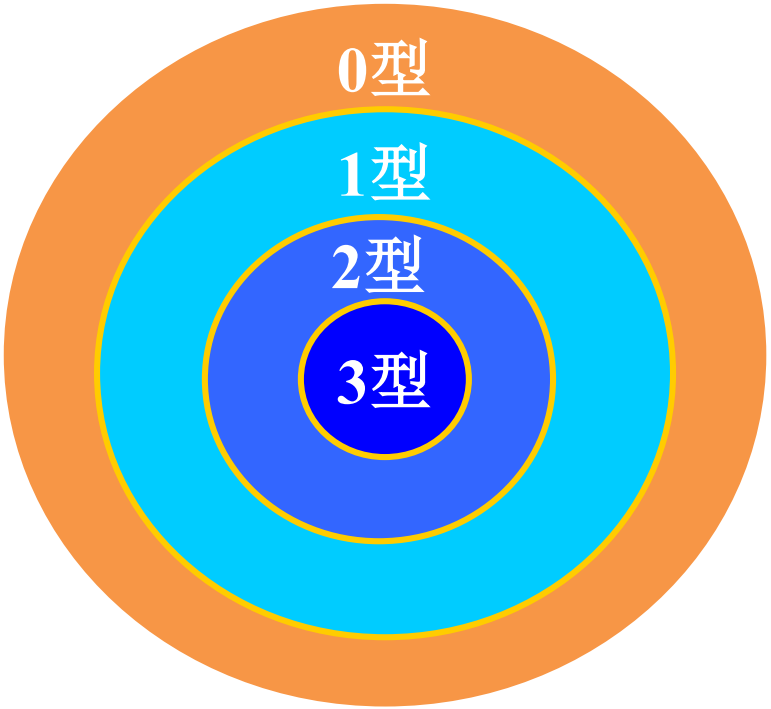
S
⇒aSBC
⇒ aaSBCBC
⇒ aaaBCBCBC
⇒ aaaBBCCBC
⇒ aaaBBCBCC
⇒ aaaBBBCCC
⇒ aaabBBCCC
⇒ aaabbBCCC
⇒ aaabbbCCC
⇒ aaabbbcCC
⇒ aaabbbccC
⇒ aaabbbccc

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**

- 某些语言集合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

四种类型文法描述能力比较



文法的实用限制（**不能含有**）两点：

1. **有害规则：形如A →A 的规则。**

2. **多余规则：**

1）非终结符A无法推出任何终结符号串（不终结）

2）规则A →α 不被使用（不可用）

删除不终结产生式算法

（1）构造能推导出终结符符号串的非终结符集V_{VT}：

①若有A →α 且α ∈V_T*；则令A ∈V_{VT}；

②若有B →β 且β ∈(V_T+V_{VT})*，则令B∈V_{VT}；

③重复① ② ，直到V_{VT}不再扩大为止。

（2）删除不在中的非终结符及其产生式。

删除不终结产生式算法

（1）构造能推导出终结符符号串的非终结符集V_{VT}：

①若有$A \rightarrow \alpha$ 且 $\alpha \in V_T^*$；则令$A \in V_{VT}$；

②若有$B \rightarrow \beta$ 且 $\beta \in (V_T+V_{VT})^*$，则令$B \in V_{VT}$；

③重复① ② ，直到V_{VT}不再扩大为止。

（2）删除不在中的非终结符及其产生式。

删除不可用产生式算法

（1）构造可用的非终结符集V_{US}：

①若S是文法开始符号，则令$S \in V_{US}$；

②若$Z \in V_{US}$，$Z \Rightarrow^+ \cdots A \cdots$，则令$A \in V_{US}$；

③重复② ，直到V_{US}不再扩大为止。

（2）删除不在V_{US}中的非终结符及其产生式。

例如 设有文法G[S]:

P: S→ Bd

A→ Ad | d | A

B→ Cd | Ae

C→ Ce

D→ e

删除多余规则后的文法变换为G’:

P’: S→ Bd

A→ Ad | d

B→ Ae

思考题：类比多余规则，
设计删除 ε 产生式算法；ε $\bar{\epsilon}L(G)$

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