Chapter 2 高级语言及其语法描述

Outlines

- 程序语言的定义
- 高级语言的一般特性
  - 数据类型与操作
  - 语句与控制结构
- 程序语言的语法描述
  - 上下文无关文法
  - 语法分析树与二义性
  - 形式语言

Natural language vs. computer language

- 自然语言(Natural Language)
  - 是人与人的通讯工具
  - 语义(Semantics):环境、背景知识、语气、二义性——<u>难以形式化</u>
- 计算机语言(Computer Language)
  - 计算机系统间、人机间通讯工具
  - 严格的语法(Grammar)、语义(Semantics) ——易于形式化：严格
- 语言是用来交换信息的工具——<u>功能性描述</u>

Programming language

- 程序语言
  - 一套记号系统（符号体系）
  - 有规则、有意义、可以使用
  - <u>**语法、语义**</u>**、语用**
  - 基本功能
    - 描述数据
    - 描述数据运算
- 程序
  - 使用程序语言描述一定数据的处理过程

Programming language

语法	表示构成语言句子的各个记号之间的组合规律(构成规则)。
语义	表示按照各种表示方法所表示的各个记号的特定含义（各个记号和记号所表示的对象之间的关系）。
语用	表示各个记号或语言词句与其使用之间的关系。

Example - C

赋值语句

- **语法**：赋值语句由一个变量，后随一个符号“=”，再在后面跟一个表达式所构成。
- **语义**：先对该语句的右部表达式求值，然后把所得结果与语句左部的变量相结合，并取代该变量原有的值。
- **语用**：赋值语句可用来计算和保存表达式的值。

Grammar

- 字母表
  - 一个有限字符集
    - 包括英文字母(26个)、数字(10个)、其它字符等
- 单词符号
  - 词法规则
  - 具有独立意义的最基本结构
    - 如 0.5，:=
- 语法单位（语法范畴）
  - 语法规则
  - 如表达式、语句、函数、过程、子程序等
    - 如 0.5*X1+C

定义符

标识符

分界符

运算符

常数

Architecture

程序

子程序 或 分程序

语句

表达式

数据引用　　算符　　函数调用

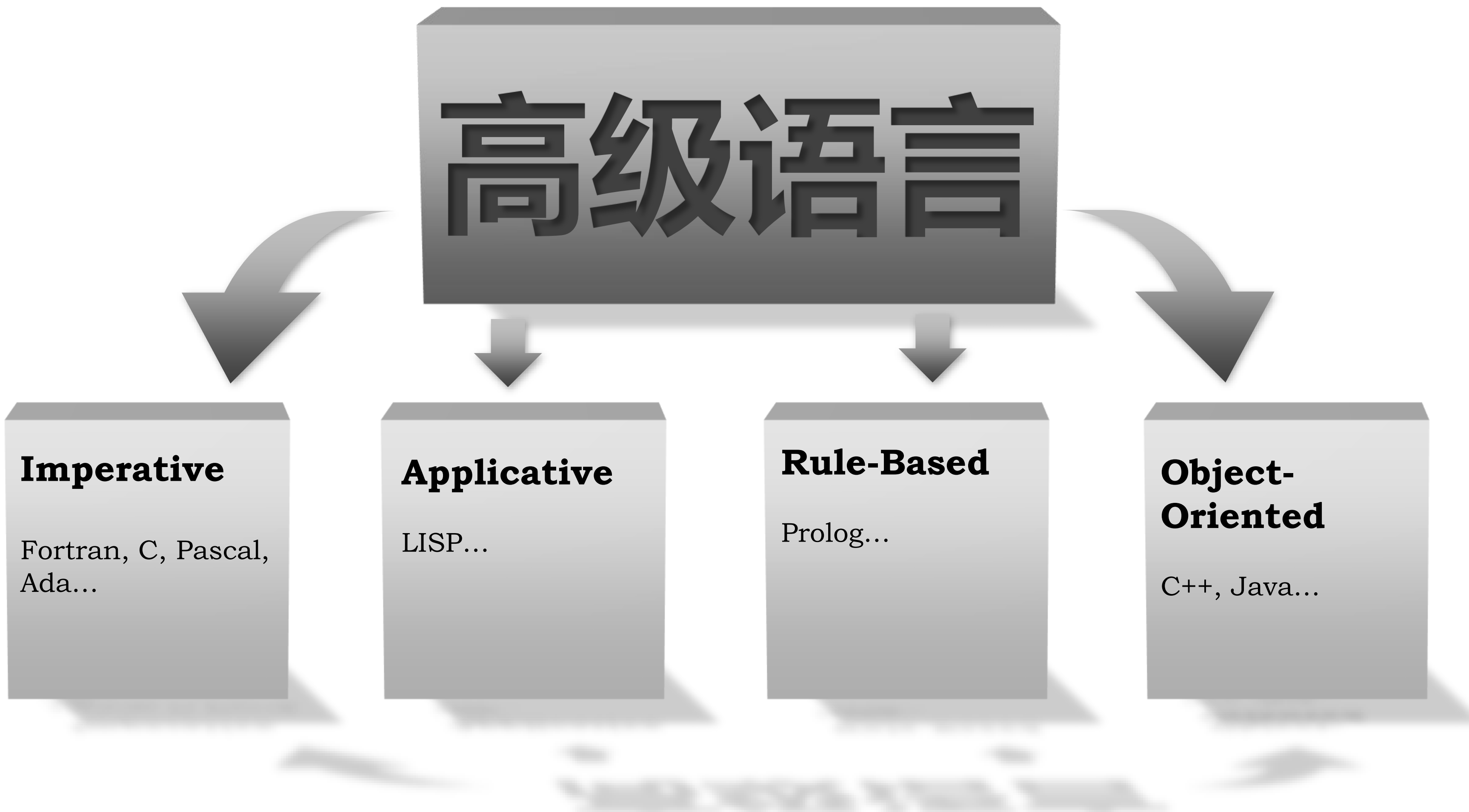
A bit of history

A bit of history

- 1950s
  - 1951 UNIVAC
  - 1954 John Backus FORTRAN
- 1960s
  - 1960 Algol 60
- 1970s
  - 1970 Niklaus Wirth Pascal
  - 1972 Dennis Ritchie C
- 1980s
  - 1980 smalltalk

https://blog.csdn.net/cnliuyong/article/details/82982966

Categories


高级语言
Imperative
Fortran, C, Pascal, Ada…
Applicative
LISP…
Rule-Based
Prolog…
Object-Oriented
C++, Java…

Architecture

- 程序结构 – 单层结构

- 程序结构 – 多层结构

Data Type

- 1）每个被计算对象都带有自己的类型，以类型作为值的属性的概括，因此每个类型都意味着一个值的集合。
- 2）不同类型的值具有不同的操作运算
- 3）类型是一个值的集合和定义在这个值集上的一组操作的总称。
  - 如C语言中的整型变量(int)，其值集为某个区间上的整数，定义在其上的操作为+, -, *, /等

Data type

- 数据类型
  - 包含
    - 数据取值集
    - 数据操作集
  - 分类
    - 基本数据类型
    - 结构数据类型
    - 访问指针
    - 抽象数据类型

Abstract Data Type

- 一个抽象数据类型（ADT）定义为：
  - （1）一个数据对象集，数据对象由一个或多个类型定义；
  - （2）一个作用于这些数据对象的抽象操作集；
  - （3）完全封装，用户除了能使用该类型的操作来处理这类数据对象之外，不能作其他的处理。
- 抽象数据类型有两个重要特征：信息隐蔽和数据封装，使用与实现相分离

Sentence

- 表达式
- 语句
  - 简单语句：不含其它语句成分的基本句
    - 说明语句
    - 赋值语句
    - 控制语句
    - 过程调用语句
  - 复合语句：句中有句的语句

Sentence

- 不同程序语言含有不同形式和功能的各种语句。
- 从功能上说语句大体可分执行性语句和说明性语句两大类：
  - 说明性语句旨在定义不同数据类型的变量或运算。
  - 执行性语句旨在描述程序的动作。
  - 执行性语句又可分为赋值语句、控制语句和输入/输出语句
- 从形式上说，语句可分为简单句、复合句和分程序等。
- 有赋值句、控制语句、说明语句等

Grammar

- 问题：　　如何描述语言
- 定义：　　文法是描述语言的语法结构的形式规则（即语法规则）
- 目的：　　解决语言的有穷说明问题，包含对语法的描述，但却不表达任何语义

Grammar

- 形式上严格、准确；
- 易于理解；
- 具有较强的描述能力；
- 有利于句子的分析和翻译，构造语法分析器

- 文法分类：分为四类（0、1、2、3型文法），对应四类语言；
- 与程序语言语法有关的是上下文无关文法

Grammar

- 字母表 (Alphabet) Σ
  - 符号的有穷集合 (Character)
    - 符号：相互区别的元素
  - 符号串 (String)
    - 符号构成的有穷序列
    - 空字ε：不包含任何符号的序列
    - 全集Σ^*：表示所有符号串的全体
    - 空集Φ：不含任何元素
    - 区分ε, {ε}, Φ

- 例 Σ={a,b}
  - Σ^* ={ε,a,b,aa,ab,aabba,…}

Grammar

- 符号串s的头（前缀 prefix）
  - 如：b是符号串banana的一个前缀
- 符号串s的尾（后缀 suffix）
  - 如：nana是符号串banana的一个后缀
- 符号串s的子串 (substring)
  - 如： ana是符号串banana的一个子串
- 符号串s的真(true)前缀，真后缀，真子串
  - s和ε两者都是符号串s的前缀/后缀/子串
  - 非空符号串x是s的前缀/后缀/子串，则 $s \neq x$

Grammar

- 符号串的长度 (length)
  - 符号的个数
  - 符号串s的长度记为|s|
  - ε的长度为0
- 符号串x、y的连接积 (concatenation product)
  - 符号串xy
    - 如 x=ab,y=cd 则 xy=abcd
  - εa = aε
- 方幂 (power)
  - 符号串自身连接n次得到的符号串
  - a^n定义为n个a的连接
    - $a^0=\varepsilon, a^1=a, a^2=aa$

Grammar

- 符号串集合
  - 若集合A中所有元素都是某字母表Σ上的符号串
  - 则称A为字母表Σ上的符号串集合
- 两个符号串集合A和B的乘积(product)定义为
  - $AB = \{xy | x \in A 且 y \in B\}$
    - 若集合$A = \{ab, cde\}$ $B = \{0, 1\}$
    - 则$AB = \{ab0, ab1, cde0, cde1\}$
- 定义 设Σ_1、Σ_2是两个字母表，Σ_1与Σ_2 的乘积

$$\Sigma_1\Sigma_2=\{ab \mid a\in\Sigma_1,\ b\in\Sigma_2\}$$

Grammar

- 定义 设Σ是一个字母表，Σ的n次幂(Power)递归地定义为：
  - (1) $\Sigma^0=\{\varepsilon\}$
  - (2) $\Sigma^n=\Sigma^{n-1}\Sigma$ $\qquad n\geq 1$

- 例1: 设 $\Sigma_1=\{0,1\}$,

则 $\Sigma_1{}^3 = ?$

Grammar

- 定义　设Σ是一个字母表，Σ的正闭包(Positive Closure):
  - $\Sigma^+=\Sigma\cup\Sigma^2\cup\Sigma^3\cup\Sigma^4\cup\ldots\ldots$
- Σ的克林闭包(Kleene Closure):
  - $\Sigma^*=\Sigma^0\cup\Sigma^+$
  - $\quad=\Sigma^0\cup\Sigma\cup\Sigma^2\cup\Sigma^3\cup\ldots\ldots$

Grammar

- 例2:
- $\{0,1\}^+ = ?$
- $\{a，b，c，d\}^+ = ?$

Grammar

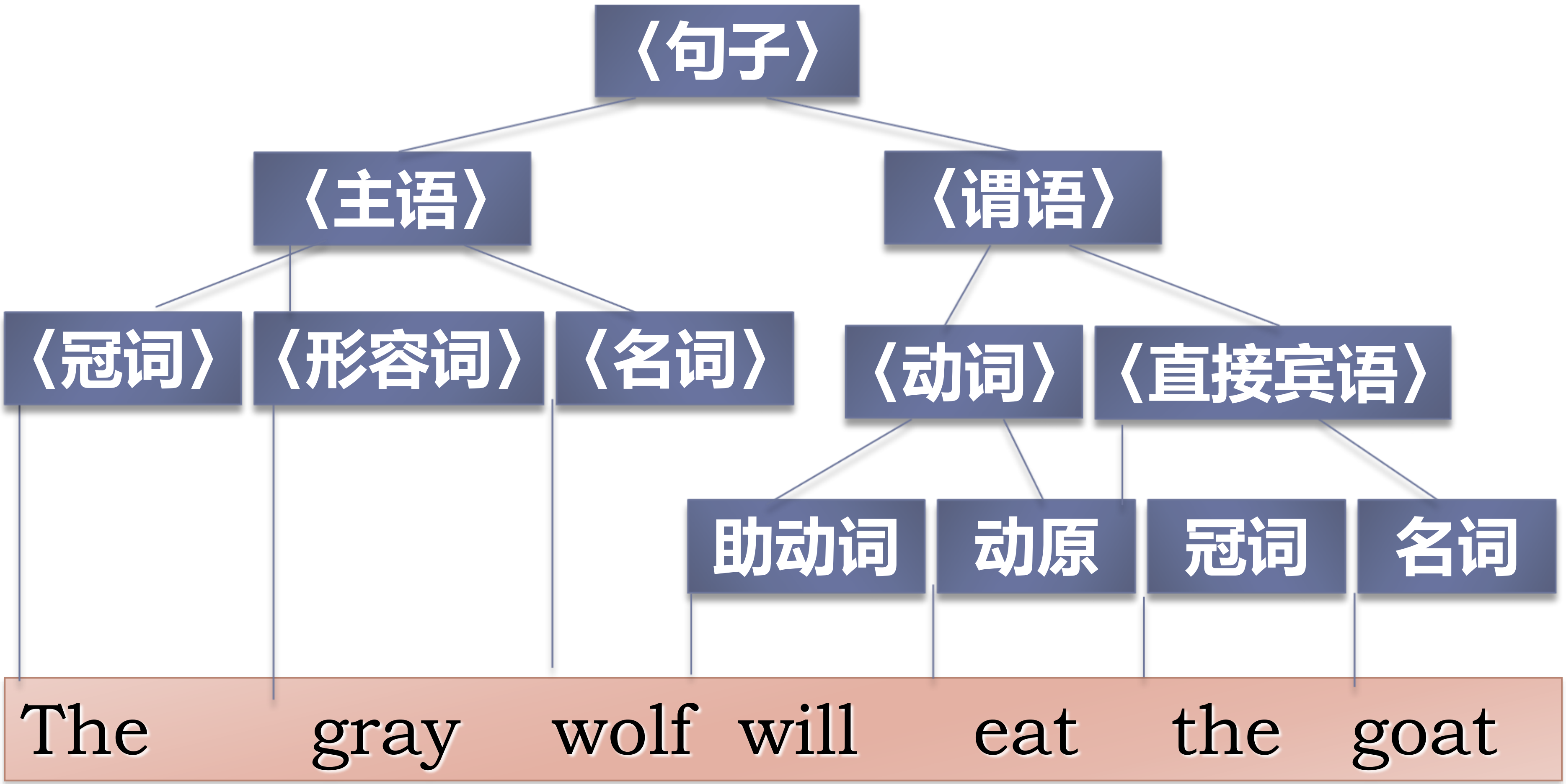
- 例2:
- $\{0,1\}^+ = \{0，1，00，01，10，11，000，001，010，011，100，......\}$
- $\{a，b，c，d\}^+ = \{a，b，c，d，aa，ab，ac，ad，ba，bb，bc，bd，......，aaa，aab，aac，aad，aba，abb，abc......\}$
- What about $\{0,1\}^*$ and $\{a，b，c，d\}^*$?

Context-free grammar

- 上下文无关文法
  - 它所定义的语法范畴（或语法单位）完全独立于这种范畴可能出现的环境之外
  - 不宜描述自然语言
    - 自然语言中，句子和词等往往与上下文紧密相关
  - 四个组成部分
    - 一组**终结符号**
    - 一组**非终结符号**
    - 一个**开始符号**
    - 一组**产生式**

Example

分析：The gray wolf will eat the goat



Example

<句子> → <主语> <谓语>

<主语> → <冠词> <形容词> <名词>

< 谓语> → < 动词> <直接宾语>

< 动词> → <助动词> <动词原形>

<直接宾语> → < 冠词> <名词>

<冠词> → the

<形容词> → gray

<助动词> → will

<动词原形> → eat

<名词> → wolf

<名词> → goat

Context-free grammar

- **A context-free grammar G = (V_T, V_N, S, P)**
  - **终结符 A finite terminal vocabulary V_T**
    - 不可再分
    - 如：基本字、标识符、常数、算符和界符等
    - The token set produced by scanner

Denote symbols in V_T : a,b,c

  - **非终结符 A finite set of nonterminal vocabulary V_N**
    - 代表语法范畴，也称语法变量
    - 一定符号串的集合
    - 如：表达式、赋值句、分程序、过程等
    - Intermediate symbols

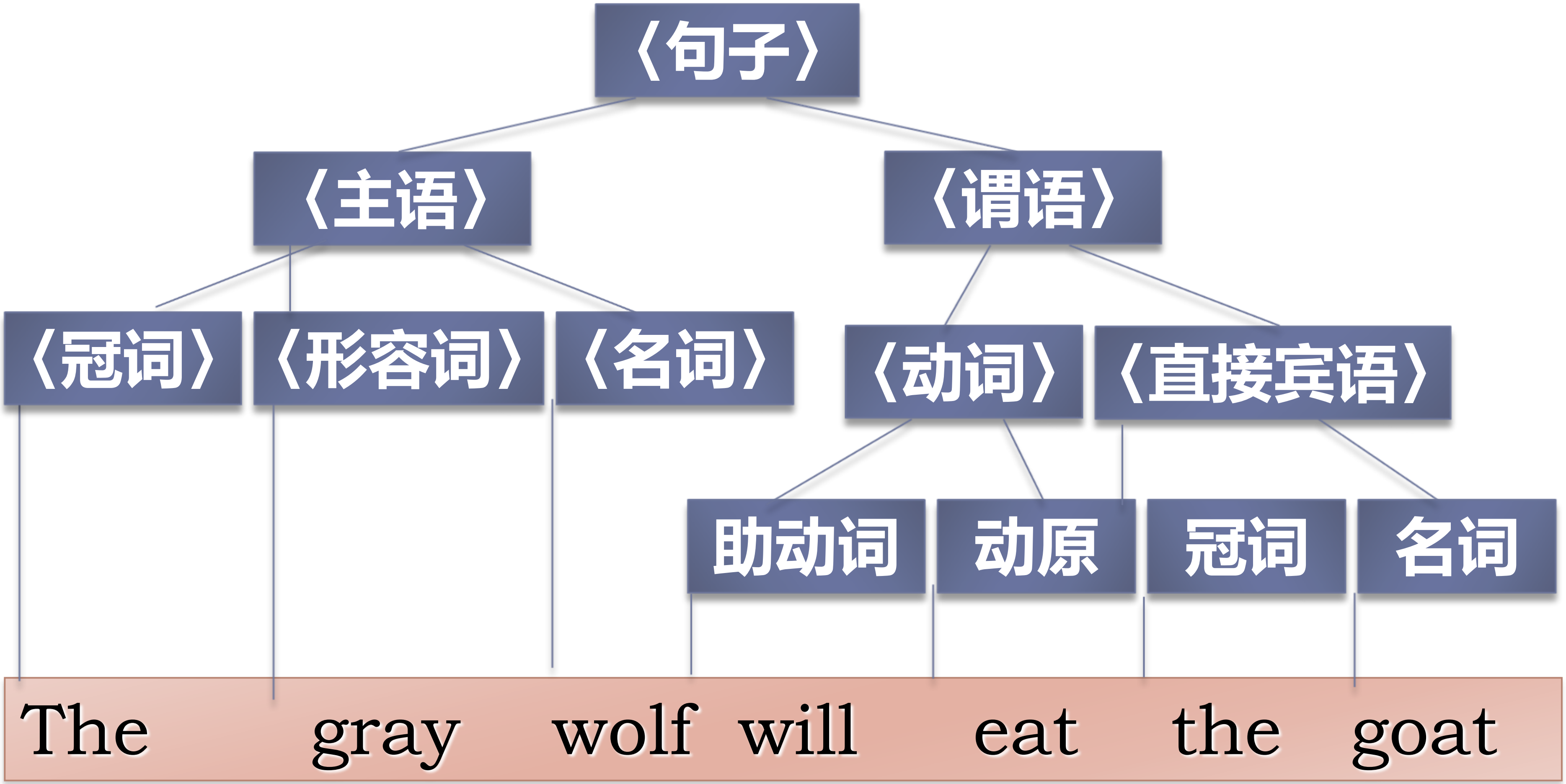
Denote symbols in V_N :A,B,C

Context-free grammar

- **A context-free grammar G = (V_T, V_N, S, P)**
  - **开始符号 A start symbol S ∈ V_N that starts all derivations**
    - 特殊的非终结符
    - Also called **goal** symbol
  - **产生式 P, a finite set of productions (rewriting rules) of the form P →α | β**
  - 左部 P ∈ V_N
  - 右部 α, β ∈Σ^*
  - 元语言符号
    - →读为“定义为”，用“::=”表示，则称为**巴科斯范式(BNF)**
    - | 读为“或”

Example

分析：The gray wolf will eat the goat



Context-free grammar

一个上下文无关文法 **G** 是一个四元式 **(V_T, V_N, S, P)**，其中:

V_T：是非空有限集，它的每个元素是终结符号；

V_N：是非空有限集，它的每个元素是非终结符号；

$V_T \cap V_N = \Phi$; $V_T \cup V_N = \Sigma$;

S：$S \in V_N$，称为开始符号；

P：产生式集合（有限），每个产生式形式是

{ P->α | $P \in V_N$，$\alpha \in (V_T \cup V_N)^*$，S至少一次为P }；

Symbols

- **V_N:** 大写字母A、B、C、S等
- **V_T:** 小写字母，0~9，+、 - 等运算符，
- 标点，分界符，黑体字母串id、if
- **X、Y、Z:** 文法符号，或V_N或V_T一个符号
- **u、v、 w…z:** V_T中串
- **α、β、γ:** 文法符号串$\in(V_T \cup V_N)^*$
- **S:** 开始符号，第一个产生式中出现
- **->:** 定义为（元语言符号）
- **|:** 或（元语言符号）

Example

- **G_1 = < { i, +, *, (,) }, {E}, E, P >**
  - 终结符号集 V_T = { i, +, *, (,) }
  - 非终结符号集 V_N = {E}
  - 开始符号 S = E
  - 产生式集 P: E→ E+E | E*E | (E) | i
    - $E \rightarrow E+E$
    - $E \rightarrow E*E$
    - $E \rightarrow (E)$
    - $E \rightarrow i$

Derivation

- 有穷条产生式，产生无穷集，要求产生式必须递归
- 定义算术表达式，用了一条浓缩的产生式，一般定义一个语言的产生式是很复杂的
- 对递归的算术表达式的产生式，进行反复推导产生表达式语言

- 一个CFG如何定义一个语言呢？
  - 从开始符号出发
  - 反复连续使用产生式
  - 对非终结符施行替换和展开

替换->推导->句型->句子->语言

One step derivation

- 直接推导：是两个字符串之间的一种关系R

如：（αAβ） R （αγβ），它表示：

若 $A\text{->}\gamma \in P$， α、$\beta \in \sum^*$

则 R 就是直接推导，R 记为 $\Rightarrow$

即：$\alpha A \beta \Rightarrow \alpha \gamma \beta$

Derivation

- 如果$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$
  - 则称这个序列是从α_1到α_n的一个推导
- 若存在一个从α_1到α_n的推导
  - 则称α_1可以推导出α_n

- $\alpha_1 \overset{+}{\Rightarrow} \alpha_n$表示：从$\alpha_1$出发，经过一步或若干步，可以推出$\alpha_n$
- $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$表示：从$\alpha_1$出发，经过 0步或若干步，可以推出$\alpha_n$

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

Example

- 例3：
- $\mathbf{G_1} = < \{ \mathbf{i},\ +,\ *,\ (,\ ) \},\ \{\mathbf{E}\},\ \mathbf{E},\ \mathbf{P} >$
  - 产生式集 P: E→ E+E | E*E | （E） | i
    - $E \rightarrow E+E$
    - $E \rightarrow E*E$
    - $E \rightarrow (E)$
    - $E \rightarrow i$
  - $E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (i+E) \Rightarrow (i+i)$

Language derivation

✓只需在推导中加一些限制

即对：$\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ 或 $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$

- 如令α_1为S，即推导要从开始符号开始，那么：
  $S \overset{*}{\Rightarrow} \alpha, \alpha \in V^*$，称α为 G 的**句型**
- 如再要求$\alpha \in V_T^*$，则 α为 G 的**句子**
- 文法G所产生的句子的全体是一个**语言**，记为L(G)
  $L(G) = \{\alpha | S \overset{+}{\Rightarrow} \alpha \ \& \ \alpha \in V_T^*\}$

Example

- 例4:
- **G_{1}: E → E+E | E*E | (E) | i**

- $E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (E*E+E) \Rightarrow (i*E+E) \Rightarrow (i*i+E) \Rightarrow (i*i+i)$
- **句型**：E, (E), (E+E), (E*E+E), (i*E+E), (i*i+E), (i*i+i)
- **句子**：(i*i+i)

Leftmost derivation

- 从一个句型到另一个句型的推导过程并不唯一，但通常只考虑最左推导和最右推导。
  - 最左推导
    - 任何一步推导都是从最左非终结符进行替换
    - $E+E \Rightarrow_{lm} i+E \Rightarrow_{lm} i+i$
    - $E \Rightarrow_{lm} (E) \Rightarrow_{lm} (E+E) \Rightarrow_{lm} (E*E+E) \Rightarrow_{lm} (i*E+E) \Rightarrow_{lm} (i*i+E) \Rightarrow_{lm} (i*i+i)$

Rightmost Derivation

- 最右推导
  - 任何一步推导都是从最右非终结符进行替换
  - 最右推导被称为规范推导(Canonical derivation)
  - 由规范推导所得的句型称为规范句型

  - $E+E \Rightarrow_{rm} E+i \Rightarrow_{rm} i+i$

  - $E \Rightarrow_{rm} (E) \Rightarrow_{rm} (E+E) \Rightarrow_{rm} (E+i) \Rightarrow_{rm} (E*E+i) \Rightarrow_{rm} (E*i+i) \Rightarrow_{rm} (i*i+i)$

Example

- 例5：文法G_2：$S \to bA$，$A \to aA|a$
  - 推导过程
    - $S \Rightarrow bA \Rightarrow ba$
    - $S \Rightarrow bA \Rightarrow baA \Rightarrow baa$
    - …
    - $S \Rightarrow bA \Rightarrow baA \Rightarrow \ldots \Rightarrow ba\ldots a$
  - 归纳得出
    - $L(G_2)=\{ba^n \mid n \geq 1\}$

Example

- 例6：文法**G_3：$S \rightarrow AB$，$A \rightarrow aA|a$，$B \rightarrow bB|b$**
  - 推导过程
    - $S \Rightarrow AB \Rightarrow ab$
    - $S \Rightarrow AB \Rightarrow aAB \Rightarrow aAb \Rightarrow aab \Rightarrow a^2b$
    - $S \Rightarrow AB \Rightarrow abB \Rightarrow abb \Rightarrow ab^2$
    - …
  - 归纳得出
    - $L(G_3)=\{a^m b^n \mid m,n \geq 1\}$

Example

- 例7：构造一个文法G_4使$L(G_4)=\{a^n b^n \mid n\geq 1\}$
  - 观察文法特点
    - 每个句子中，a与b的个数必须相同
  - 根据经验猜测文法
    - **G_4：　$S\rightarrow aSb \mid ab$**

Parse tree

- **语法树**
  - 将一个句型的推导过程表示成一棵倒立的树
    - 根在上，枝叶在下
- 目的：为了理解句子的语法，得到句子如何从开始符号推导得到，因此引入“图”
- 定义：句型推导的图形表示，它与替换顺序的选取无关
- 作用：明显的形成文法所暗含的句子分层语法结构，为语法分析提供一些新的途径

Parse tree

- 设对应G=（V_T,V_N,S,P）的一棵树满足下列4个条件，则此树称作*G的**语法树***
  - 根的标记是S
  - 每个结点都有一个标记，此标记是$V_T \cup V_N$的一个符号
  - 若某结点至少有一个自己除外的子孙，且有标记A
    - 则$A \in V_N$
  - 若某结点有标记A，其直接子孙结点从左到右的次序是$n_1,n_2,\ldots,n_k$，标记分别为$A_1,A_2,\ldots,A_k$
    - 那么$A \to A_1A_2\ldots A_k$一定是P中的一个产生式
- 语法树的结果
  - 由从左到右读出叶子的标记构成

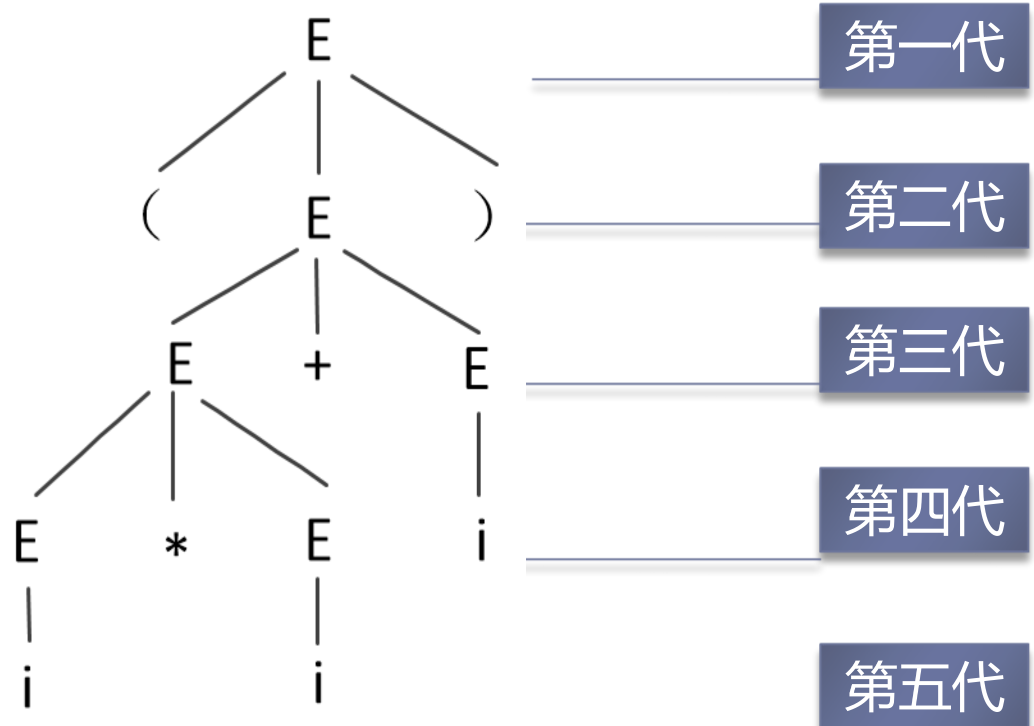
Example

- 语法树的层次（代次）

G_1: $E \rightarrow E+E \mid E*E \mid (E) \mid i$

一棵语法树是不同推导过程的共性抽象

<u>**一棵语法树表示一个句型种种可能的推导过程**</u>



一个句型是否对应唯一的最左/最右推导呢?

否

Example

例：G[S]:

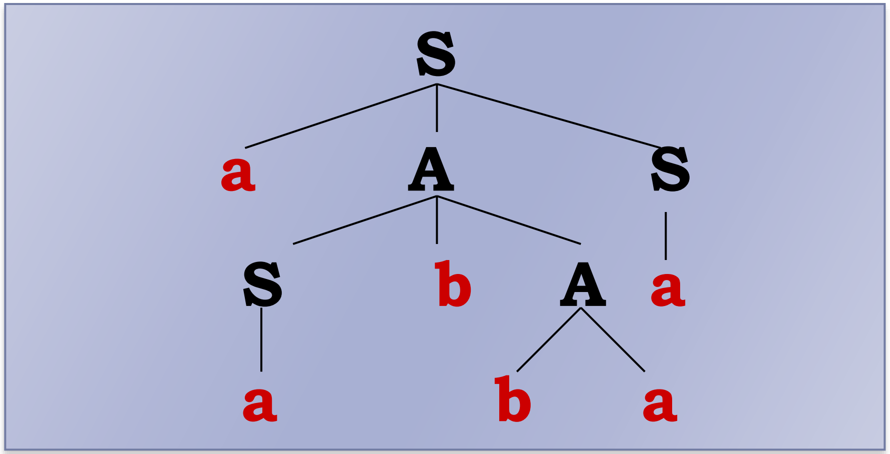
S→aAS

A→SbA

A→SS

S→a

A→ba



S⇒aAS⇒aAa⇒aSbAa⇒aSbbaa⇒aabbaa

S⇒aAS⇒aSbAS⇒aabAS⇒aabbaS⇒aabbaa

S⇒aAS⇒aSbAS⇒aSbAa⇒aabAa⇒aabbaa

Ambiguity

- **G_1: $E \rightarrow E+E \mid E*E \mid (E) \mid i$**
- (i*i+i)

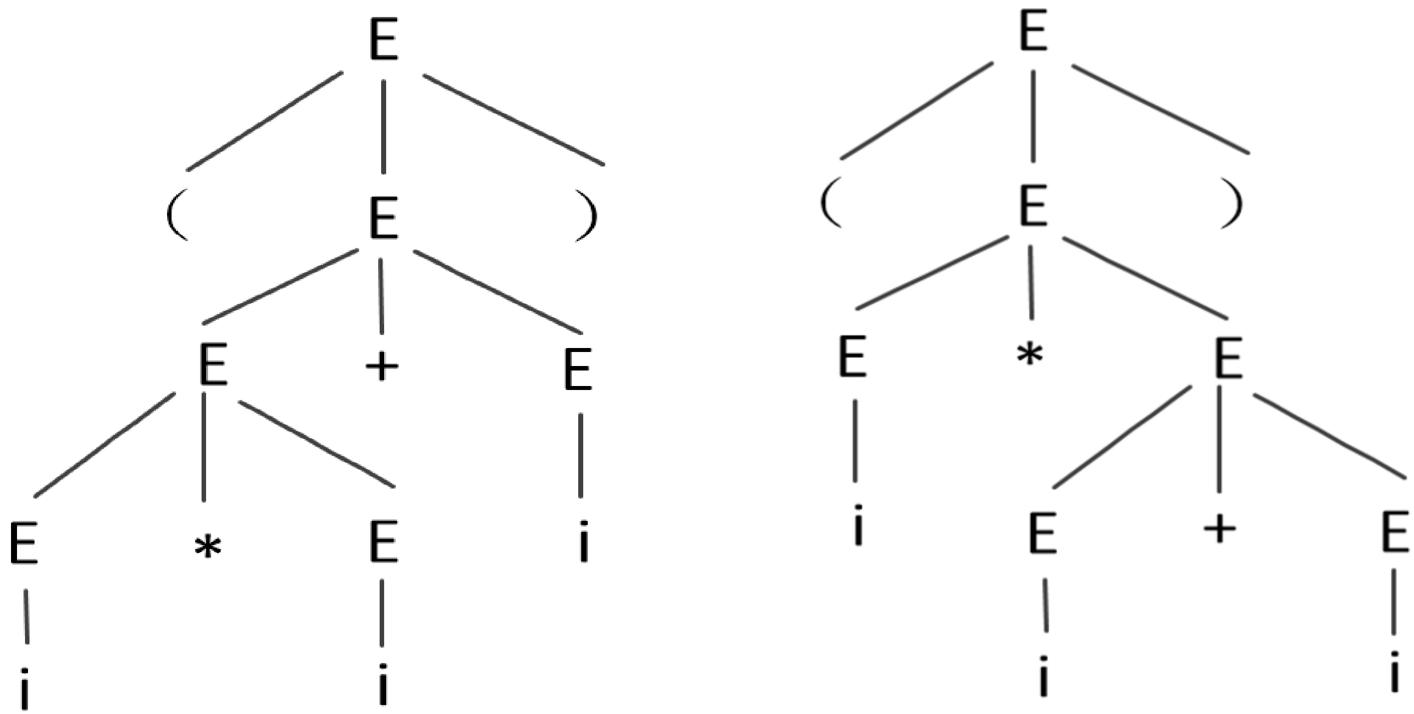
- $E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (E*E+E) \Rightarrow (i*E+E) \Rightarrow (i*i+E) \Rightarrow (i*i+i)$

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

Ambiguity

$G_{1:}$ E → E+E | E*E | (E) | i

(i*i+i)



Ambiguity

- 二义性问题
  - 定义 文法G的某一句子有两棵不同的树，则G为二义的
  - 或者，若一个文法存在某个句子有两个不同的最左（右）推导，则称这个文法是二义的
- 处理二义性对语法分析不便，因此希望：
  - 判定二义否
  - 控制充分条件，消除二义性
- 消除二义性
  - 优先性，结合性

Disambiguation

- 例8：二义文法改造为无二义文法

$$G[E]: E \rightarrow E+E \qquad E \rightarrow i$$

$$G[E]: E \rightarrow i+E \qquad E \rightarrow i$$

$$G[E]: E \rightarrow E+E \qquad E \rightarrow E*E \qquad E \rightarrow i$$

$$G[E]: E \rightarrow T+E \mid T \qquad T \rightarrow i*T \mid i$$

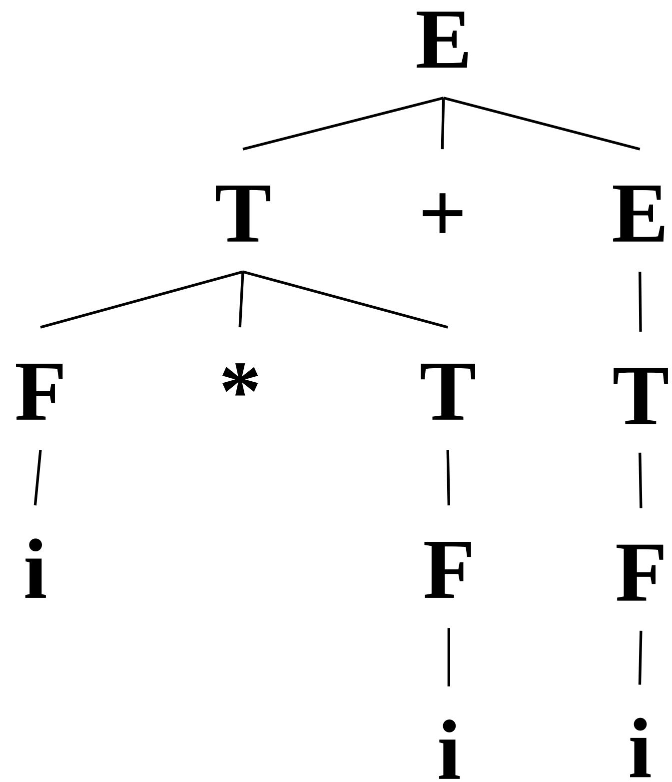
$$G[E]: E \rightarrow E+E \qquad E \rightarrow E*E \qquad E \rightarrow (E) \qquad E \rightarrow i$$

$$G[E]: E \rightarrow T+E \mid T \qquad T \rightarrow F*T \mid F \qquad F \rightarrow (E) \mid i$$

Disambiguation

$$\begin{aligned} G[E]:\ E &\rightarrow T+E \mid T \\ T &\rightarrow F*T \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

i*i+i



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

Disambiguation

- 文法的二义性和语言的二义性是两个不同的概念。因为可能有两个不同的文法G和G′，其中G是二义的，但是却有L(G)=L(G′)，也就是说，这两个文法所产生的语言是**相同**的。

- 如果产生上下文无关语言的每一个文法都是二义的，则说此语言是先天二义的。对于一个程序设计语言来说，常常希望它的文法是**无二义**的，因为希望对它的每个语句的分析是唯一的。

Disambiguation

- 文法的二义性是<u>不可判定</u>的。
  - 不存在一种算法，**有限步**内确切判定一个文法是否为二义的
  - 上下文无关方法的二义性也是不可计算的

- 描述程序设计语言时，对于上下文无关文法的限制：
  - 不含P→P形式的产生式
  - 每个非终结符P必须有用处

Disambiguation

$E \rightarrow T \mid E + T \mid H$

$T \rightarrow T \mid (E) \mid i$

$M \rightarrow T$

开始符号E

单一产生式(T)

派生不出终结符号(H)

从开始符号无法派生出来(M)

可计算性理论

- 数学体系
  - 完备性：一切命题能被证明或者证伪
  - 一致性：命题之间无矛盾
  - 可计算：能找到一种方法，在有限步内判定一个命题的正确性
- 哥德尔
  - 哥德尔不完备性定理：初等代数不能同时完备和一致
- 丘奇-图灵
  - 停机问题

可计算性理论

- 无穷集之间比多少
  - f(A) → B，f是单射，则认为|A|≤|B|
  - |A|≤|B| & |B|≤|A|，则认为|A|=|B|
  - 可数集：|A|=|N|，N是自然数集合

	1	2	3	……
1				
2				
3				
……				

可计算性理论

- 对角线法

	1	2	3	…
1	0	1	1	
2	1	0	1	
3	0	1	1	
…				

.110……

参考资料：
罗杰·彭罗斯，《皇帝新脑》

Formal language

- 形式语言理论
  - 创始人
    - 乔姆斯基（Chomsky），1956年
  - 研究内容
    - 符号串集合的**表示法**、**结构**及其**特性**
  - 对程序设计语言进行语法分析研究的基础
    - 程序语言的设计
    - 编译方法
    - 计算复杂性等

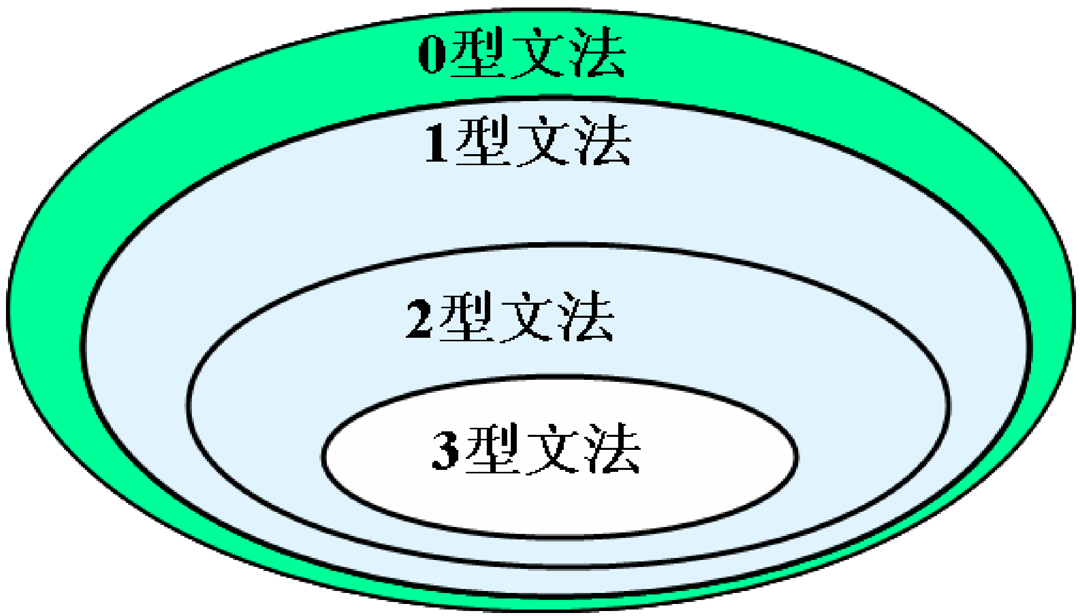
Formal language

- 如何来描述一种语言?
  - 文法G所产生的句子的全体是一个语言L(G)
    - 若语言只含有穷多个句子，则可以将句子逐一列出来表示
    - 若语言包含无穷多个句子，则应该采取语言的有穷表示方法
- 语言的有穷表示有两种途径
  - 生成方式 （文法）
    - 语言中的每个句子可以用严格定义的规则来构造
    - **如：上下文无关文法能够构造程序设计语言**
  - 识别方式（自动机）
    - 描述字符串的识别过程
      - 当输入的一任意串属于语言时，该过程经有限次计算后就会停止并回答“是”
      - 若不属于，要么能停止并回答“不是”，要么永远继续下去
    - **如：有限自动机能够识别单词符号**

Formal language

- Chomsky将文法分成四种类型
  - 0型文法：**短语文法 (Phrase grammar)**
    - 能力相当于图灵机
    - 任何0型语言都是递归可枚举的
    - 递归可枚举集必定是一个0型语言
  - 1型文法：**上下文有关文法 (Context-sensitive grammar)**
    - 替换非终结符时必须考虑上下文
    - 产生式的形式为$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$
    - 只有A出现在α_1和α_2的上下文中时，才允许β取代A
    - 识别系统是线性界限自动机
  - 2型文法：**上下文无关文法**
    - 产生式的形式为$A \rightarrow \beta$
    - β取代A时与A的上下文无关
    - 识别系统是非确定的下推自动机
    - 足以描述现今多数程序设计语言的语法结构
  - 3型文法：**正规文法 (Regular grammar)**
    - 产生的语言是有穷自动机所接受的集合

Formal language



逐级“包含”关系

Formal language

- 0型文法
  - 对任一产生式$\alpha \to \beta$,都有
    - $\alpha \in (V_N \cup V_T)^*$且至少含有一个非终结符
    - $\beta \in (V_N \cup V_T)^*$
- 1型文法
  - 除$S \to \varepsilon$外,对任一产生式$\alpha \to \beta$都有$|\alpha| \le |\beta|$
  - S不得出现在任何产生式的右部
- 2型文法
  - 对任一产生式$\alpha \to \beta$,都有
    - $\alpha \in V_N$
    - $\beta \in (V_N \cup V_T)^*$
- 3型文法
  - 任一产生式$\alpha \to \beta$的形式都为$A \to aB$或$A \to a$
    - $A \in V_N$
    - $B \in V_N$
    - $a \in V_T^*$

随着对产生式的约束条件逐渐增强，文法描述语言的能力逐渐减弱

Final note

- 若$L(G_1)=L(G_2)$,则称文法G_1和G_2是等价的
  - 例如
    - $G_1[A]$：$A \rightarrow 0R$, $R \rightarrow A1$, $A \rightarrow 01$
    - $G_2[S]$：$S \rightarrow 0S1$, $S \rightarrow 01$

Example

例9： **G_1： $S \rightarrow 0S1$，**

$S \rightarrow 01$

求： $G_1(S)$的语言?

$L(G)=\{0^n1^n | n \geqslant 1\}$

Example

例10：若已知文法**G_2(A)：**　　**A → c | Ab**

请给出G_2(A)的语言?

- L(G_2)={c， cb， cbb， …}

  以c开头，后继若干个b

Example

例11：给出产生语言为$\{a^n b^m \mid 1 \le n \le m \le 2n\}$的文法

- G(S):

  $S \to aSb \mid aSbb$

  $S \to ab$

Example

例12：若已知文法**$G_4(S)$:**

$S \rightarrow aSBE$　　**$S \rightarrow aBE$**

$EB \rightarrow BE$　　**$aB \rightarrow ab$**

$bB \rightarrow bb$　　**$bE \rightarrow be$**

$eE \rightarrow ee$

请给出$G_4(S)$的语言?

L（G4(S)）$=\{ a^n b^n e^n \mid n \geqslant 1 \}$

Example

- 例13： **文法G[S]：S→AB　　A→A0 | 1B　B→0 | S1，请给出句子101001的最左和最右推导。**

最左推导：$S \Rightarrow AB \Rightarrow 1BB \Rightarrow 10B \Rightarrow 10S1 \Rightarrow 10AB1$

$\Rightarrow 101BB1 \Rightarrow 1010B1 \Rightarrow 101001$

最右推导：$S \Rightarrow AB \Rightarrow AS1 \Rightarrow AAB1 \Rightarrow AA01 \Rightarrow A1B01$

$\Rightarrow A1001 \Rightarrow 1B1001 \Rightarrow 101001$

Example

例14：是非题

（1）因名字都是用标识符表示的，故名字与标识符没有区别

×

（2）正规文法产生的语言都可以用上下文无关文法来描述

√

（3）上下文无关文法比正规文法具有更强的描述能力

√

Example

例15：指出下述文法属性，并给出其描述的语言：

$G_1(S)$： $S \rightarrow Be$, $B \rightarrow eC \mid Af$, $A \rightarrow Ae \mid e$

$C \rightarrow Cf$, $D \rightarrow fDA$

2型文法

$G_2(S)$: $A \rightarrow aB$, $B \rightarrow Ab \mid a$

2型文法

$G_3(S)$: $S \rightarrow abcA$, $S \rightarrow Aabc$, $A \rightarrow \varepsilon$,

$Aa \rightarrow Sa$, $cA \rightarrow cS$

0型文法

Example

例16：给出文法G：

G：　S→aSb | P

P→bPc | bQc

Q→Qa | a

(1)它是乔姆斯基哪一型文法

(2)它生成的语言是什么?

2型文法

$$L(G)=\{a^i b^j a^k c^j b^i\}$$

Example

例17：给出如下文法：

G： $P \to aPQR \mid abR$

$RQ \to QR$

$bQ \to bb$

$bR \to bc$

$cR \to cc$

(1)它是乔姆斯基哪一型文法

(2)证明aaabbbccc是G的一个句子

1型文法

$P \Rightarrow aPQR$

$\Rightarrow aaPQRQR$

$\Rightarrow aaabRQRQR$

$\Rightarrow aaabQRQRR$

$\Rightarrow aaabbRQRR$

$\Rightarrow aaabbQRRR$

$\Rightarrow aaabbbRRR$

$\Rightarrow aaabbbcRR$

$\Rightarrow aaabbbccR$

$\Rightarrow aaabbbccc$

Example

例18：给出产生语言为$\{a^n b^m c^m d^n \mid 0 \le n，1 \le m\}$的文法

G (S):

$S \to aSd \mid A$

$A \to bAc \mid bc$

Example

例19：文法G的产生式集为

{**S → S+S | S*S | i | (S)** }

对于输入串i+i*i:

(1)给出一个推导

(2)画出一棵语法树

(3)文法G是否是二义性的，请证明你的结论

Example

例20：文法G：

$$S \rightarrow SaS \mid SbS \mid cSd \mid eS \mid f$$

文法G是否是二义性的，请证明你的结论