



上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

# 第五章 上下文无关语言

姚 刚

中国科学院信息工程研究所



# 目录

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## ① 上下文无关文法

## ② 分析与二义性

## ③ 上下文无关语言和程序设计语言



# 扩大语言族

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

因为正则语言可以有效描述某种简单模式，所以我们可以比较简单地找到非正则语言的例子。在程序设计语言中，有一种简单的嵌套结构，如 $(( ))$ 和 $(( ( )) )$ 的形式，而 $(( ))$ 则不会出现。这意味着程序设计语言的某些属性需要某些正则语言不具有的性质。

为此，我们需要扩大语言族，着导致我们考虑上下文无关(context-free)语言及其文法。



# 正则文法的限制

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

正则文法的产生式有两方面的限制：

- 左侧必须是一个简单变量；
- 右侧有固定的形式。

为了创造出更强大的文法，我们放宽这些限制：保留左侧的限制，允许右侧是任何形式，我们就获得了上下文无关文法。



# 上下文无关文法

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

## 定义 (上下文无关文法)

文法  $G = (V, T, S, P)$  称为是上下文无关的 (*context-free*), 如果  $P$  中的产生式具有形式  $A \rightarrow x$ , 其中  $A \in V$ ,  $x \in (V \cup T)^*$ 。语言  $L$  是上下文无关的, 当且仅当存在一个上下文无关文法  $G$ , 使得  $L = L(G)$ 。



# 上下文无关文法

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

每个正则文法都是上下文无关文法，因此正则语言也是上下文无关语言。

$\{a^n b^n\}$  是上下文无关语言，但不是正则语言。因此，正则语言族是上下文无关语言族的真子集。

上下文无关文法得名于句型中出现的产生式的左部可以在任何时候被替换，这种替换不依赖于句型中的其他符号。



# 例子

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

文法  $G = (\{S\}, \{a, b\}, S, P)$ , 产生式

$$S \rightarrow aSa, \quad S \rightarrow bSb, \quad S \rightarrow \varepsilon,$$

这个文法是上下文无关的。

文法  $G = (\{S, A, B\}, \{a, b\}, S, P)$ , 产生式

$$S \rightarrow abB, \quad A \rightarrow aaBb,$$

$$A \rightarrow \varepsilon, \quad B \rightarrow bbAa,$$

这个文法是上下文无关的。



# 例子

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

语言  $L = \{a^n b^m : n \neq m\}$  是上下文无关的。

注

线性文法和正则文法都是上下文无关的，但是上下文无关文法不一定是线性的。这个例子给出的文法是上下文无关的，不是线性的，但是不能说这个语言就不是线性的。

考虑文法  $S \rightarrow aSb | SS | \varepsilon$ ，这个文法是上下文无关的，给出其生成的语言。





# 推导过程

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

在上下文无关文法中，推导可能包括由多个变量构成的句型。这样，在替换变量的时候，我们可以进行选择。

为了表示使用的是哪个产生式，我们将产生式进行编号，并在 $\Rightarrow$ 上标出相应的编号。



# 例子

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

文法  $G = (\{S, A, B\}, \{a, b\}, S, P)$ , 产生式

①  $S \rightarrow AB,$

②  $A \rightarrow aaA,$

③  $A \rightarrow \varepsilon,$

④  $B \rightarrow Bb,$

⑤  $B \rightarrow \varepsilon,$

考察文法生成的语言, 并给出  $aab$  的推导过程。



# 推导顺序

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

在上下文无关语法中，不同的推导可能得到相同的句子，而且使用了相同的产生式。它们的不同仅仅是使用产生式的顺序不同。

为了去掉这些不相关的因素，我们要求变量按照固定的顺序被替换。



# 最左推导与最右推导

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## 定义 (最左推导与最右推导)

如果每次都替换句型中最左端的变量, 那么这种推导称为最左的 (*leftmost*)。

如果每次都替换句型中最右端的变量, 那么这种推导称为最右的 (*rightmost*)。

文法  $G = (\{S, A, B\}, \{a, b\}, S, P)$ , 产生式  $S \rightarrow aAB$ ,  $A \rightarrow bBb$ ,  $B \rightarrow A|\varepsilon$ , 给出  $abbbb$  的最左推导与最右推导。



# 另一种推导方式

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

我们给出另一种表示推导过程的方式，叫做推导树(derivation tree)，这种方式与产生式 $P$ 的使用顺序无关。

推导树是一个有序树，其中父结点标记的是产生式左部的符号，子结点标记的是该产生式右部的符号。



# 推导树

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## 定义 (推导树)

$G = (V, T, S, P)$  是一个上下文无关文法。当且仅当下列性质成立时，一个有序树才能称为是  $G$  的推导树：

- 根结点标记为  $S$ 。
- 每个叶结点有一个属于  $T \cup \{\epsilon\}$  的标记。
- 每个中间结点(不是叶结点的结点)都有一个属于  $V$  的标记。



# 推导树

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## 定义 (推导树(续))

- 如果一个结点有标记  $A \in V$ , 它的子结点标记(从左到右)为  $a_1, a_2, \dots, a_n$ , 那么,  $P$  中一定存在包含形式为  $A \rightarrow a_1 a_2 \dots a_n$  的产生式。
- 标记为  $\varepsilon$  的叶结点没有兄弟, 即: 一个结点标记为  $\varepsilon$ , 则这个结点是叶结点, 并且是它的父结点的唯一子结点。



# 部分推导树

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

## 定义

如果一棵树拥有部分推导树的后三条性质，但是第一条性质不成立，并且把第二条性质替换为：

- 每个叶结点都有一个属于  $V \cup T \cup \{\varepsilon\}$  的标记。

那么，这样的一棵树就称为部分推导树 (*partial derivation tree*)。





# 树的果(yield)

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

通过从左向右地读出树上的叶结点获得的符号串，忽略遇到的任何 $\epsilon$ ，称为树的果。

“从左到右”的术语描述可以给出一个准确的意义。果是终结符构成的符号串，是按照深度优先遍历树时遇到的字母顺序构成的，深度优先遍历总是先遍历最左部没有被遍历过的分支。



# 例子

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

文法  $G = (\{S, A, B\}, \{a, b\}, S, P)$ , 产生式

$$S \rightarrow aAB, \quad A \rightarrow bBb, \quad B \rightarrow A|\varepsilon,$$

给出  $abbbb$  的推导树。



# 定理

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

## 定理

设 $G = (V, T, S, P)$ 是一个上下文无关文法，那么对于每个 $w \in L(G)$ ， $G$ 中总存在一个推导树，它的果是 $w$ 。反过来，任何推导树的果都属于语言 $L(G)$ 。

同样，如果 $t_G$ 是 $G$ 的任何一个部分推导树，且根结点标记为 $S$ ，那么 $t_G$ 的果就是 $G$ 的的一个句型。



# 推导树与产生式的顺序

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

推导树指出了获取句子的过程中用到了哪些产生式，但是没有给出这些产生式应用的顺序。推导树能够表示任何推导，反映了顺序是无关的。

对于  $w \in L(G)$  都有一个推导，但是不能说一定有最左推导和最右推导。有了推导树，通过把树看成是如下方法构造的，即总是由树的最左边变量开始扩展，来得到最左推导。任何  $w \in L(G)$ ，既有最左推导，也有最右推导。



# 分析

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

人们在学习自然语言的时候都熟悉一种方式，就是通过文法推导解释一条语句，我们称之为分析(parsing)。

分析是描述语句结构的一种形式。每当我们需要理解语义的时候，例如要把一种语言翻译成另一种语言，分析是非常重要的。



# 分析

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

给定文法 $G$ ，我们学习用 $G$ 可以推导出符号串的集合。

在实际中，我们也关注文法的分析性质：给定终结符构成的符号串 $w$ ，我们想要知道 $w$ 是否属于 $L(G)$ 。

成员资格判定算法可以告诉我们 $w$ 是否属于 $L(G)$ 。术语分析描述的是寻找 $w \in L(G)$ 的推导过程中使用的一系列产生式。



# 分析过程

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

给定 $L(G)$ 中的符号串 $w$ ，我们系统地构造所有可能的(比如：最左)推导，看看推导出的结果是否能够匹配 $w$ 。

首先看所有具有形式 $S \rightarrow x$ 的产生式，从而发现可以由 $S$ 一步推导出的 $x$ 。

如果结果都不匹配 $w$ ，那么将所有可以应用的产生式都用在每个 $x$ 的最左边的变量上。这样得到一个句型的集合，或许其中有的句型就能推导出 $w$ 。



# 分析过程(续)

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

在后续的过程中，考察所有最左边的变量，应用所有可以应用的产生式。如果句型不能推出 $w$ ，就抛弃它。因而，每个过程都会获得一些可能的句型。

这样，首先获得使用一个简单产生式而生成的句型，然后获得使用两步推导可以得到的句型……如果 $w \in L(G)$ ，那么，它一定有有限长度的最左推导。因此，这种方法最后会给出 $w$ 的最左推导。





# 穷举搜索分析

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

上面所述的这个分析方法，我们称之为穷举搜索方法(exhaustive search parsing)。它是自顶向下分析(top-down parsing)的一种形式，我们可以把它看成是自根部向下构造推导树。

文法  $G = (\{S\}, \{a, b\}, S, P)$ ，产生式

$$S \rightarrow SS|aSb|bSa|\varepsilon,$$

分析符号串  $w = aabb$ 。



# 穷举搜索的缺陷

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

- ① 最明显的缺陷是过程冗长，不适用于需要使用高效率分析的地方。
- ② 用这种方法分析 $w \in L(G)$ 时，如果得不到属于 $L(G)$ 的符号串，就可能永远都不会停止。除非使用某种方式使它停下来。



# 限制文法

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

如果限制文法的形式，穷举搜索分析的不终止问题就相对容易解决。

我们看到使用  $S \rightarrow \varepsilon$  的产生式会减少推导出的后继句型的长度，因此不容易判断出什么时候停止推导。如果没有这样的产生式，困难就少些。

事实上，我们要消除两种形式的产生式： $A \rightarrow \varepsilon$  和  $A \rightarrow B$ 。后面我们会看到，这种限制不影响最终的文法表达能力。



# 例子

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

文法  $G = (\{S\}, \{a, b\}, S, P)$ , 产生式

$$S \rightarrow SS|aSb|bSa|ab|ba,$$

能够生成前面例子中的语言, 除了空串。



# 定理

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言和程序设计语言

## 定理

假设文法  $G = (V, T, S, P)$  是上下文无关文法，并且不具备下面形式的产生式：

$$A \rightarrow \varepsilon \text{ 或者 } A \rightarrow B,$$

其中， $A, B \in V$ 。那么由穷举搜索分析法可以生成一个算法，它能完成：对于任何  $w \in \Sigma^*$ ，要么生成  $w$  的一种分析，要么告诉我们不可能存在分析。



# 产生句型的数量的上限

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

穷举搜索分析法得到的句型数量，没有准确的结果，但是可以给出一个严格的上限。

如果我们只限制最左推导，一步推导后，得到的句型不超过 $|P|$ 个，两步推导后，得到的句型不超过 $|P|^2$ 个...

由上面定理，推导过程不超过 $2|w|$ 步，因此，得到的句型不超过

$$M = |P| + |P|^2 + \cdots + |P|^{2|w|}。$$



# 定理

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## 定理

每个上下文无关文法都存在一个算法，它对于任何  $w \in L(G)$ ，都可以和  $|w|^3$  成正比的步数进行分析。

上面的算法，工作量与符号串长度的三次幂成比例增加，虽然比指数算法好些，但是也不是太有效。我们想要的是和符号串的长度成正比的时间复杂度的分析法，我们把这种方法称为线性时间(linear time)分析算法。



# 简单文法

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## 定义 (简单文法)

上下文无关文法  $G = (V, T, S, P)$  是简单文法 (*simple grammar* 或 *s-grammar*), 如果它的产生式形式都是

$$A \rightarrow ax,$$

其中,  $A \in V$ ,  $a \in T$ ,  $x \in V^*$ , 并且, 任何对  $(A, a)$  在  $P$  中至多出现一次。





# 例子

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

文法 $G$

$$S \rightarrow aS|bSS|c$$

是简单文法；而文法

$$S \rightarrow aS|bSS|aSS|c$$

不是简单文法。

如果 $G$ 是简单文法，那么 $L(G)$ 中的任何符号串 $w$ ，都可以在与 $|w|$ 成正比的时间复杂度范围内分析完。



# 二义性

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

对任何给定的 $w \in L(G)$ ，穷举搜索分析一定会产生一棵 $w$ 的推导树。我们说“一棵”，而不是特指某棵的原因在于可能存在大量不同的推导树。

## 定义 (二义性)

如果对某个 $w \in L(G)$ ，至少存在两棵不同的推导树，那么，上下文无关文法 $G$ 称作是二义性的(*ambiguous*)。也就是说，二义性意味着存在两个或者两个以上的最左或最右推导。



# 例子

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

已知文法  $G = (V, T, S, P)$ , 其中  $V = \{E, I\}$ ,  $T = \{a, b, c, +, *, (, )\}$ , 产生式为

$$E \rightarrow I, \quad E \rightarrow E + E,$$

$$E \rightarrow E * E, \quad E \rightarrow (E),$$

$$I \rightarrow a|b|c,$$

看  $a + b * c$  的两棵推导树。



# 去除二义性

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

注

在程序设计语言中，每条语句只应该有一种解释，要去除二义性，通常是通过重写一个等价的、无二义性的文法来达到这个目的。

重写上面例子中的文法。



# 去除二义性

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

在一般情况下，一个给定的上下文无关文法是否是二义性的，或者两个上下文无关文法是否等价，这两个问题都是非常困难的。

事实上，不存在通用的算法来解决这些问题。



# 无二义性与固有二义性

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

## 定义

如果一个上下文无关语言 $L$ 存在一个无二义性文法，那么 $L$ 就是无二义性的。如果每个生成 $L$ 的文法都是二义性的，那么这个语言是固有二义性的(*inherently ambiguous*)。

语言 $L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$ 是一个具有固有二义性的上下文无关语言，其中 $n$ 和 $m$ 是非负整数。



# 形式语言的应用

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

形式语言理论的一个最重要应用是程序设计语言的定义和它们的解释器、编译器的构造。

这里的基本问题是准确地定义程序设计语言，并使用这个定义作为书写有效、可靠的翻译程序的起点。



# 文法定义程序设计语言

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

通常，我们习惯使用一种特殊的文法来定义语言，这种文法称为巴克斯-诺尔形式(Backus-Naur Form, BNF)，这种形式在本质上和我们使用的表示方式相同，但是表现形式不同。

在BNF中，变量放在尖括号中表示，终结符不用特殊标记。BNF也使用辅助符号，例如|等。





# 文法定义程序设计语言

上下文无关语言

姚刚

目录

上下文无关文法

分析与二义性

上下文无关语言  
和程序设计语言

一些程序设计语言的很多部分使用了上下文无关文法的严格方式表示，容易受到定义的影响。

然而，并不是一个典型程序设计语言的所有特性都可以被简单文法表示。

此外，程序设计语言的规约必须是无二义性的。



# 研究的问题

上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

程序设计语言的语义是一个复杂的问题。程序设计语义的规约不像上下文无关文法这样优雅而简洁。

无论从程序设计语言方面，还是从形式语言理论方面，定义程序设计语言语义还是一个正在研究中的问题。



上下文无关语  
言

姚刚

目录

上下文无关文  
法

分析与二义性

上下文无关语  
言和程序设计  
语言

# 谢谢!

主 讲 人： 姚 刚

电子邮箱： [yaogang@iie.ac.cn](mailto:yaogang@iie.ac.cn)