程序设计语言的语法通常是由上下文无关的文法规则给出的。

按照构造分析树或语法树的方式，算法大致可分为两种：自顶向下分析和由底向上分析

语法分析程序

1、任务

根据扫描程序产生的token确定程序的语法结构并显式或隐式地构造出表示该结构的分析树或语法树，因此称为语法分析。

输入

扫描程序生成的token序列

语法分析程序的输出

语法树

2、一些同义名词

这里说的"同义"和"="有可能并不严谨，（甚至说文章其他的一些部分也并不严谨，懂的人会看出来点东西的）但针对目前需求，严谨程度并不重要。

产生式=文法规则

名字=结构=产生式头=非终结符

终结符=token

2、上下文无关文法

定义

上下文无关文法由以下各项组成：

令G为一个上下文无关文法，即$G = (T, N, P, S)$。

- 终结符集合T
- 非终结符集合N

  非终结符集合和非终结符集合不相交
- 产生式集合P

  产生式$A \to \alpha$，其中A是一个非终结符，α是终结符或非终结符的克林闭包，即$\alpha \in (T \cup N)^*$。
- 开始符号S

  开始符号是一个非终结符

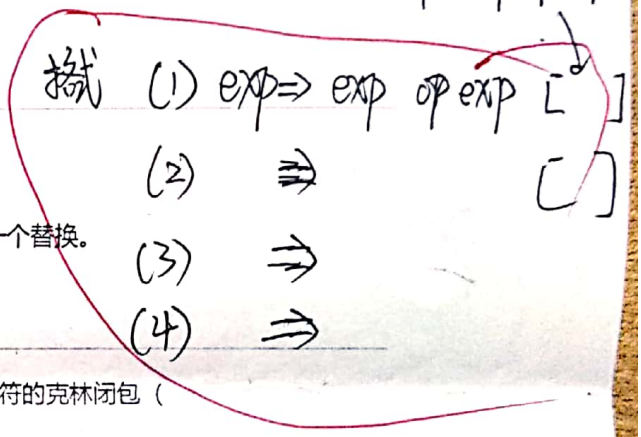
BNF文法

BNF文法，又称Backus-Naur范式、巴科斯范式。

- 文法规则定义了在箭头左边名字的结构
- 默认第一个列出的是开始符号
- 名字用斜体表示（同时字体不同，因此可与正则表达式中的斜体区分开）
  这个斜体也可用尖括号代替，同时将用大写替代之前正则中的斜体
- |作为选择的元符号，并置是标准运算，但没有重复的元符号（比如正则表达式中的*）
- 用箭头→代替了等号来表示名字的定义
  用来代替箭头元符号的通常有等号、冒号、双冒号等号
- 将正则表达式作为部件
- 当将括号作为一个元符号时，通过将括号放在双引号中以区分括号token和括号元符号，这和正则表达式中一样
- 通常把文法规则写成每个名字的所有选择都可在一个规则中列出来，而且每个名字在箭头左边只出现一次

exp→exp op exp

推导



- 在文法规则的右边选择一个序列替换名字。
- 推导以一个结构名字开始并以token序列结束。
- 在每一个推导步骤中，使用来自文法规则的选择每一次生成一个替换。

G上的推导步骤

G上的推导步骤格式为$\alpha A\gamma \Rightarrow \alpha\beta\gamma$，其中$\alpha$和$\gamma$是终结符或非终结符的克林闭包（$\alpha \in (T \cup N)^*, \gamma \in (T \cup N)^*$），且$A \to \beta \in P$。

G的符号集

终结符集合和非终结符集合的并被称作G的符号集。

句型

$(T \cup N)^*$中的串α被称作句型。

推导步骤关系的传递闭包

将关系$\alpha \Rightarrow^* \beta$定义为推导步骤关系$\Rightarrow$的传递闭包，即$\alpha$经过0个或多个推导步骤得到$\beta$。

句子

在文法G上的推导$S \Rightarrow^* \omega$，其中ω是终结符的克林闭包（$\omega \in T^*$）。

由G生成的语言

$L(G)$是由开始符号推导出的句子的集合。

$L(G) = \{\omega \in T^* | 存在\ S \Rightarrow^* \omega\}$。

最左推导

有推导$S \Rightarrow^* \omega$，若其中的每一个推导步骤$\alpha A \gamma \Rightarrow \alpha\beta\gamma$都有$\alpha \in T^*$，则其为最左推导，即先推导左边再推导右边。

最右推导

有推导$S \Rightarrow^* \omega$，若其中的每一个推导步骤$\alpha A \gamma \Rightarrow \alpha\beta\gamma$都有$\gamma \in T^*$，则其为最右推导，即先推导右边再推导左边。

文法G上的分析树

文法G上的分析树是一个带有以下属性的做了标记的树：

- 每个节点都用终结符、非终结符或ϵ标出
- 根节点都用开始符号标出
- 每个叶子节点都用终结符或ϵ标出
- 每个非叶子节点（内部节点）都用非终结符标出

  每个内部节点到其所有子节点表示一个推导步骤中的相关非终结符的替换
- 如带有标记$A \in N$的非叶子节点有n个带有标记$X_1, X_2, \ldots, X_n$的孩子节点，就有

  $A \to X_1 X_2 \ldots X_n \in P$

每一个推导都引出一个分析树；许多推导可引出相同的分析树，但每个分析树只有唯一的一个最左推导和唯一的一个最右推导（最左推导与分析树的前序编号相对应，最右推导与分析树的后序编号相对应）

上下文无关语言

若存在上下文无关文法G有$L = L(G)$，则L就成为上下文无关语言。

不同的文法可以生成相同的语言，但根据文法的不同，语言中的串会有不同的分析树。

二义性

若存在串$\omega \in L(G)$，其中ω有两个不同的分析树，那么文法G就有二义性。

几个符号的区别

- 推导步骤中用⇒。
- 文法规则中用→。
- 正则表达式中用=。

基础情况

递归地定义一个结构的文法规则必须总是有至少一个非递归情况（称之为基础情况）

ϵ产生式

$A \to \epsilon$

上下文无关文法与正则表达式

两者的主要区别

上下文无关文法利用了与正则表达式中极为类似的命名惯例和运算，两者的主要区别是上下文无关文法的规则是递归的。

两者主要区别造成的影响

1. 上下文无关文法识别的结构类比正则表达式识别的要大得多。
2. 识别上下文无关文法的算法必须使用递归调用或显示管理的分析栈。
3. 用作表示语言语义结构的数据结构也必须是递归的（分析树和语法树），而不是词法分析中的线性结构。

乔姆斯基层次

- 0型文法：非限制文法，与图灵机对等
- 1型文法：上下文有关文法
- 2型文法：上下文无关文法，与下推自动机对等
- 3型文法：正则文法，和有限自动机对等

分析树

对于同一个串可有多个推导，一个分析树可对应多个推导。

分析树的内部节点用非终结符标记，叶子节点用终结符标记，每个内部节点到其所有子节点表示一个推导步骤中的相关非终结符的替换。

语法树

又叫抽象语法树。

存在同属连接

EBNF表示法

重复和可选的结构在程序设计语言中极为普通，BNF文法包含了这两个情况后就形成了扩展的BNF或EBNF表示法。

重复

EBNF使用花括号{}来表示重复（正则表达式中的克林闭包）。

可选

EBNF使用方括号[]表示可选结构。

语法图

可视表示EBNF规则的图形表示法称作语法图。

- 矩形/正方形

  表示非终结符

- 圆形/椭圆形

  表示终结符
- 带箭头的线

  表示序列和选择
- 非终结符标记

  表示该语法图对应的文法规则

重复

相应语法图

可选

相应语法图