# 人工智能-第三次课程作业报告

授课教师：张宇 作者：<张博彦>-<58121124>

## 1 问题描述

### 1.1题目介绍

给定语言的词典和语言的语法，现输入字符串语句S = “the wumpus is dead”。 请采用短语结构句法解析语句S，并输出句法解析树的线性表示 “[*S*:[*NP*:[*Article*: **the**] [*Noun*: **wumpus**]] [*VP*:[*Verb*: **is**][*Adjective*: **dead**]]]”。

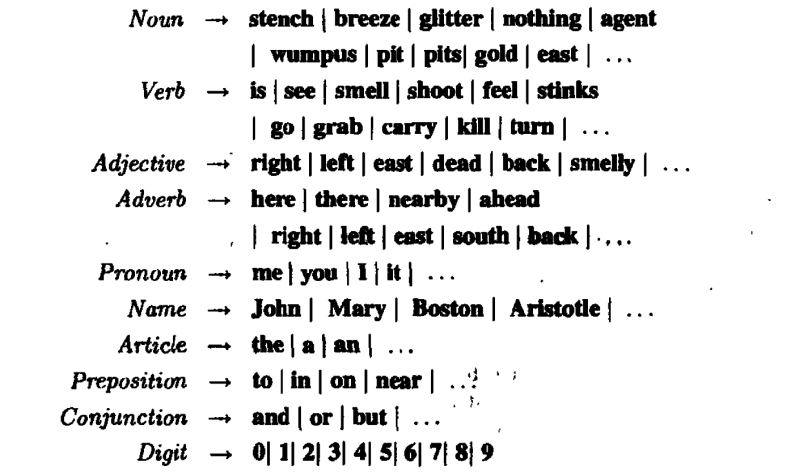


图1

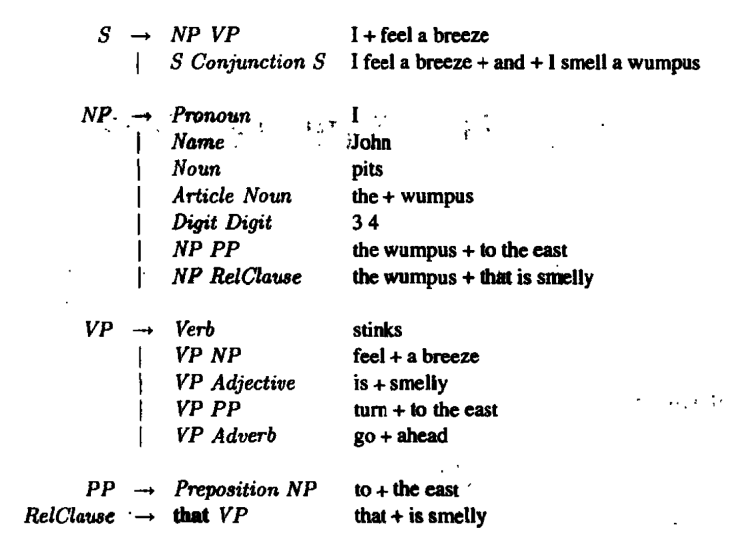


图2

### 1.2任务说明

使用打开Assignment4.sln项目，实现Nlp.cpp中的三个函数：1）get\_possible\_chunkings；

2）parse；3）parse\_rec。完成函数后运行项目针对“The Wumpus is dead”应得到图3中的结果。

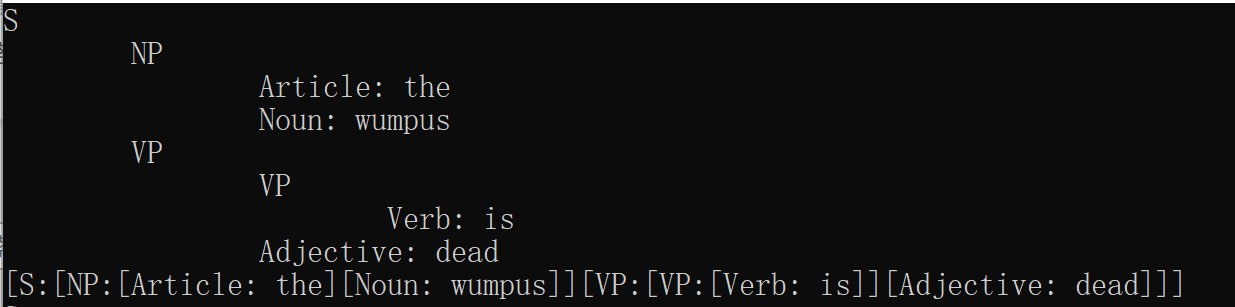


图3 句法解析树的线表示

Nlp类介绍：

Nlp类中保存两个重要的成员变量，分别为字典和语法。

字典的数据结构为：map< string, vector<LexRule> >，字典的键为词性，值为vector<LexRule>。

形式如下：

{

*"Noun"*: [

{*"pos"*: "Noun", *"lexicon"*: "stench", *"prob"*: 0.05},

{*"pos"*: "Noun", *"lexicon"*: "breeze", *"prob"*: 0.1}

]

}

最外层的“Noun”代表该键储存所有名词的词汇规则，LexRule有三个成员变量，分别为词性（pos），词汇（lexicon），此种规则出现的概率（prob）。

语法的数据结构为：map< string, vector<CFGRule> >，字典的键为语法规则的左边，值为vector<CFGRule>。

形式如下：

{

*"S"*: [

{*"lhs"*: "S", *"rhs"*: ["NP", "VP"], *"prob"*: 0.9},

{*"lhs"*: "S", *"rhs"*: ["S", "Conj", "S"], *"prob"*: 0.1}

]

}

最外层的“S”代表该键储存所有句子语法规则，CFGRule有三个成员变量，分别为规则左边（lhs），规则右边（rhs），此种规则出现的概率（prob）。上面的例子表示，一个句子有90%的概率会被拆解为名词性词组+动词性词组（NP+VP）的形式，有10%的概率会被拆解为句子+连接词+句子（S+Conj+S）的形式。

### 1.3实验环境

Microsoft Visual Studio Community 2022

版本 17.1.2

VisualStudio.17.Release/17.1.2+32319.34

Microsoft .NET Framework

版本 4.8.04084

已安装的版本: Community

Visual C++ 2022 00482-90000-00000-AA689

Microsoft Visual C++ 2022

ASP.NET and Web Tools 2019 17.1.358.51495

ASP.NET and Web Tools 2019

Azure 应用服务工具 3.0.0 版 17.1.358.51495

Azure 应用服务工具 3.0.0 版

C# 工具 4.1.0-5.22165.10+e555772db77ca828b02b4bd547c318387f11d01f

IDE 中使用的 C# 组件。可能使用其他版本的编译器，具体取决于你的项目类型和设置。

Microsoft JVM Debugger 1.0

Provides support for connecting the Visual Studio debugger to JDWP compatible Java Virtual Machines

Microsoft MI-Based Debugger 1.0

Provides support for connecting Visual Studio to MI compatible debuggers

Microsoft Visual C++ 向导 1.0

Microsoft Visual C++ 向导

Microsoft Visual Studio VC 软件包 1.0

Microsoft Visual Studio VC 软件包

NuGet 包管理器 6.1.0

Visual Studio 中的 NuGet 包管理器。有关 NuGet 的详细信息，请访问 https://docs.nuget.org/

TypeScript Tools 17.0.1229.2001

TypeScript Tools for Microsoft Visual Studio

Visual Basic 工具 4.1.0-5.22165.10+e555772db77ca828b02b4bd547c318387f11d01f

IDE 中使用的 Visual Basic 组件。可能使用其他版本的编译器，具体取决于你的项目类型和设置。

Visual Studio Code 调试适配器主机包 1.0

用于在 Visual Studio 中托管 Visual Studio Code 调试适配器的互操作层

Visual Studio IntelliCode 2.2

Visual Studio 的 AI 协助开发。

Visual Studio Tools for CMake 1.0

Visual Studio Tools for CMake

适用于 Google Test 的测试适配器 1.0

启用带有针对 Google Test 编写的单元测试的 Visual Studio 测试工具。扩展安装目录中提供了使用条款和第三方通知。

用于 Boost.Test 的测试适配器 1.0

通过针对 Boost.Test 编写的单元测试启用 Visual Studio 测试工具。扩展安装目录中提供用户条款和第三方通知。

### 1.4评价标准

**3.1文档部分（2分）**

* 针对本题目撰写文档，保证文档结构完整性，文档可包括 题目、问题描述、实验方案、实验结果、分析、结论，也可参考其他标准文档或者学术论文的格式。
* 文档内容及结构均纳入评分范围。
* 按照A4纸排版（可以一页或多页），签上姓名、学号，打印或手写后交给助教，具体时间等待群通知。
* **文档：1分。源代码风格：1分**

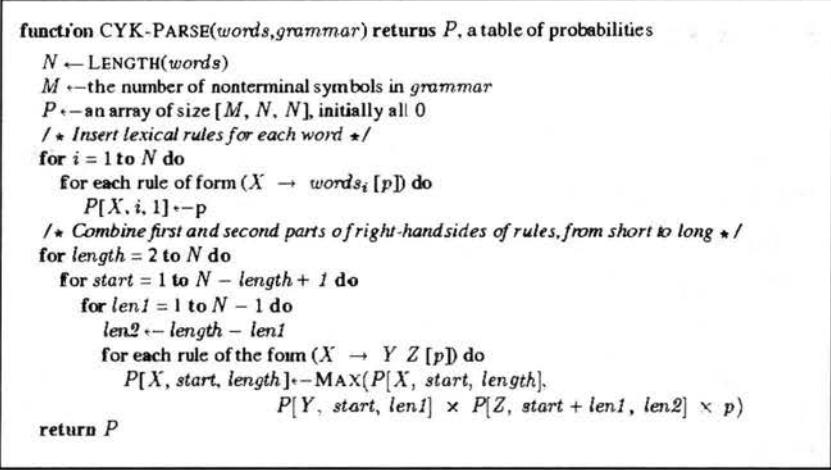
**3.2源代码部分（8分）**

1个函数，8分，解析树的正确率。由于该句子在给定句法下没有歧义，所以只有正确和错误两种结果。要比较标准答案的解析树和同学们提交的解析树是否完全相同。

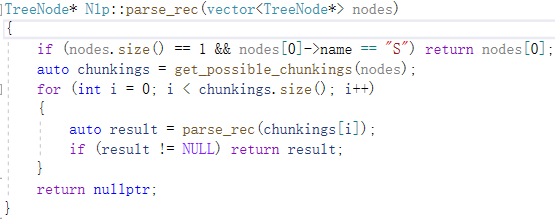
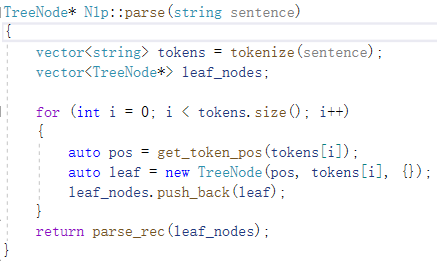
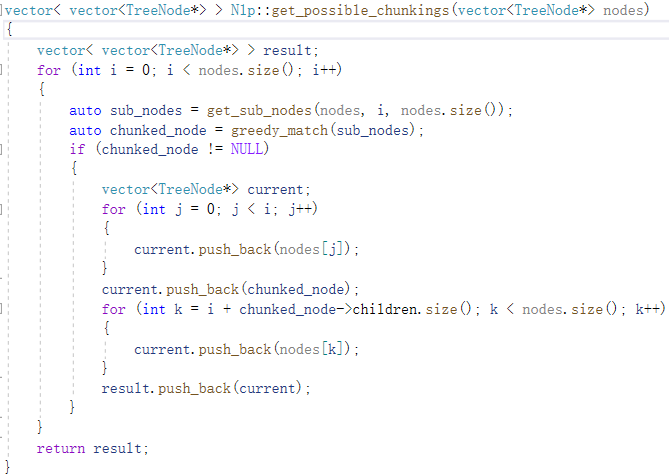
## 2 实验方案

**2.1理论基础**

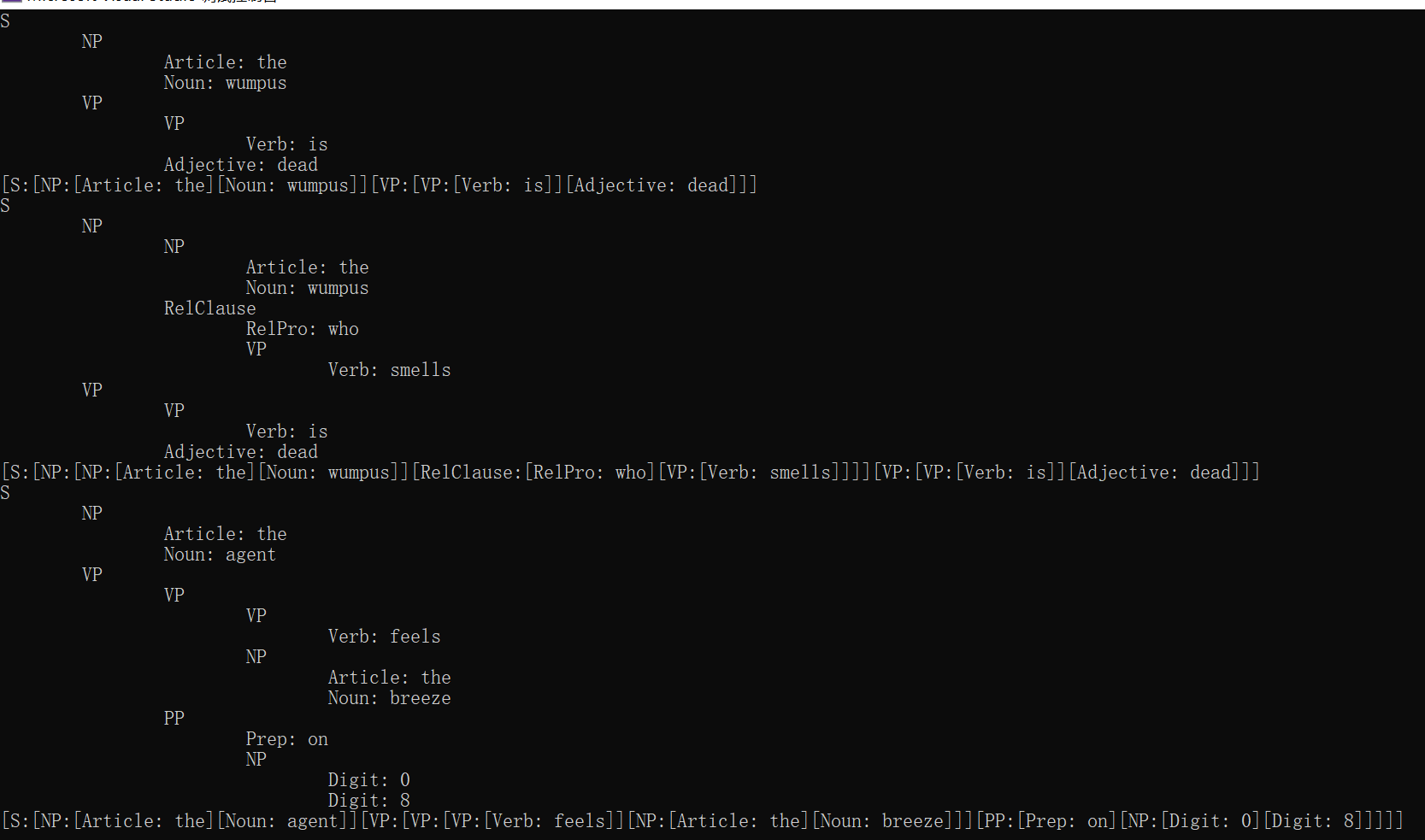
使用动态规划方法：每次分析子串时，就将结果存储起来，将来再碰到就无需重新分析了。我们可以将结果记录在一种被称为图（chart) 的数据结构中。做这种工作的算法被称为图分析器（chart parser)。因为我们使用的是上下文无关文法，因此在搜索空间中，我们在某一分支的上下文中对任何短语的分析，也可以在其他分支中适用。图分析器有很多种类，我们介绍的是一种自底而上的方法，被称为 CYK 算法（CYK algorithm)。



**2.2具体代码**



## 3实验结果



## 4实验分析

如果一个子句包含有 c 个双歧义词，我们就会得出 2^c种不同的途径去挑选这个句子的 解释。那么 CYK 算法是如何在 O(c^3)的时间内处理 2^c的分析树的？答案是该算法并没有 检査所有的分析树；它所做的只是计算可能性最大的树的概率。这些子树都表示在表 P中，并且通过少量的工作我们就可以将他们全部枚举出来（在指数时间内），但 CYK 算法的美 妙之处就在于我们不必枚举出他们，除非我们需要这么做。

在实践中，我们通常只对最好的分析结果感兴趣，而不是所有的分析结果。可以将 CYK 算法看成是定义了按照“运用文法规则”这一操作而得到的完全状态空间，因而可以利用A\*算法实现只搜索空间的某一部分。空间中的每一个状态都是一个项目列表（单词或句法范畴)，如自底向上分析表（图 23.4) 中所示。起始状态是一个单词序列，而目标状态是一个项目&每个状态的代价为按照当前已使用规则的概率而计算得到的概率的倒数，也有许多启发式方法可用来估计到目标的剩余距离。最佳的启发式是通过机器学习从句子语料库上获取。利用算法我们没必要搜索整个状态空间，而且我们也能确保我们首先找到的分析结果就是可能性最大的分析。

## 5 结论

算法要求文法的所有规则都符合下面两种形式之一：词法规则的形式是X->word,而句法规则的形式是义X->YZ。这种文法形式被称为Chomsky范式, 虽然看上去是受限的，其实不然：任何上下文无关语法的都可以被自动转换为Chomsky范式。

CYK算法中表P占用的空间为其中n是句子中单词的数目，m是文法中非终结符的数目，其时间花费为O(n^2\*m)（由于m对于某一具体语法而言是常量，通常也被描述为 O(n^3))。虽然对于限制更加严格的文法可以找到更快的算法，但对于一般的上下文无关语法而言，已经没有比CYK更好的算法。事实上，让一个算法在O(n^3)的时间内完成分析，也是件非常需要技巧的工作，因为一个句子可能会有指数级数目的分析树。

PCFG 的问题在于它们是上下文无关的。也就意味着P(“eat a banana”)和P(“eat a bandanna”)之间的差别仅仅取决于P（Noun->“banana”)和P（Noun->“bandanna”)之间的差异，而不依赖于 “eat” 和相应宾语之间的关系。利用 2 阶或更高阶的马尔科夫模型，当给定一个足够大的语料库时，就会知道“eat a banana”的可能性更大。我们可以将 PCFG 和马尔科夫模型结合起来以达到最佳。估算句子概率最简单的方法是取两种模型分别计算出的概率的几何平均数。然后我们就知道到从文法和词法的角度上看 “eat a banana” 都是可能的。但这种方法依然无法从句子 “eat a slightly aging but still palatable banana” 中找出 “eat” 和 “banana” 之间的关系，因为在这里关系的距离超过了两个单词。增加马尔科夫模型的阶数也不能更精确地获取关系，要做到这点，可以利用词汇化的 PCFG（lexicalized PCFG)。

PCFG 的另一个问题是它们对于短句子有着强烈的偏置。在“Wall Street Journal”这样的语料库中，句子的平均长度是 25 个单词。但是，PCFG 通常会赋予短句子很高的概率，例如“He slept, ”。然而在杂志中我们更容易看到“It has been reported by a reliable source that the allegation that he slept is credible.” 这样的长句。看上去杂志中的短语确实不是上下文无关的，相反，作者会对于句子有一个预期长度，并把这个长度作为他书写句子的全局的软约束。这些在 PCFG 中很难被反映出来。