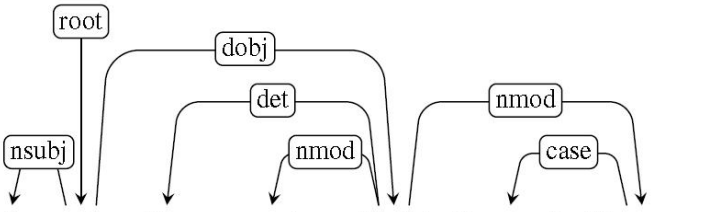
14. 依存解析

前两章的重点是上下文无关的语法以及它们在自动生成基于成分的表示中的使用。在这里，我们提出了另一组语法形式主义，称为**依存语法(dependency grammars)**，这是相当重要的当代语音和语言处理系统。在这些形式主义中，短语成分和短语结构规则并不直接起作用。相反，一个句子的句法结构仅由句子中的词(或 lemmas，即词元)和包含在词之间的一组相关的定向二元语法关系来描述。

下面的图说明了使用依存关系解析社区中常用的标准图形化方法进行的依存关系风格分析。



(14.1)

单词之间的关系，在句子上方，用带方向、带标签的圆弧(从中心词到依存)来表示。我们称其为**类型依存(typed dependency)**结构，因为标签是从语法关系的固定目录中提取的。它还包括一个根节点，它显式地标记出树的根，即整个结构的中心词(head)。

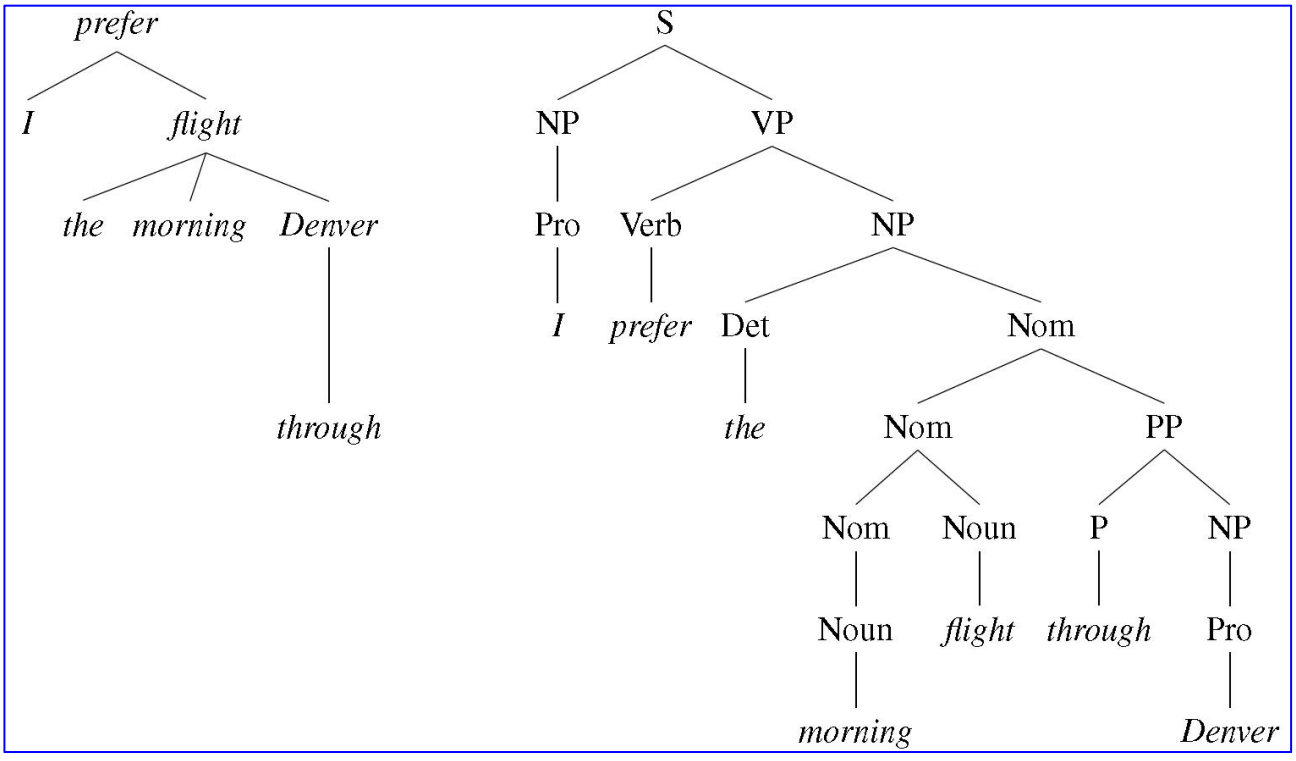


图 14-1：一个依存类型的解析及其相应的基于成分的分析

图注：针对句子：I prefer the morning flight through Denver。

图 14.1 显示了与树一样的依存关系分析，以及第十二章中给出的相应的短语结构分析。注意在依存解析中没有对应于短语成分或词汇范畴的节点;依存解析的内部结构仅由句子中词汇项之间的有向关系构成。这些关系直接编码了通常隐藏在更复杂的短语结构解析中的重要信息。例如，动词 prefer 的论元直接连接依存结构，而它们与主要动词的连接在短语结构树中更遥远。类似地，morning 和 Denver，flight 的修饰语，在依存结构中直接与之关联。

依存语法的主要优点是它们能够处理形态丰富且具有相对**自由词序(free word order)**的语言。例如，

捷克语的单词顺序比英语的灵活得多。语法宾语可能出现在位置状语之前或之后。对于语法树中可能出现此类副词短语的每个可能位置，短语结构语法都需要一个单独的规则。基于依存的方法只有一种连接类型，该类型代表此特定的状语关系。因此，依存语法方法从单词顺序信息(仅表示解析所需的信息)中抽象出来。

基于依存关系的方法的另一个实际动机是，依赖于中心词的关系为谓词及其论元之间的语义关系提供了近似值，从而使谓词直接可用于许多应用程序，例如共指解析、问题回答和信息提取。基于成分的解析方法提供了类似的信息，但通常必须通过第 12 章中讨论的查找规则等技术从树中提取出这些信息。

在下面几节中，我们将更详细地讨论在依存解析中使用的关系清单，以及这些依存结构的形式基础。然后，我们将继续讨论用于自动生成这些结构的主要算法系列。最后，我们将讨论如何评估依存分析器，并指出在语言处理应用程序中使用它们的一些方法。

14.1. 依存关系

语法关系(grammatical relation)的传统语言学概念为包含这些依存结构的二元关系提供了基础。这些关系的论元包括一个**中心词(head)**和一个**依存(dependent)**。在成分结构的背景下，我们已经在第 12 章和附录 C 中讨论了中心词的概念。在那里，一个成分的中心词是较大成分的中心组织词(例如，名词短语中的主要名词，或动词短语中的动词)。成分中的其余单词是其中心词的直接或间接依存项。在基于依存的方法中，通过将中心词直接连接到依存于它们的单词，绕过对成分结构的需求，从而使中心词-依存关系很明确。

除了指定中心词-依存偶对之外，按照依存相对于中心词扮演的角色，依存语法还允许我们进一步分类语法关系或语法功能的种类。诸如主语，直接宾语和间接宾语之类的熟悉概念都是我们想到的那种关系。在英语中，这些概念与句子中的位置和成分类型都紧密相关(但又不确定)，因此这些概念在短语结构树中发现的信息有点多余。但是，在更灵活的语言中，直接基于这些语法关系编码的信息至关重要，因为基于短语的成分句法几乎没有帮助。

毫不奇怪，语言学家已经发展出了关系分类，远远超出了我们熟悉的主语和宾语的概念。虽然理论与理论之间有相当大的差异，但有足够的共性，使开发一个计算上有用的标准成为可能。**通用依存(Universal Dependency)**项目(Nivre 等人 2016b)提供了一个依存关系清单，这些依存关系是语言驱动的、计算有用的和跨语言适用的。图 14.2(de Marneffe 等人，2014)显示了该工作的关系子集。

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

图 14-2：选自通用依存集合的依存关系

图 14.3 提供了一些例子来说明所选的关系。

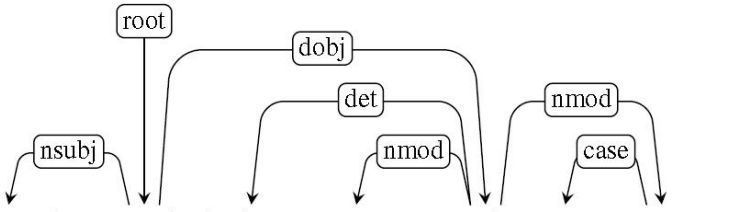
通用依存关系方案中所有关系的动机都超出了本章的范围，但是经常使用的关系的核心集可以分为两类：从句关系，描述关于谓语(通常是动词)的句法作用 ；修饰符关系，这些关系对单词可以修改其中心词

的方式进行了分类。

Relation	Examples with head and dependent
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which flight was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

图 14-3：核心通用依存关系的例子

考虑以下示例语句：



(14.2)

从句关系 NSUBJ 和 DOBJ 标识谓词 cancel 的主语和直接宾语，而 NMOD，DET 和 CASE 关系表示名词 Flight 和 Houston 的修饰语。

14.2. 依存形式

在它们最一般的形式中，我们正在讨论的简单有向图的依存结构是 G=(V,A)，其中：V 是顶点(Vertices)的集合；A 弧线(Arcs)的集合（由有序的、成对的顶点组成）。

在大多数情况下，我们将假定顶点集合 V，完全对应于给定句子中的单词集合。然而，它们也可能对应于标点符号，或者在处理形态复杂的语言时，顶点集可能由词干和词缀组成。弧线集合 A 捕获了 V 中元素之间的中心词-依存和语法功能的关系。

这些依存结构的进一步约束是特定于潜在的语法理论或形式主义的。更常见的限制是结构必须是连接的，有指定的根节点，并且是非循环的或平面的。与本章讨论的解析方法最相关的是对根树(rooted tree)的常见的、计算驱动的限制。也就是说，**依存树(dependency tree)**是满足以下约束的有向图:

1.有单个指定的根节点，没有传入的弧线。

2.除了根节点之外，每个顶点都有一条传入的弧线。

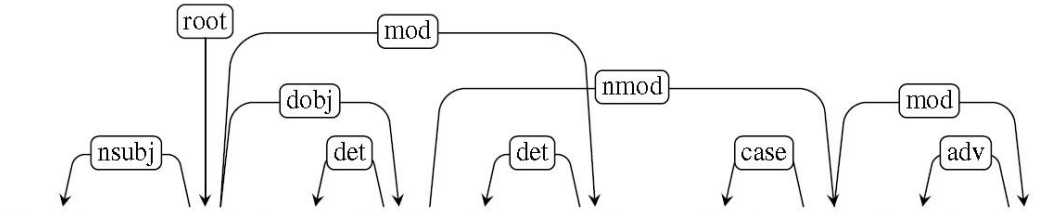
3.在 V 中，从根节点到每个顶点都有一条唯一的路径。

总的来说，这些约束确保每个单词都只有一个中心词，依存结构是连接的，并且只有一个根节点，从这个根节点可以沿着一个唯一的有向路径到达句子中的每个单词。

14.2.1. 投射性

投射性的概念强加了一个附加的约束，该约束源自输入中的单词顺序。如果句子中从中心词到从属关联词之间有一条路径，那么从中心词到从属关联词的弧线就是投射的。如果一个依存树上的所有弧线都是投射的，那么这个依存树就是投射的。到目前为止，我们看到的所有依存树都是投射的。然而，有许多完全有效的结构会导致非投射树，特别是在词序相对灵活的语言中。

考虑以下示例。



(14.3)

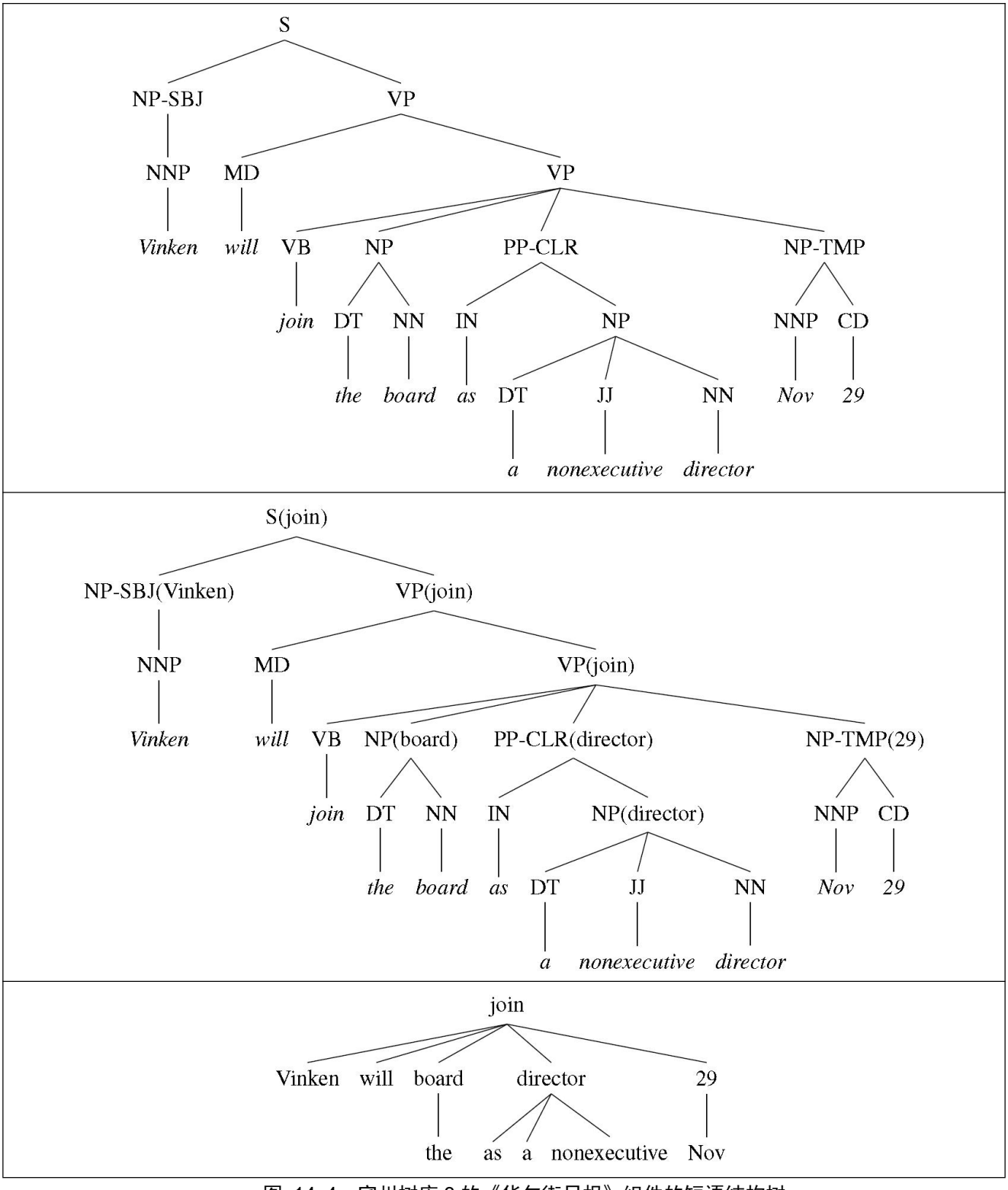


图 14-4：宾州树库 3 的《华尔街日报》组件的短语结构树

在这个例子中，从 flight 到它的修饰词 was 的弧线是非投射的，因为从 flight 到 this 和 morning 中间的单词之间没有路径。正如我们从这张图中看到的，投射性(和非投射性)可以通过我们绘制树的方式检测到。如果一个依存树没有交叉的边，那么它就是投射的。在这种情况下，如果不交叉将 morning 连接到其中心词的弧线，那么就无法将 flight 连接到其依存 was。

我们对投射性的关注来自两个相关问题。首先，最广泛使用的英语依存树库是通过使用查找规则从短语结构树库中自动得出的(第 12 章)。这样生成的树是从上下文无关的语法生成的，故可保证是投射的。

其次，最广泛使用的解析算法家族存在计算限制。第 14.4 节中讨论的基于转移的方法只能产生投射树，因此任何具有非投射结构的句子都必然包含一些错误。这个限制是第 14.5 节中描述的更灵活的基于图的解析方法的动机之一。

14.3. 依存树库

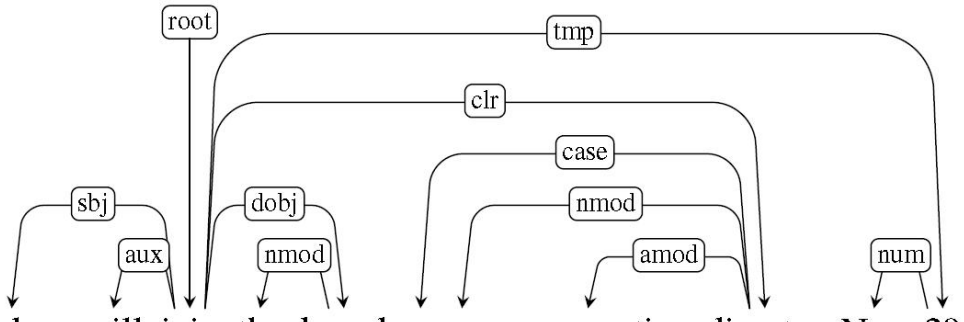
与基于成分的方法一样，树库在依存解析器的开发和评估中扮演着关键角色。依存树库的创建使用了与第 12 章中讨论的方法类似的方法——让人类注释器直接为给定的语料库生成依存结构，或者使用自动解析器提供初始解析，然后让注释器手动更正这些解析器。我们还可以使用确定性过程，通过使用中心词规则将现有的基于成分的树库转换为依存树。

在大多数情况下，已经为形态丰富的语言(如捷克语、印地语和芬兰语)创建了直接注释的依存树库，这些语言可以使用依存语法方法，其中捷克语的布拉格依赖树库(Bejcek 等人，2013)是最著名的成果。主要的英国依存树库主要是从现有资源中提取的，如宾州树库的《华尔街日报》部分(Marcus 等人，1993 年)。最近的 OntoNotes 项目(Hovy 等人 2006, Weischedel 等人 2011)扩展了这种方法，超越了传统的新闻文本，包括会话电话语音、网络日志、usenet 新闻组、广播和英语、汉语和阿拉伯语的脱口秀。

从成分到依存结构的转换过程有两个子任务:识别结构中所有的中心词-依存关系，并为这些关系识别正确的依存关系。第一个任务很大程度上依赖于第 12 章中讨论的中心词规则的使用，该规则最初是为词汇化概率分析器开发的(Magerman 1994, Collins 1999, Collins 2003)。下面是 Xia 和 Palmer(2001)提出的一个简单而有效的算法。

1. 使用适当的中心词规则在短语结构中标记每个节点的中心词孩子节点。
2. 在依存结构中，使每个非中心词孩子节点的中心词依赖于中心词孩子节点的中心词。

当短语结构解析以语法关系和功能标记的形式包含附加信息时(如宾州树库的情况)，这些标记可用于标记结果树中的边。当应用到图 14.4 中的解析树时，该算法将产生示例 14.4 中的依存结构。



(14.4)

这些提取方法的主要缺点是它们受到原始成分树中所呈现信息的限制。其中最重要的问题是未能将形态信息与短语结构树相结合，不能很容易地表示非投射结构，以及大多数名词短语缺乏内部结构，这反映在大多数树库语法中普遍使用的扁平化规则。由于这些原因，除了英语之外，大多数依存树库都是直接使用人类注释器开发的。

14.4. 基于转移的依存解析

我们的第一种依存解析方法是由一种称为**位移减少(shift-reduce)**解析的基于堆栈的方法驱动的，这

种方法最初是为分析编程语言而开发的(Aho 和 Ullman, 1972)。这种经典方法简单而优雅，使用上下文无关的语法、堆栈和要解析的符记列表。输入符记依次移动到堆栈上，堆栈的顶部两个元素与语法规则的右侧进行匹配;当找到一个匹配的元素时，匹配的元素将在堆栈上被匹配规则左边的非终端符替换(减少)。在为依存解析调整这种方法时，我们放弃了显式使用语法，并更改了减少操作，以便在一个单词和它的中心词之间引入依存关系，而不是向解析树添加非终端符。更具体地说，减少操作被替换为两个可能的操作:断言堆栈顶部的单词和它下面的单词之间的中心词-依存关系，或者反之亦然。图 14.5 演示了这种解析器的基本操作。

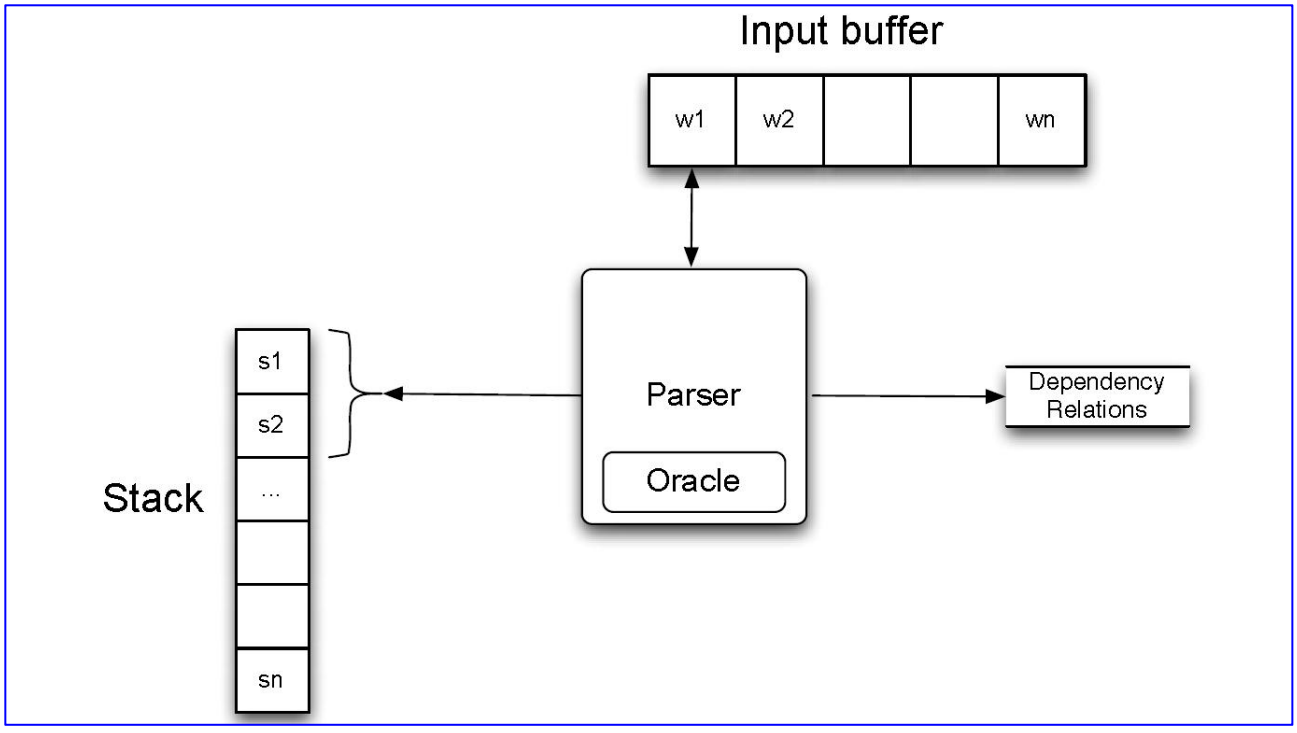


图 14-5：基于转移的基本解析器

图注：解析器检查堆栈的顶部两个元素，并根据检查当前配置的预言机选择一个操作。

基于转移的解析中的一个关键概念是**配置(configuration)**，它由堆栈、单词的输入缓冲区(或者符记)、以及一组表示依赖树的关系组成。在此框架下，解析过程由一系列通过可能配置空间的转移组成。这个过程的目标是找到一个最终的配置，其中所有的单词都已被解释，并且合成了一个适当的依存树。

为了实现这样的搜索，我们将定义一组转移操作符，当把它们应用于配置时，将产生新的配置。在这种设置下，我们可以将解析器的操作视为在配置空间中搜索从开始状态到期望的目标状态的一系列转移。在这个过程的开始，我们创建了一个初始配置，在该配置中，堆栈包含根(ROOT)节点，单词列表用句子中的单词集或词元化的符记进行初始化，并且创建一个空的关系集来表示解析。在最终的目标状态中，堆栈和单词列表应该是空的，关系集将表示最终的解析。

在基于转移的解析的标准方法中，用于产生新配置的操作符非常简单，并且与通过从左到右单次检查输入中的单词来创建依存关系树时可能采取的直观操作相对应 (Covington,2001):

•指定当前的单词作为之前看到的单词的中心词，

•指定一些以前看到的单词作为当前单词的中心词，

•或者推迟对当前单词的处理，将其添加到存储中以便后续处理。

为了使这些操作更精确，我们将创建三个转移操作符，它们将对堆栈顶部的两个元素进行操作：

•LEFTARC：在堆栈顶部的单词与其正下方的单词之间建立中心词-依存关系，从堆栈中删除低位字；

•RIGHTARC：在堆栈中的第二个单词和顶部的单词之间建立中心词-依存关系，删除堆栈顶部的单词；

•SHIFT：从输入缓冲区的开头删除单词并将其推入堆栈。

这组特定的操作符实现了基于过渡的解析的**弧线标准(arc standard)**方法(Covington 2001，Nivre 2003)。此方法有两个显著的特征：过渡操作符仅在堆栈顶部声明元素之间的关系，并且一旦分配了元素

的中心词，就将其从堆栈中移除，无法进行进一步处理。就像我们将看到的那样，还有其他的转移系统可以演示不同的解析行为，但是弧线标准方法非常有效并且易于实现。

为了确保正确地使用这些操作符，我们需要为它们的使用添加一些先决条件。首先，因为根据定义，根节点不能有任何传入的弧线，所以我们将添加一个限制，即当根是堆栈的第二个元素时，不能应用 LEFTARC 操作符。其次，两个减少操作符都要求在堆栈上应用两个元素。有了这些转移操作符和先决条件，基于转移的解析器的规范就相当简单了。图 14.6 给出了基本算法。

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)         ; choose a transition operator to apply
    state ← APPLY(t, state)   ; apply it, creating a new state
  return state
```

图 14-6：通用的基于转移的依存关系解析器算法

在每一步中，解析器都要咨询**预言机(oracle)**(我们稍后再讨论这个问题)，该预言机提供了给定当前配置的正确转移操作符。然后将该操作符应用到当前配置，生成一个新配置。当句子中的所有单词都被消费完，并且根节点是堆栈上唯一剩下的元素时，进程结束。

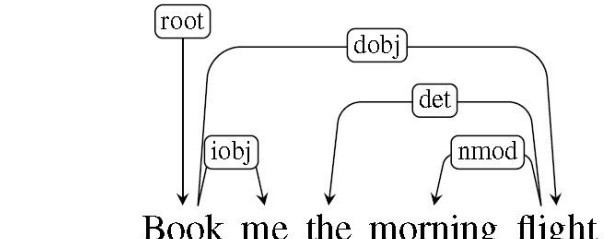
基于转移的解析器的效率从算法中可以明显看出。复杂性在句子长度上是线性的，因为它基于从左到右的单一遍历句子中的单词。更具体地说，每个单词必须先移到堆栈上，然后再减少。

请注意，与第 12 和 13 章中讨论的基于动态规划和基于搜索的方法不同，该方法是一种简单的贪心算法-预言机在每个步骤中都提供了一个选择，并且解析器继续该选择，没有探索其他选择，没有使用回溯，最后返回单个解析。

图 14.7 演示了解析器的操作，其中的转移序列将导致下面示例的解析。

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

图 14-7：基于转移的解析的跟踪



(14.5)

让我们考虑第 2 步的配置状态，在单词 me 被压入堆栈之后。

Stack	Word List	Relations
[root, book, me]	[the, morning, flight]	

这里应用的正确操作符是 RIGHTARC，它将 book 指定为 me 的中心词，并将 me 从堆栈中弹出，从而产生以下配置。

Stack	Word List	Relations
[root, book]	[the, morning, flight]	(book → me)

在多次使用 SHIFT 和 LEFTARC 操作符之后，第 6 步的配置如下所示:

Stack	Word List	Relations
[root, book, the, morning, flight]	[]	(book → me)

这里，所有剩余的字都已经传递到堆栈上，剩下要做的就是应用适当的减少操作符。在当前配置中，我们使用 LEFTARC 操作符产生以下状态。

Stack	Word List	Relations
[root, book, the, flight]	[]	(book → me) (morning ← flight)

此时，这个句子的解析由以下结构组成。

(14.6)

在检查图 14.7 中的序列时，有几件重要的事情需要注意。首先，给定的序列不是唯一可能导致合理解析的序列。通常，可能有不止一条路径导致相同的结果，而且由于歧义，可能有其他转移序列导致不同的同样有效的解析。

其次，我们假设预言机总是在解析的每个点上提供正确的操作符——这个假设在实践中不太可能是正确的。因此，鉴于这种算法的贪心本性，不正确的选择将导致不正确的解析，因为解析器没有机会返回并寻求其他选择。第 14.4.2 节将介绍几种技术，这些技术允许基于转移的方法更充分地探索搜索空间。

最后，为了简单起见，我们在没有依存关系标签的情况下演示了这个示例。要生成标记树，我们可以用依存项标签参数化 LEFTARC 和 RIGHTARC 操作符，如 LEFTARC(NSUBJ)或 RIGHTARC(DOBJ)。这等价于将原来的三个转移操作符的集合扩展为一个包含所使用的依存关系集合中的每个关系的 LEFTARC 和 RIGHTARC 操作符的集合，再加上一个额外的 SHIFT 操作符。当然，这使得预言机的工作更加困难，因为它现在有更多的操作符可供选择。

14.4.1. 创建一个预言机(Oracle)

最先进的(SOTA)基于转移的系统使用监督机器学习方法来训练扮演预言机角色的分类器。给定适当的训练数据，这些方法学习从配置映射到转移操作符的函数。

与所有有监督的机器学习方法一样，我们将需要访问适当的训练数据，我们将需要提取对做出决策的特征有用的特征。这个训练数据的源将是包含依存树的代表性树库。这些特性将包括我们在第 8 章的词类标记中遇到的许多相同的特性，以及附录 C 中用于统计解析模型的特性。

生成的训练数据

让我们从图 14.6 中的算法重新审视预言机，以充分了解学习问题。预言机将配置作为输入，并返回转移操作符作为输出。因此，要训练分类器，我们将需要与转移操作符(即 LEFTARC，RIGHTARC 或 SHIFT)

配对的配置。不幸的是，树库将整个句子与相应的树配对，因此它们无法直接提供我们所需的内容。

为了生成所需的训练数据，我们将以一种巧妙的方式采用基于预言机的解析算法。我们将向预言机提供要解析的训练语句以及来自树库的相应参考解析。为了产生训练实例，我们将通过运行算法并依靠新的**训练预言机(training oracle)**来为每个连续的配置提供正确的转移操作符，从而模拟解析器的操作。

要了解其工作原理，首先让我们回顾一下解析器的操作。它以默认的初始配置开始，其中堆栈包含ROOT，输入列表仅是单词列表，关系集为空。LEFTARC 和 RIGHTARC 操作符分别将堆栈顶部的单词之间的关系添加到给定句子累积的关系集。由于我们对每个训练句子都有一个黄金标准的参考解析，因此我们知道对于给定句子哪些依存关系有效。因此，当解析器逐步执行一系列配置时，我们可以使用参考解析来指导操作符的选择。

更精确地说，给定一个参考解析和配置，预言机的训练过程如下:

•如果在给定参考解析和当前配置的情况下产生正确的中心词-依存关系，则选择 LEFTARC;

•否则，如果(1)在给定参考解析的情况下产生正确的中心词-依存关系，并且(2)已分配了堆栈顶部单词的所有从属关系，则选择 RIGHTARC;

•否则，选择 SHIFT。

需要限制选择 RIGHTARC 操作符，以确保在未将所有单词分配给其依赖项之前，不会从堆栈中弹出一个单词，从而避免该单词被进一步处理。

更正式地说，在训练期间，预言机可以访问以下信息：

•当前配置有一个栈和一组依存关系 R_c

•一个由顶点 V 和依存关系 R_p 组成的参考解析

有了这些信息，预言机会选择如下的转换:

LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$

RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t. (S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$

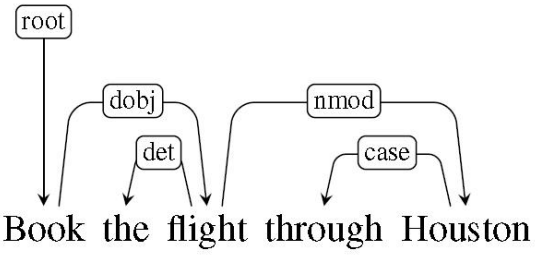
SHIFT: **otherwise**

让我们通过图 14.8 所示的示例来完成这个过程的步骤。

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

图 14-8：用参考解析生成训练项

图注：通过使用给定的参考解析来模拟解析，生成由配置/预测动作对组成的训练项



(14.7)

在第 1 步，LEFTARC 在初始配置中不适用，因为它断言一个关系(root←book)不在参考答案中;RIGHTARC 确实断言了包含在最终答案(root → book)中的关系，然而 book 还没有附加到它的任何依存项上，所以我们不得不服从（defer），留下 SHIFT 作为唯一可能的行动。在接下来的两个步骤中也存在相同的条件。在步骤 3 中，LEFTARC 被选中以连接 the 到其中心词。

现在考虑第 4 步中的情况。

Stack	Word buffer	Relations
[root, book, flight]	[through, Houston]	(the ← flight)

在这里，我们可能想在 book 和 flight 之间添加一个依存关系，该关系出现在参考解析中。但现在这样做可以防止 Houston 的后续附着，因为 flight 将从堆栈中删除。幸运的是，选择 RIGHTARC 的先决条件阻止了此选择，我们再次将 SHIFT 留为唯一可行的选择。其余选择完成了此示例所需的一组操作符。

回顾一下，我们通过在参考依存树的上下文中模拟解析器的操作，从树库中获得由配置-转移偶对组成的适当训练实例。我们可以确定地在每个训练示例中记录正确的解析器操作，从而创建我们需要的训练集。

特征

生成了适当的训练实例(配置-转移偶对)之后，我们需要从配置中提取有用的特性，以便训练分类器。用于训练基于转移的系统的特征因语言、体裁和所使用的分类器类型而不同。例如，取决于正在处理的语言，诸如主语或直接宾语上的案例标记之类的语态特征可能或多或少重要。也就是说，我们已经看到的关于词类标记和部分解析的基本特性已经被证明在训练各种语言的依存解析器时非常有用。词形、词元和词类都是强有力的特征，就像中心词以及与中心词的依存关系一样。

在基于转移的解析框架中，需要从组成训练数据的配置中提取这些特性。回想一下，配置由三个元素组成:堆栈、缓冲区和当前关系集合。原则上，这些元素的任何属性都可以用通常的训练方法表示为特征。然而，为了避免稀疏性和鼓励泛化，最好将学习算法集中在解析过程中每个点上最有用的决策制定方面。因此，基于转移的解析的特征提取的重点是堆栈的顶层、靠近缓冲区前端的单词以及已经与这些元素相关联的依存关系。

通过将简单的特征(如单词形式或词类)与配置中的特定位置相结合，我们可以使用我们在情感分析和词类标注中已经遇到的特征模板(feature template)的概念。特征模板允许我们从一个训练集自动生成大量的特定特征。作为一个例子，考虑以下基于配置中的单个位置的特征模板。

$$\langle s_1.w,op\rangle,\ \langle s_2.w,op\rangle\langle s_1.t,op\rangle,\ \langle s_2.t,op\rangle$$
$$\langle b_1.w,op\rangle,\ \langle b_1.t,op\rangle\langle s_1.wt,op\rangle \quad (14.8)$$

在这些示例中，单个特征被表示为位置.属性(location.property)，其中 s 表示堆栈，b 表示单词缓冲区，r 表示关系集合。位置的个别属性包括 w 表示词形，l 表示词元，t 表示词类。例如，对应于堆栈顶部的单词形式的特征将被表示为 $s_1.w$，以及位于缓冲区 $b_1.t$ 前端的词类部分。我们也可以通过连接将单个的特征组合成更具体的特征，这可能证明是有用的。例如，$s_1.wt$ 指定的特性表示单词的形式与堆栈顶部单词的词类相连接。最后，op 代表所讨论的训练示例的转移操作符(即，训练实例的标签)。

让我们考虑一下上面给出的一组简单的单个元素特征模板，这些模板在下面的中间(intermediate)配置的上下文中，这些中间配置来自于一个训练预言机示例 14.2。

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

这里正确的转移是 SHIFT(在继续之前你应该说服自己)。将我们的一组特性征模板应用到这个配置中，将产生以下一组实例化的特征。

$$\langle s_1.w = \text{flights},op = \text{shift}\rangle \quad (14.9)$$

<s_2.w = canceled,op = shift>
<s_1.t = NNS,op = shift>
<s_2.t = VBD,op = shift>
<b_1.w = to,op = shift>
<b_1.t = TO,op = shift>
<s_1.wt = flightsNNS,op = shift>

考虑到左右圆弧转换操作在堆栈的顶部两个元素上，将这些位置的属性组合起来的特征更加有用。例如，像 $s_1.t \circ s_2.t$ 这样的特征将堆栈顶部的单词的词类标记与它下面的单词的标记连接起来。

$$\langle s_1.t \circ s_2.t = NNSVBD, op = shift \rangle \tag{14.10}$$

毫不奇怪，如果两个属性是有用的，那么三个或更多应该更好。

图 14.9 给出了一组已经使用的基本特征模板(Zhang 和 Clark 2008, Huang 和 Sagae 2010, Zhang 和 Nivre 2011)。注意，这些特征中有一些使用了动态特征，例如在解析过程的早期步骤中预测的中心词和依存关系等特性，而不是从输入的静态属性派生的特征。

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

图 14-9：训练基于转移的依存分析器的标准特征模板

图注：s_n 是指堆栈上的一个位置，b_n 是指单词缓冲区中的一个位置，w 是指输入的单词形式，t 是指输入的词类部分。

学习

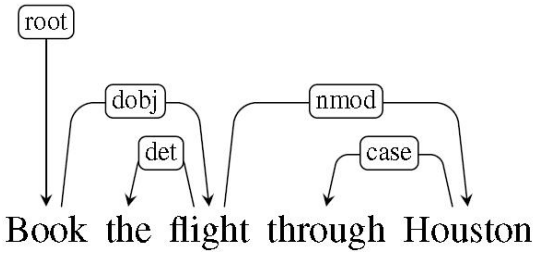
多年来，训练基于转移的依存分析器的主要方法是多元逻辑回归和支持向量机，这两种方法都可以有效地利用上一节中描述的大量稀疏特征。最近，第 8 章中描述的神经网络或深度学习方法已成功应用于基于转移的解析(Chen 和 Manning, 2014)。这些方法消除了对复杂的、手工制作的特性的需要，并且在克服通常与训练基于转移的解析器相关的数据稀疏性问题方面特别有效。

14.4.2. 基于转移解析中的高级方法

基本的基于转移的方法可通过多种方式加以阐述，通过解决该方法中一些最明显的缺陷来提高性能。

替代转移系统

上面描述的弧线标准转换系统只是许多可能的系统之一。一种常用的替代方法是弧线急转(arc eager)系统。弧线急转方法得名于它能够比弧线标准方法更快地断言右向(rightward)关系。为了看到这一点，让我们回顾一下例子 14.7 中的弧线标准轨迹，在这里重复。



在这个分析中考虑 book 和 flight 之间的依存关系。如图 14.8 所示，尽管 book 和 flight 在第 4 步更早地到达堆栈上，但弧线标准方法将在第 8 步确定这种关系。这个关系在这一点上不能被捕获的原因是由于存在后名词类(postnominal)修饰语 through Houston。在弧线标准的方法中，依存项一旦被分配了中心

词就会被从堆栈中移除。如果 flight 在步骤 4 中被指定 book 作为它的中心词，它将不能再担任 Houston 的中心词。

虽然这个延迟在这个例子中不会造成任何问题，但通常一个单词等待被分配到它的中心词的时间越长，出错的机会就越多。弧线急转系统解决了这个问题，它允许单词尽可能早地连接到它们的中心词，在所有依赖于它们的后续单词被看到之前。这是通过对 LEFTARC 和 RIGHTARC 操作符的微小更改和添加一个新的 REDUCE 操作符来实现的。

•LEFTARC:断言在输入缓冲区前面的单词和堆栈顶部的单词之间的中心词-依存关系;弹出堆栈。

•RIGHTARC:在堆栈顶部的单词和输入缓冲区前面的单词之间断言一个中心词-依存关系;将输入缓冲区前面的单词移到堆栈上。

•SHIFT:从输入缓冲区的前面移除单词，并将其压入堆栈。

•REDUCE:弹出堆栈。

LEFTARC 和 RIGHTARC 操作符应用于堆栈的顶部和输入缓冲区的前端，而不是像弧线标准方法中那样应用于堆栈的顶部两个元素。现在，RIGHTARC 操作符将依赖项从缓冲区移动到堆栈，而不是删除它，从而使它可以作为后续单词的中心词。新的 REDUCE 操作符从堆栈中移除顶部元素。这些更改加在一起允许一个单词被急切地分配它的中心词，并且仍然允许它作为后续依存的中心词。图 14.10 所示的轨迹说明了这个例子的新决策序列。

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

图 14-10：使用弧线急转转移操作符的处理轨迹

图注：针对句子：Book the flight through Houston

除了演示弧线急转转换系统之外，这个示例还演示了整个基于转移方法的强大功能和灵活性。我们能够在无需对底层解析算法做任何更改的情况下切换到一个新的转移系统。这种灵活性的发展导致了一组不同的转移系统,解决句法和语义的不同方面，包括:分配词类标记(Choi 和 Palmer，2011),允许生成非投射依存结构(Nivre, 2009)，分配语义角色(Choi 和 Palmer, 2011b)，以及解析包含多种语言的文本(Bhat 等人，2017)。

波束搜索

前面讨论过的基于转移的方法的计算效率源于这样一个事实，即它只通过一个句子，贪心地做出决定而不考虑其他选择。当然，这也是其最大弱点的根源——一旦做出了决定，就无法撤销，即使在之后的句子中出现了压倒性的证据。另一种方法是系统地探索备选决策序列，在这些选择中选择最好的。这种搜索的关键问题是管理大量的潜在序列。波束搜索通过将宽度优先搜索策略与启发式过滤器相结合来实现这一点，启发式过滤器修剪搜索边界以保持在一个固定大小的波束宽度内。

在将波束搜索应用于基于转移的解析时，我们将详细说明图 14.6 中给出的算法。我们不会在每次迭代中选择单个最佳转移操作符，而是将所有适用的操作符应用于议程上的每个状态，然后对结果进行评分。然后，在波束中必须有空间的约束下，我们将这些新配置中的每一个都添加到边界。只要议程的大小在指定的波束宽度内，我们就可以向议程添加新的配置。一旦议程达到极限，我们将仅添加比议程上最差的配置更好的新配置(删除最糟糕的元素，使我们保持在极限之内)。最后，为了确保我们能检索到议程上的最

佳状态，只要议程上存在非最终状态，while 循环就会继续。

波束搜索方法需要一个比我们使用的贪心算法更详细的评分概念。在这里，我们假设使用监督机器学习训练的分类器将作为一个预言机，根据从当前配置中提取的特征选择最佳的转移操作符。不管具体的学习方法是什么，这个选择都可以看作是给所有可能的转移打分，然后选出最好的一个。

$$\hat{T}(c) = argmaxScore(t, c)$$

使用波束搜索，我们现在正在搜索决策序列的空间，因此根据整个历史来为配置评分是有意义的。更具体地说，我们可以将新配置的分数定义为其前身的分数加上用于生成该配置的操作符的分数。

$$\begin{aligned} ConfigScore(c_0) &= 0.0 \\ ConfigScore(c_i) &= ConfigScore(c_{i-1}) + Score(t_i, c_{i-1}) \end{aligned}$$

这个分数用于筛选议程和选择最终答案。基于转移的句法分析的波束搜索新版本如图 14.11 所示。

```
function DEPENDENCYBEAMPARSE(words, width) returns dependency tree

  state ← {[root], [words], [], 0.0}     ;initial configuration
  agenda ← ⟨state⟩                       ;initial agenda

  while agenda contains non-final states
    newagenda ← ⟨⟩
    for each state ∈ agenda do
      for all {t | t ∈ VALIDOPERATORS(state)} do
        child ← APPLY(t, state)
        newagenda ← ADDTOBEAM(child, newagenda, width)
    agenda ← newagenda
  return BESTOF(agenda)

function ADDTOBEAM(state, agenda, width) returns updated agenda

  if LENGTH(agenda) < width then
    agenda ← INSERT(state, agenda)
  else if SCORE(state) > SCORE(WORSTOF(agenda))
    agenda ← REMOVE(WORSTOF(agenda))
    agenda ← INSERT(state, agenda)
  return agenda
```

图 14-11：波束搜索应用于基于转移的依存解析

14.5. 基于图的依存解析

基于图的依存解析方法在可能的树空间中搜索给定的句子，搜索最大化某个分数的树。这些方法将搜索空间编码为有向图，并利用图论的方法在空间中搜索最优解。更正式地说，给定一个句子 S，我们要在 $\mathcal{G}_S$ 中寻找最好的依存树，即这个句子的所有可能的树的空间，来最大化某个分数。

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}}\, score(t, S)$$

与附录 C 中讨论的上下文无关解析的概率方法一样，树的总体得分可以看作是树各部分得分的函数。本节的重点是**边因子(edge-factored)**方法，其中树的分数基于组成树的边的分数。

$$score(t,S) = \sum_{e \in t} score(e)$$

使用基于图的方法有几个动机。首先，与基于转移的方法不同，这些方法能够生成非投射树。尽管对英语来说，投射性不是一个重要的问题，但对世界上许多语言来说，它绝对是一个问题。第二个动机与解析的准确性有关，尤其是对较长的依存关系而言。从经验上看，基于转移的方法在依存关系较短的情况下具有较高的准确性，但随着中心词与依存之间距离的增加，准确性显著下降(McDonald 和 Nivre, 2011)。基于图的方法通过对整个树进行评分来避免这个困难，而不是依赖贪心的局部决策。

下一节研究一种广泛研究的方法，该方法基于对加权有向图使用**最大生成树(maximum spanning tree，MST)**算法。然后我们讨论通常用于评分树的特征，以及用于训练评分模型的方法。

14.5.1. 解析

这里描述的方法使用一个有效的贪心算法在有向图中寻找最优生成树。给定一个输入句子，它首先构造一个全连接的、有权的、有方向的图，其中顶点是输入词，有向边代表所有可能的中心词-依存赋值。一个附加的根节点包含了指向所有其他顶点的出边。图中的权重反映了由训练数据生成的模型提供的每个可能的中心词-依存的得分。给定这些权重，从根产生的该图的最大生成树表示句子的首选依赖项解析。图 14.12 所示的 Book that flight 的有向图，与所需解析对应的最大生成树用蓝色表示。为了便于说明，这里我们将重点讨论无标记的依赖项解析。第 14.5.3 节将讨论基于图的标记解析方法。

在描述算法之前，考虑关于有向图及其生成树的两种直觉是有用的。第一个直觉是从这样一个事实开始的:一个生成树中的每个顶点都有一条传入的边。由此可以得出，每一个生成树的连通分量也会有一条进来的边。第二种直觉是，边的分数的绝对值对确定最大生成树并不重要。相反，重要的是进入每个顶点的边的相对权值。如果我们从每条进入一个给定顶点的边中减去一个常量，这对最大生成树的选择没有影响，因为每棵可能的生成树都会减少相同的数量。

算法本身的第一步非常简单。对于图中的每个顶点，选择一个得分最高的传入边(代表可能的中心词赋值)。如果这些边形成了一棵生成树，那么我们就完成了。更正式地说，给定原全连通图 G = (V,E)，一个子图 T = (V,F)是一棵生成树，如果它没有环且每个顶点(除根之外)都有一条入边。如果贪心选择过程产生了这样一棵树，那么它就是可能的最佳树。

不幸的是，这种方法并不总是生成树，因为所选的边集可能包含循环。幸运的是，在另一种多重发现的情况下，有一种直接的方法可以消除贪心选择阶段产生的循环。Chu 和 Liu(1965)和 Edmonds(1967)独立开发了一种方法，它从贪心选择开始，然后是一个优雅的递归清理阶段，消除了循环。

清理阶段首先通过从所有进入顶点的边的分数减去进入每个顶点的最大边的分数来调整图中的所有权值。这就是前面提到的直觉发挥作用的地方。我们对边的值进行了缩放，这样循环中边的权值就不会对任何可能生成树的权值产生影响。从进入顶点的每条边减去权值最大的边的值，结果在贪心选择阶段选中的所有边的权值为零，包括循环中涉及的所有边。

在调整了权重之后，该算法通过选择一个循环并将其折叠成一个新节点来创建一个新图。进入或离开循环的边被更改，以便它们现在进入或离开新折叠的节点。不循环的边被包括进来，在循环中的边被丢弃。

现在，如果我们知道这个新图的最大生成树，我们就得到了消去这个循环所需要的东西。最大生成树的边指向表示折叠循环的顶点，告诉我们删除哪条边以消除循环。我们如何找到这个新图的最大生成树呢?我们递归地把算法应用到新图上。这将产生一个生成树或一个带有循环的图。只要遇到循环，递归就可以继续。当每次递归完成时，我们展开折叠的顶点，恢复循环中的所有顶点和边(要删除的单条边除外)。

将所有这些放在一起，最大生成树算法包括贪心的边选择、边成本的重新评分和需要时的递归清理阶段。完整算法如图 14.13 所示。

图 14.14 通过 Book that flight 示例逐步介绍了算法。该图的第一行说明了贪心的边选择，其中选择的边以蓝色显示(对应于算法中的集合 F)。这导致了 that 和 flight 之间的循环。右图显示了使用输入每个节点的最大值的缩放后的权重。

第二行显示两个内容：将 that 和 flight 之间的循环分解到单个节点(标记为 tf)；新缩放的成本递归。递归中的贪心选择步骤生成了一棵生成树，它将 root 连接到 book，并生成一条边，将 book 连接到收缩节点。将收缩的节点展开，我们可以看到这样一条边，这条边对应原始图中从 book 到 flight 的边。这反过来告诉我们应该放下哪条边来消除循环。

在任意有向图上，此版本的 CLE 算法运行时间为 O(mn)，其中 m 为边数，n 为节点数。因为这个算法的特殊应用始于构造一个全连接图 $m = n^2$，产生 $O(n^3)$的运行时间。Gabow 等人(1986)提出了一个运行时间为 O(m+nlogn)的更高效的实现。

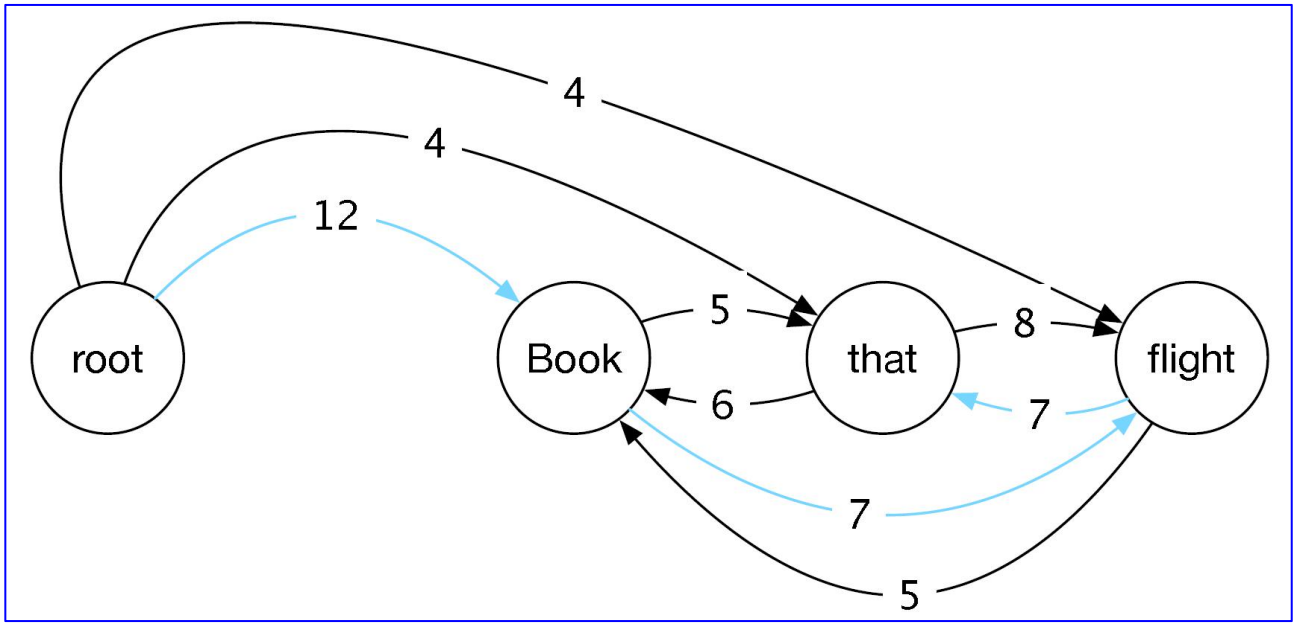


图 14-12：Book that flight 的初始根有向图

```
function MaxSpanningTree(G=(V,E), root, score) returns spanning tree

    F ← []
    T' ← []
    score' ← []
    for each v ∈ V do
        bestInEdge ← argmax_{e=(u,v)∈E} score[e]
        F ← F ∪ bestInEdge
        for each e=(u,v) ∈ E do
            score'[e] ← score[e] − score[bestInEdge]

        if T=(V,F) is a spanning tree then return it
        else
            C ← a cycle in F
            G' ← Contract(G, C)
            T' ← MaxSpanningTree(G', root, score')
            T ← Expand(T', C)
            return T

function Contract(G, C) returns contracted graph

function Expand(T, C) returns expanded graph
```

图 14-13：在加权有向图中寻找最大生成树的 CLE 算法

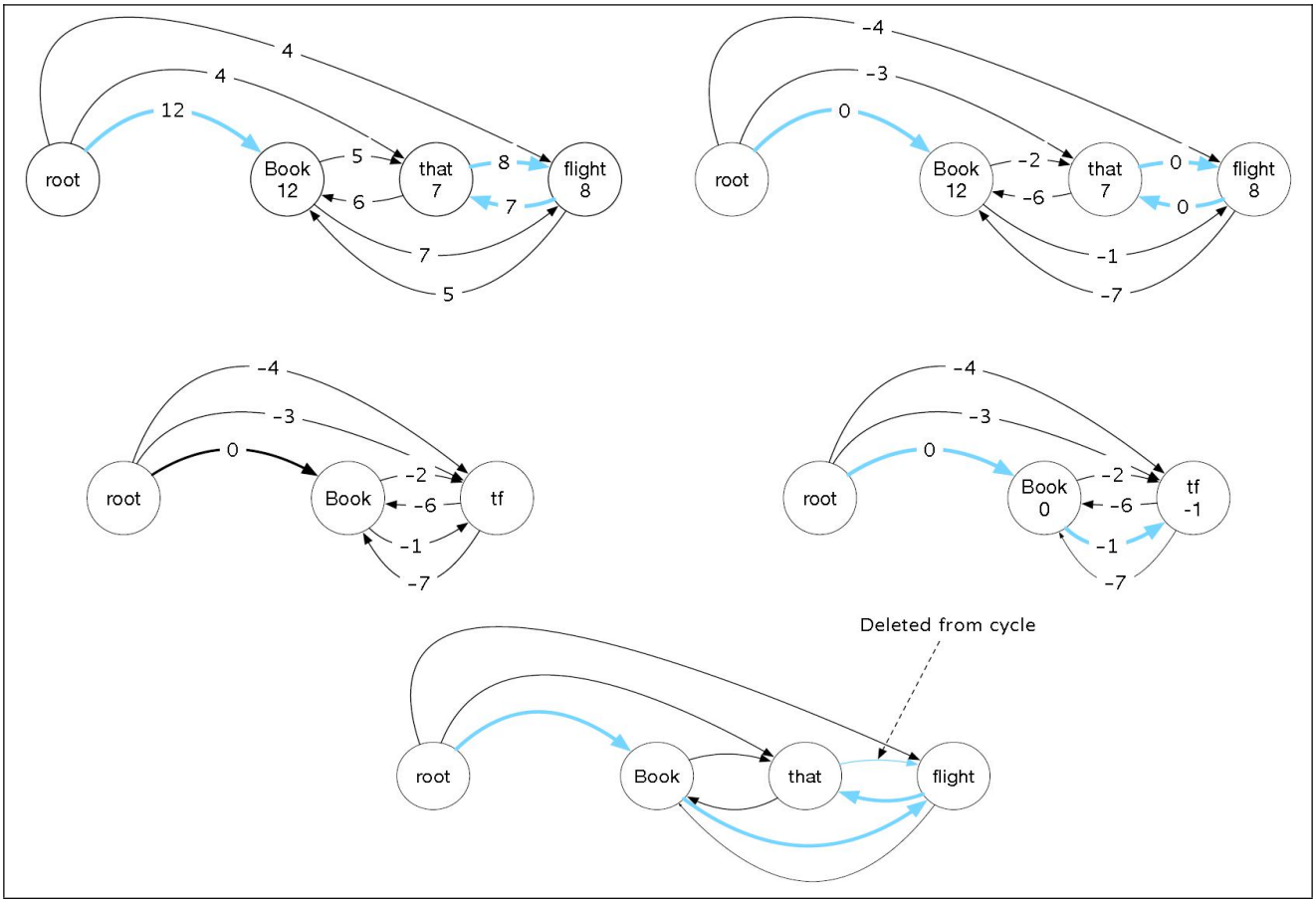


图 14-14：基于图的 CLE 算法示例

14.5.2. 特征和训练

给定一个句子 S 和一个候选树 T，边因子的解析模型将树的分数降低为构成树的边的分数之和：

$$score(S,T) = \sum_{e \in T} score(S,e)$$

每个边分数可以依次减少为从它提取的特征的加权之和：

$$score(S,e) = \sum_{i=1}^{N} w_i f_i(S,e)$$

或者更简洁：

$$score(S,e) = w \cdot f$$

有了这个公式，我们在训练解析器时面临两个问题:识别相关的特征和找到用于为这些特征评分的权重。用于训练边因子模型的特征与训练基于转移的解析器的特征相似(如图 14.9 所示)。这并不奇怪，因为在这两种情况下，我们都试图在单一关系的上下文中捕捉关于中心词和它们的依存的信息。总结前面的讨论，常用的特征包括:

- 词形、词元、中心词及其依存的词类。
- 从词前、词后和词间的上下文中衍生出相应的特征。
- 单词嵌入。
- 依存关系本身。
- 关系的方向(向右或向左)。

•从中心词到依存的距离。

与基于转移的方法一样，也经常使用预先选定的这些特征组合。

给定一组特征，我们的下一个问题是学习一组与每个特征对应的权值。与前面章节中讨论的许多学习问题不同，这里我们没有训练一个模型来将训练项与类标签或解析器动作关联起来。相反，我们试图训练一个模型，让正确的树比错误的树得分更高。对于这样的问题，一个有效的框架是使用基于推理的学习(inference-based learning)结合感知器学习规则。在这个框架中，我们使用初始权值的初始随机集从训练集中解析一个句子(即执行推理)。如果结果解析与训练数据中相应的树相匹配，我们就不对权重做任何处理。否则，我们会在错误解析中找到那些在参考解析中不存在的特征，然后根据学习速率降低它们的权重。我们对训练数据中的每句话都递增地这样做，直到权值收敛。

最先进的多语言解析算法是基于循环神经网络(RNN) (Zeman 等人 2017, Dozat 等人 2017)。

14.5.3. 基于图解析的高级问题

（待定）

14.6. 评估

与基于短语结构的解析一样，依存解析器的评估是通过测量它们在测试集上的工作情况来进行的。一个明显的度量标准是**精确匹配(exact match，EM)**——有多少句子被正确解析。这个度量相当悲观，大多数句子都被标错了。这样的度量不够细粒度，不足以指导开发过程。我们的度量标准必须足够敏感，以判断是否进行了实际的改进。

由于这些原因，评估依存解析器最常见的方法是有标签的和未标签的附着正确度。有标签的附着是指将一个词恰当地赋值给它的中心词，并建立正确的依存关系。未标签的附着只是查看分配的中心词的正确性，忽略了依存关系。给定一个系统输出和相应的参考解析，正确度就是输入中单词的百分比，这些单词被赋予了正确的中心词和正确的关系。这些度量标准通常称为**有标签的附着得分(LAS)**和**未标签的附着得分(UAS)**。最后，我们可用**标签正确度分数(LS)**，即带有正确标签的符记的百分比，忽略关系来自何处。

作为一个例子，考虑下面图 14.15 中所示的例子的参考解析和系统解析。

(14.11) Book me the flight through Houston.

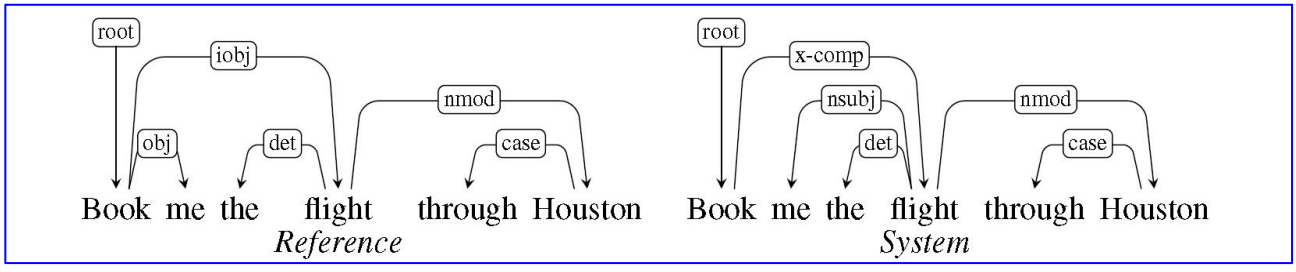


图 14-15：参考解析和系统解析示例

图注：句子 Book me the flight through Houston 的参考和系统解析，结果是 LAS 为 2/3,UAS 为 5/6。

系统正确地找到参考解析中存在的 6 个依存关系中的 4 个，并接收到 LAS 的 2/3。然而，系统发现的两个错误关系中的一个存在于 book 和 flight 之间，它们在参考解析中是中心词-依存关系;因此，该系统的 UAS 为 5/6。

除了附着评分以外，我们还可能对系统在整个开发语料库上对特定种类的依存关系(例如 NSUBJ)的性能表现有兴趣。在这里，我们可以利用第 8 章中介绍的精度和召回率概念，测量这两个指标：系统标记为 NSUBJ 的正确关系的百分比(精度)；系统实际上发现的开发集中存在的 NSUBJ 关系的百分比(召回率)。我们可以使用混淆矩阵来跟踪每种依存类型与另一种依存类型混淆的频率。

14.7. 总结

本章介绍了**依存语法**和**依存解析**的概念。以下是我们总结的要点:

•在**基于依存句法**方法中，一个句子的结构是用一组二元关系来描述的，这些关系保持在一个句子中的两个单词之间。在依存关系分析中没有直接编码更大的成分概念。

•**依存结构**中的关系捕获了句子中单词之间的**中心词-依存**关系。

•**基于依存的分析**为进一步的语言处理任务提供了直接有用的信息，包括信息提取、语义解析和问答。

•**基于转移的解析**系统采用贪心的基于堆栈的算法来创建依存结构。

•创建依存结构的基于图的方法是基于图论中**最大生成树**方法的使用。

•**基于转移**和**基于图**的方法都是使用**监督机器学习**技术开发的。

•树库提供训练这些系统所需的数据。**依存树库**可以由人工注释人员直接创建，也可以通过对短语结构树库的自动转换来创建。

•**依存解析器**的评估基于对保留开发和测试语料库测量的**有标签**的和**未标签**的**正确度分数**。

14.8. 文献和历史说明

基于依存的语法方法比相对较新的短语结构或成分语法要古老得多，这些语法多年来一直是理论和计算语言学的主要焦点。它起源于古希腊和印度的语言传统。当代的依存语法理论都大量借鉴了Tesniere(1959)的著作。最具影响力的依存语法框架包括意义文本理论(MTT) (Melcuk, 1988)、单词语法(Hudson, 1984)、功能生成描述(FDG) (Sgall 等人，1986)。这些框架在许多方面有所不同，包括它们处理形态、句法、语义和语用因素的程度和方式，它们对多层表示的使用，以及用于分类依存关系的一组关系。

戴维·海斯(David Hays)领导的兰德(RAND)公司在机器翻译方面的早期工作首次将依存语法的自动解析引入了计算语言学。这一依存解析的工作与成分解析的工作密切相关，并明确使用语法来指导解析过程。在这个早期阶段之后，依存解析的计算工作在接下来的几十年里仍然是断断续续的。在这一时期，英语依存解析器的显著实现包括连接语法(Sleator 和 Temperley, 1993)、约束语法(Karlsson 等人，1995)和MINIPAR (Lin, 2003)。

依存解析在 20 世纪 90 年代后期随着大型基于依存的树库的出现以及本章中描述的数据驱动方法的出现而重新流行起来。Eisner(1996)基于源于宾州树库的双元语法，开发了一种高效的依存解析动态规划方法。Covington(2001)在当前基于转移的方法基础上，逐字引入了确定性方法。Yamada 和 Matsumoto(2003)、Kudo 和 Matsumoto(2002)引入了位移-减少范式，以及以支持向量机的形式使用监督机器学习来进行依存分析。

Nivre(2003)定义了现代的、确定的、基于转移的依存解析方法。Nivre 和他的同事随后的工作形式化和分析了许多转移系统、训练方法和处理非投射语言的方法的性能(Nivre 和 Scholz 2004, Nivre 2006, Nivre 和 Nilsson 2005, Nivre 等人 2007, Nivre 2007)。

基于图的最大生成树方法是由 McDonald 等人（2005）引入的。

训练和评估依存英语解析器的最早数据来源来自第 12 章中描述的 WSJ 宾州树库 (Marcus 等人，1993)。为使用概率解析而开发的中心词-寻找(head-finding)规则的使用，促进了从基于短语的解析中自动提取依存解析(Xia 和 Palmer, 2001)。

长期运行的布拉格依存树库项目(PDT)(Hajic, 1998)是直接注释具有多层次形态、句法和语义信息的语料库的最重要的努力。目前的 PDT 3.0 包含超过 150 万个符记(Bejcek 等人，2013)。

通用依存(UD) (Nivre 等人，2016b)是一个旨在为跨语言的依存树库注释创建一致框架的项目，其目标是促进世界各地语言的解析器开发。UD 注释方案由几个不同的努力发展而来，包括斯坦福依存 (de Marneffe 等人 2006, de Marneffe 和 Manning 2008, de Marneffe 等人 2014)、谷歌的通用词类标记(Petrov 等人，2012)和 Interset 国际语的形态学的标记集(Zeman, 2008)。在这项努力的支持下，超过 90 种语言的树库已经被注释，并以单一一致的格式提供(Nivre 等人，2016b)。

自然语言学习会议(CoNLL)多年来进行了一系列与依存解析相关的有影响力的共享任务(Buchholz 和 Marsi 2006, Nilsson 等人 2007, Surdeanu 等人 2008, Hajic 等人 2009)。最近的评估侧重于解析器对丰富形态语言(Seddah 等人，2013 年)和非规范语言形式(如社交媒体、文本和口语语言)的鲁棒性(Petrov 和 McDonald, 2012)。Choi 等人(2015)提出了 10 个依存解析器的性能分析，包括一系列指标，以及可靠、健壮的解析器评估工具。

14.9. 练习

(无)