13. 成分解析

One morning I shot an elephant in my pajamas.
How he got into my pajamas I don't know.

一天早晨，我穿着睡衣射杀了一头大象。
我不知道他是怎么进入我的睡衣的。

Groucho Marx, *Animal Crackers*, 1930

句法分析是将句法结构指定给句子的任务。本章重点关注成分结构，即由第 12 章中描述的那种上下文无关的语法分配的结构。在下一章中，我们将介绍依存解析，这是一种可选的解析结构。

可以在诸如语法检查等应用程序中使用解析树:不能解析的句子可能有语法错误(或至少难以阅读)。

解析树可以是语义分析表示的中间阶段(我们将在第 16 章中介绍)，因此在**问答**等应用程序中发挥作用。例如，要回答以下问题：

Which flights to Denver depart before the Seattle flight?

我们将需要知道发问者想要一个飞往丹佛的航班清单，而不是飞往西雅图的航班，并且解析结构(知道 to Denver 修饰 flights，以及 which flights to Denver 是 depart 的主语)可以帮助我们。

我们首先讨论歧义及其存在的问题，然后给出 Cocke-Kasami-Younger(CKY)算法(Kasami 1965，Younger 1967)，这是一种用于语法分析的标准动态规划方法。我们已经看到了其他动态规划算法，例如最小编辑距离(第 2 章)和 Viterbi(第 8 章)。

原始 CKY 算法返回一个句子的一组解析树的有效表示，但没有告诉我们哪一棵解析树是正确的。为此，我们需要使用每种可能成分的分数来增加 CKY。我们将看到如何使用基于神经跨度的解析器来做到这一点。并且，我们将介绍其他方法，例如用于 CCG 解析的超级标记，部分解析方法(用于需要对输入进行表层语法分析就足够的情况)以及用于评估解析器准确性的标准度量标准集。

13.1. 歧义

歧义是句法分析器面临的最严重的问题。第八章介绍了**词类歧义**和**词类消歧**的概念。这里，我们引入了一种新的歧义类型，称为**结构歧义(structural ambiguity)**，用一个新的玩具(toy)语法 $\mathcal{L}_1$ 来说明，如图 13.1 所示，它向上一章的 $\mathcal{L}_0$ 语法添加了一些规则。

Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

图 13-1：$\mathcal{L}_1$ 微型英语语法和词汇

当语法可以为一个句子分配多个解析时，就会出现结构歧义。格劳乔·马克思(Groucho Marx)在《动物

饼干》(Animal Crackers)中扮演船长斯波尔丁(Captain Spaulding)时的那句名言是模棱两可的，因为 in my pajamas 中的短语既可以是以 elephant 为中心词的 NP 短语的一部分，也可以是以 shot 为中心词的动词短语的一部分。图 13.2 用 $\mathcal{L}_1$ 中的规则说明了这两种对格劳乔·马克思路线的分析。

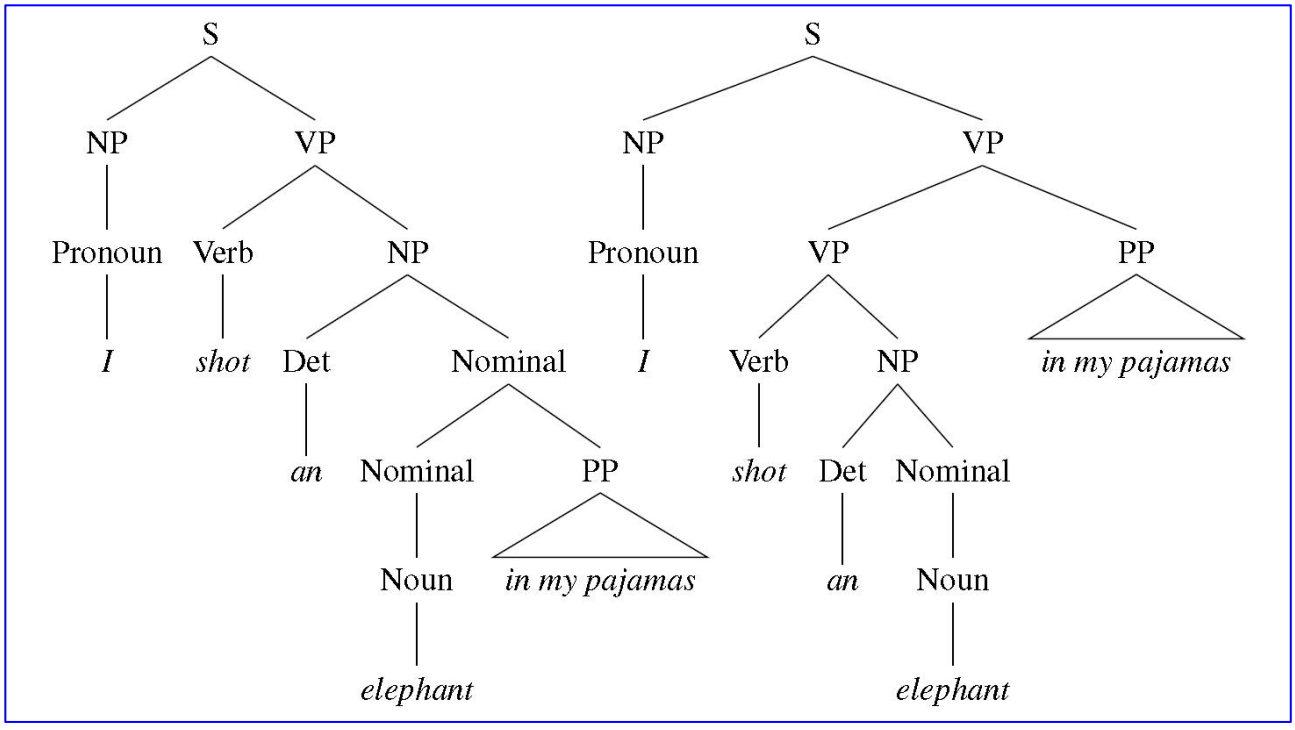


图 13-2：歧义句的两棵解析树

图注：左边的解析对应的是幽默的解读“大象穿着睡衣”，右边的解析对应的是斯波尔丁上尉“穿着睡衣开枪”。

很恰当地说，结构歧义(appropriately enough)具有多种形式。两种常见的歧义是**附着歧义(attachment ambiguity)**和**并列连接歧义(coordination ambiguity)**。如果一个特定的成分可以在一个以上的位置附加到解析树上，则句子具有附着歧义。格劳乔·马克思句子是 PP-附着歧义的一个例子。各种状语短语也受这种歧义的影响。例如，在以下示例中，flying to Paris 的动名词 VP 可以是主语为 the Eiffel Tower 的动名词从句的一部分，也可以是修饰以 saw 为中心词的 VP 的附加语(adjunct)：

(13.1) We saw the Eiffel Tower flying to Paris.

在并列连接歧义中，短语可以通过诸如 and 的连词连接起来。例如，词组 old men and women 可以被括为[old [men and women]]，指的是老男人和老女人，或[old men]和[women]，在这种情况下，只有男人是老的。这些歧义在真实的句子中以复杂的方式结合在一起，比如下面这个来自布朗语料库的新闻句子:

(13.2) President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

这个句子有一些歧义，尽管它们在语义上是不合理的，需要仔细阅读才能看到它们。最后一个名词短语可以被解析为[nationwide [television and radio]]或[[nationwide television] and radio]。“pushed aside”的直接宾语应该是“other White House business”，但也可以是一个奇怪的短语(“other White House business to devote all his time and attention to working”)(这个结构诸如 Kennedy affirmed [his intention to propose a new budget to address the deficit])。那么，这个短语 on the Berlin crisis address he will deliver tomorrow night to the American people 可能是对 pushed 这个动词的附加修饰词。一种类似于 over nationwide television and radio 的 PP 可以连接到任何更高的 VPs 或 NPs(例如，它可以修饰 people 或者 night)。

对于自然出现的句子，有许多语法正确但语义不合理的解析器，这是一个影响所有解析器的令人讨厌的问题。幸运的是，下面的 CKY 算法旨在有效地处理结构歧义。在下一节中我们将看到，我们可以使用神经方法来增强 CKY，通过**句法消歧(syntactic disambiguation)**来选择一个正确的解析。

13.2. CKY 解析：一种动态规划方法

动态规划为解决语法中的歧义问题提供了一个强大的框架。回想一下，动态规划方法系统地填充子问题的解决方案表。完整的表包含解决整个问题所需的所有子问题的解决方案。在语法解析的情况下，这些子问题表示输入中检测到的所有成分的解析树。

动态规划的优势来自于语法规则的上下文无关性质—一旦在输入的一个片段中发现了一个成分，我们就可以记录它的存在，并使它可以在任何可能需要它的后续派生中使用。这提供了时间和存储效率，因为子树可以在表中查找，而不是重新分析。本节介绍 CKY 算法，这是最广泛使用的基于动态规划的解析方法。**图表解析(chart parsing)**(Kaplan 1973, Kay 1982)是一种相关的方法，动态规划方法通常被称为图表解析方法。

13.2.1. CNF 转换

CKY 算法要求语法首先采用乔姆斯基范式(CNF)。回想一下第 12 章，CNF 中的语法被限制为 A→B C 或 A→w 形式的规则。也就是说，每个规则的右边必须扩展为两个非终端符或单个终端符。将语法限制为 CNF 并不会导致表达性的任何损失，因为任何上下文无关的语法都可以转换为相应的 CNF 语法，该语法接受与原始语法完全相同的一组字符串。

让我们从将一个通用的 CFG 转换为用 CNF 表示的 CFG 的过程开始。假设我们正在处理一个ϵ-free 语法，那么在任何通用语法中都需要解决三种情况：①右边混合了终端和非终端的规则；②右边只有一个非终端的规则；③右边长度大于 2 的规则。

①对于混合终端和非终端的规则的补救措施是简单地引入一种新的虚拟非终端，只覆盖原始终端。例如，不定式动词短语 INF-VP→to VP 的规则将被 INF-VP→TO VP 和 TO→to 这两条规则所取代。

②右边有一个单独的非终端符的规则称为**单位产生式(unit productions)**。我们可以消除单位产生式规则，方法是用最终导致的所有非单位产生式规则的右侧，来重写原始规则的右侧。更正式地讲，如果 $A \overset{*}{\Rightarrow} B$ 由一个或多个单位产生式的链组成，并且 $B \rightarrow \gamma$ 是我们语法中的非单位产生式，则我们对语法中的每个这样的规则加 $A \rightarrow \gamma$ 并丢弃所有中间单位产生式。正如我们在玩具语法中所展示的那样，这可能会导致语法的实质性**扁平化(flattening)**，并最终导致生成的树中的终端达到相当高的水平。

③右边大于 2 的规则通过引入新的非终端符来归一化，这些非终端符将较长的序列分散到几个新规则上。正式地说，如果我们有这样的规则

$$A \rightarrow B\ C\ \gamma$$

我们用一个新的非终端符替换了最左边的一对非终端符，并引入了一个新的产生式，结果产生了以下新规则:

$$A \rightarrow X1\ \gamma$$
$$X1 \rightarrow B\ C$$

在右侧较长的情况下，我们简单地迭代这个过程，直到原来触及的(offending)的那个规则被长度为 2 的规则所取代。替换最左边的一对非终端符完全是任意的;任何产生二元规则的系统方案都足够了。

在我们现有的语法中，规则 S→Aux NP VP 将被 S→X1 VP 和 X1→ Aux NP 这两个规则所取代。整个转换过程可以总结如下:

1.将所有符合的规则复制到新语法中。

2.将规则中的终端转换为虚拟非终端。

3.转换单位产生式。

4.将所有规则变为二元规则并将它们添加到新语法中。

$\mathcal{L}_1$ Grammar	$\mathcal{L}_1$ in CNF
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

图 13-3：$\mathcal{L}_1$ 语法及其到 CNF 的转换

图注：请注意，虽然这里没有显示它们，但是所有来自 $\mathcal{L}_1$ 的原始词汇条目也没有变化。

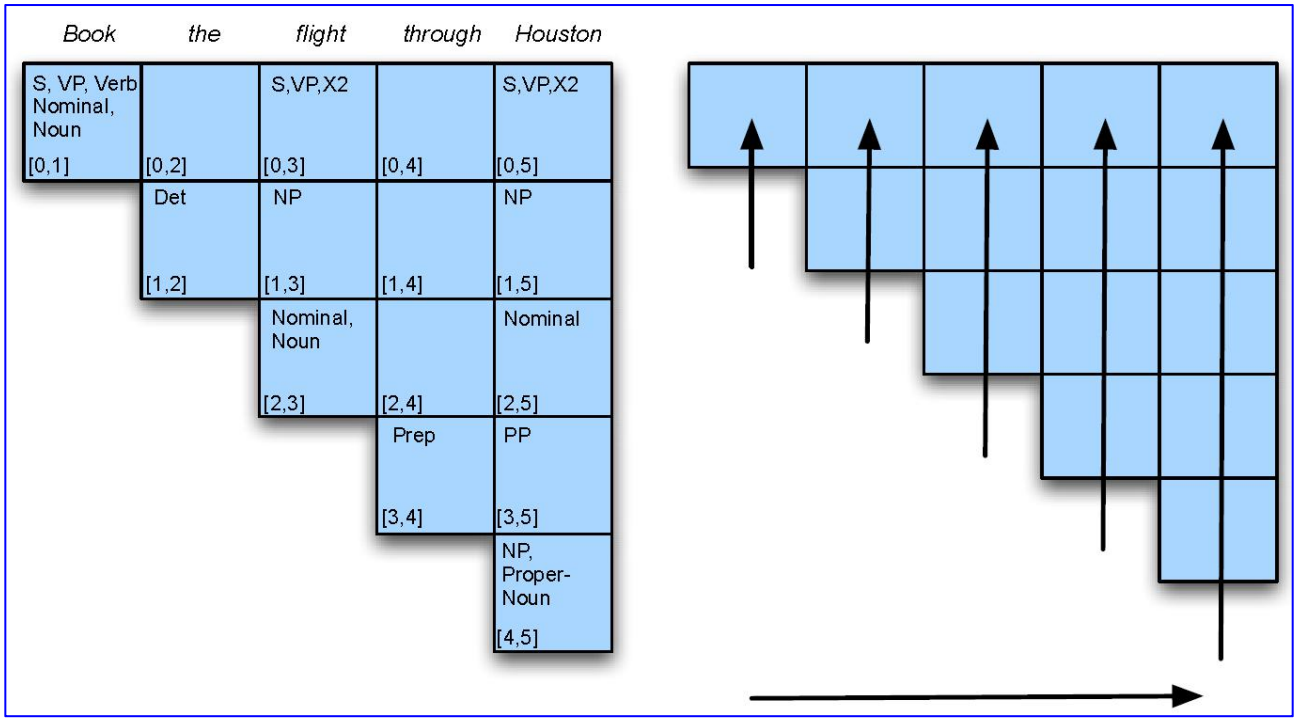


图 13-4：句子 Book the flight through Houston 的完整的解析表

图 13.3 显示了将整个转换过程应用到第 13.1 节介绍的 $\mathcal{L}_1$ 语法的结果。注意，这个图没有显示原始的

词汇规则;由于这些原始的词汇规则已经在 CNF 中，它们都原封不动地保留到新的语法中。然而，图 13.3 显示了各个不同的地方，在这些地方，消除单位产生式的过程，实际上创建了新的词汇规则。例如，在转换后的语法中，所有原来的动词都被提升为 VP 和 S。

13.2.2. CKY 识别

现在我们的语法是在 CNF 中，解析树中词类级别之上的每个非终端节点将有两个子节点。一个二维矩阵可以用来编码整个树的结构。对于长度为 n 的句子，我们将使用(n+1)×(n+1)矩阵的上三角部分。这个矩阵中的每一个单元[i, j]都包含一个非终端点集合，这些非终端点代表了所有成分输入的位置 i 到 j 的成分。由于我们的索引方案从 0 开始，很自然地将索引视为指向输入单词之间的空隙的指针(例如 $_0$ Book $_1$ that $_2$ flight $_3$)。这些间隙通常称为**围栏(fencepost)**，比喻为栅栏段之间的柱子。因此，表示整个输入的单元格位于矩阵中的位置[0,n]。

由于我们表中的每个非终端项在解析中都有两个孩子节点，因此对于由项[i，j]表示的每个成分，输入中必须有一个位置 k，这个 k 可以将[i，j]拆分为两个部分，使得 i<k <j。给定这样的位置 k，第一个成分[i,k]必须沿着第 i 行并处于项[i,j]的左侧某处，而第二个成分[k,j]则应沿第 j 列并位于其下方。为了更加具体，请考虑以下示例及其完整的解析矩阵，如图 13.4 所示。

(13.3) Book the flight through Houston.

矩阵中的超对角线行(superdiagonal row)包含输入中每个单词的词类。该超对角线上方的随后的(subsequent)对角线包含的成分将覆盖输入中所有长度增加的跨度。

```
function CKY-PARSE(words, grammar) returns table

for j ← from 1 to LENGTH(words) do
    for all {A | A → words[j] ∈ grammar}
            table[j − 1, j] ← table[j − 1, j] ∪ A
    for i ← from j − 2 down to 0 do
        for k ← i + 1 to j − 1 do
            for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
                    table[i,j] ← table[i,j] ∪ A
```

图 13-5：CKY 算法

在这种设置下，CKY 识别包括以正确的方式填充解析表。为此，我们将以自底向上的方式进行，以便在填充任何单元格[i, j]时，该单元包含可能有贡献于此条目(即左侧的单元格和下方的单元格)的部分，而且该单元已经被填充了。图 13.5 给出的算法每次填充上三角矩阵一列，从左到右，每列从下到上填充，如图 13.4 右侧所示。该方案保证在每个时间点我们都有所需的所有信息(在左边，因为左边的所有列都已经填满了；在下面，因为我们正在从下到上填满)。它也反映了在线处理，因为从左到右填充列对应着一次处理一个单词。

图 13.5 给出的算法的最外层循环在一个列上迭代，第二个循环在一个行上迭代，剖析从下到上进行。最内层循环的目的是覆盖所有位置，在这些位置上，输入信息中从 i 到 j 的子字符串可能被一分为二。当 k 横跨在可拆分字符串的位置时，我们认为成对的单元格将沿第 i 行向右移动，沿第 j 列向下移动。

图 13.6 说明了填充单元[i，j]的一般情况。在每次这样的拆分时，算法都会考虑是否可以按照语法中的规则认可的方式组合两个单元格的内容。如果存在这样的规则，则将其左侧的非终端符输入到表中。

图 13.7 显示了在读取 Houston 之后，表的第 5 列中的 5 个单元格是如何填充的。箭头指出了用于向表中添加条目的两个跨度(span)。注意，在单元[0,5]这种操作表明，对于这个输入，有三个可选的解析：一个是 PP 修饰 flight；一个是 PP 修饰 booking；一个是捕获到第二个论元(该论元在原始的 VP→Verb NP PP 规则中，但是该论元现在用 VP→X2 PP 规则间接地被捕获了)。

13.2.3. CKY 解析

图 13.5 中给出的算法是识别器，而不是解析器。为了使其成功，只需在单元[0，n]中找到一个 S。要将其转换为能够返回给定输入的所有可能分析的解析器，我们可以对算法进行两个简单的更改：第一个更改是增加表中的条目，以使每个非终端符都与指向其来源的表条目的指针配对(或多或少，如图 13.7 所示)；第二个更改是允许将同一非终端的多个版本输入到表中(再次如图 13.7 所示)。进行了这些更改后，完成的表将包含给定输入的所有可能的解析。返回的任意单个的解析包含从单元格[0，n]中选择的一个 S，然后从表中递归地检索其组成成分。

返回一个句子的每个解析可能没有用，因为解析的数目可能是指数的。在下一节中，我们将介绍如何仅检索最佳解析。

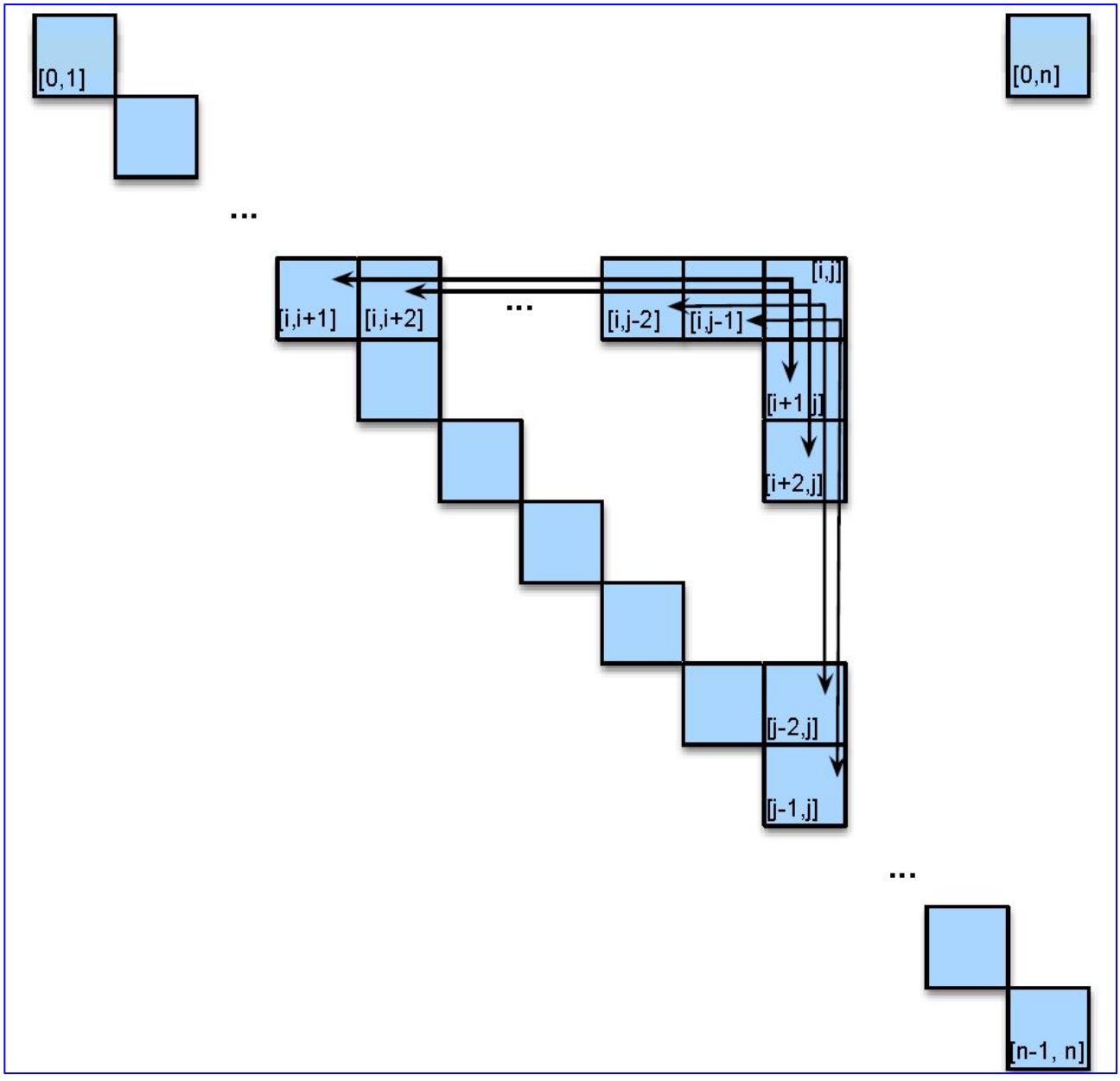


图 13-6：填充 CKY 表中第[i, j]个单元格的所有方法

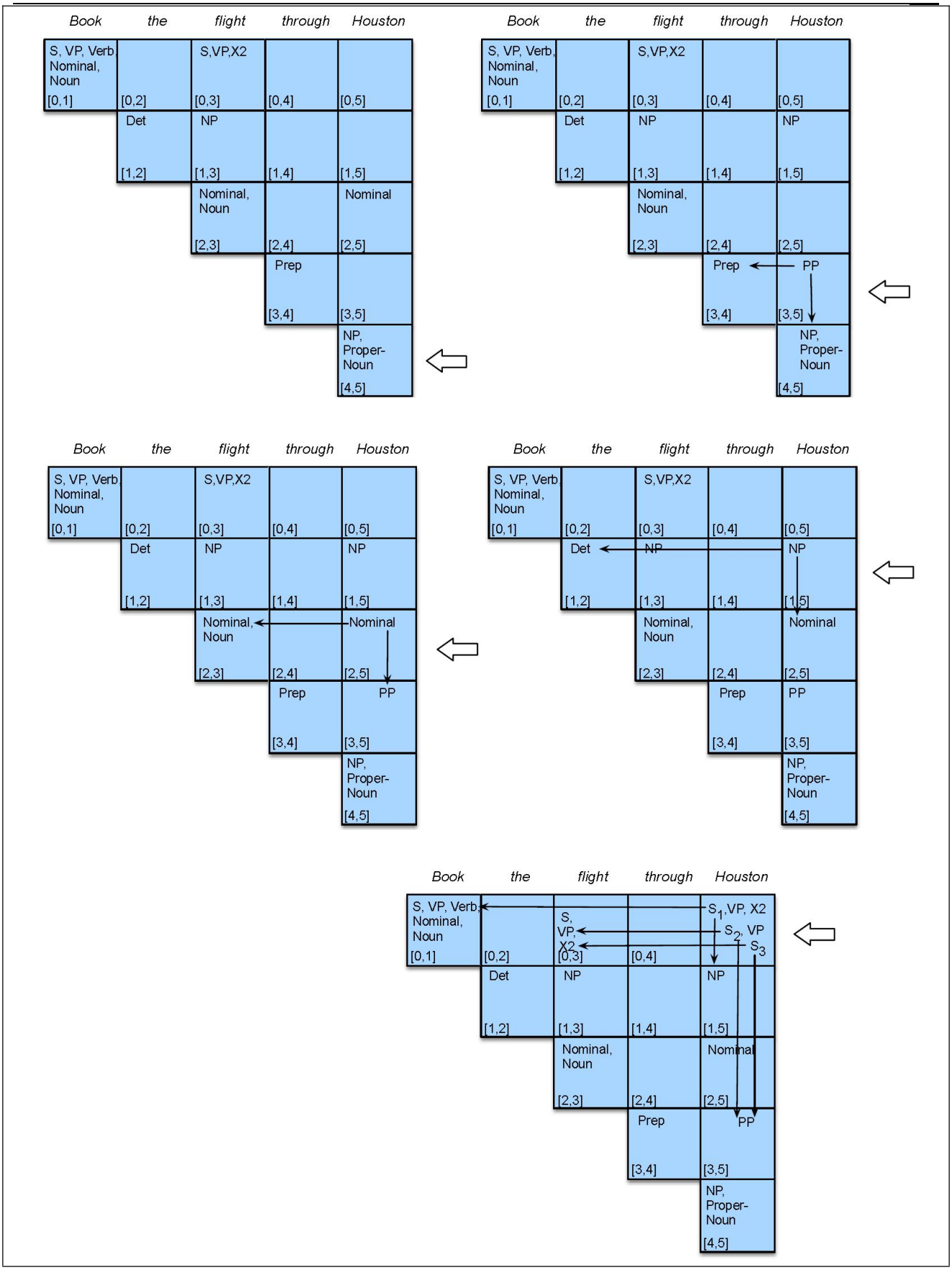


图 13-7：读取 Houston 单词后填充第 5 列的单元格

13.2.4. CKY 实践

最后，我们应该注意到，虽然对 CNF 的限制在理论上并不构成问题，但在实践中却带来了一些重要的问题。显然，就目前的情况来看，我们的解析器返回的树，与友善的句法人员提供给我们的树，在语法上不一致。除了让语法开发人员不高兴之外，转换到 CNF 将使任何句法驱动的语义分析方法复杂化。

解决这些问题的一种方法是保留足够的信息，将树转换回原始语法，作为解析的后处理步骤。对于长度大于 2 的规则所使用的转换来说，这是微不足道的。简单地删除新的伪非终端并提升它们的子节点，就可以恢复原来的树。

对于单位产生式，事实证明，更改基本 CKY 算法以直接处理它们，比存储恢复正确树所需的信息要方便得多。练习 13.3 要求您进行更改。附录 C 介绍的许多概率解析器都使用以这种方式更改的 CKY 算法。

13.3. 基于跨度的神经成分解析

虽然到目前为止我们看到的 CKY 解析算法在枚举一个句子的所有可能解析树方面做得很好，但它有一个很大的问题:它没有告诉我们哪个解析是正确的!也就是说，它不会在可能的解析中消除歧义。为了解决消歧问题，我们将使用 CKY 算法的一个简单的神经扩展。这种解析算法，通常称为**基于跨度的成分解析(span-based constituency parsing)**，或**神经 CKY**，它的直觉是训练神经分类器为每个成分分配分数，然后使用 CKY 的修改版本组合这些成分分数，以找到得分最高的解析树。这里我们将描述 Kitaev 等人(2019)的算法版本。

13.3.1. 计算跨度分数

让我们首先考虑一下位于围栏位置 i 和 j 之间(带有非终端符号标签 l)的成分(我们将其称为**跨度(span)**)。我们将构建一个分类器，为这个成分跨度分配一个分数 s(i,j,l)。图 13.8 概述了该体系结构。

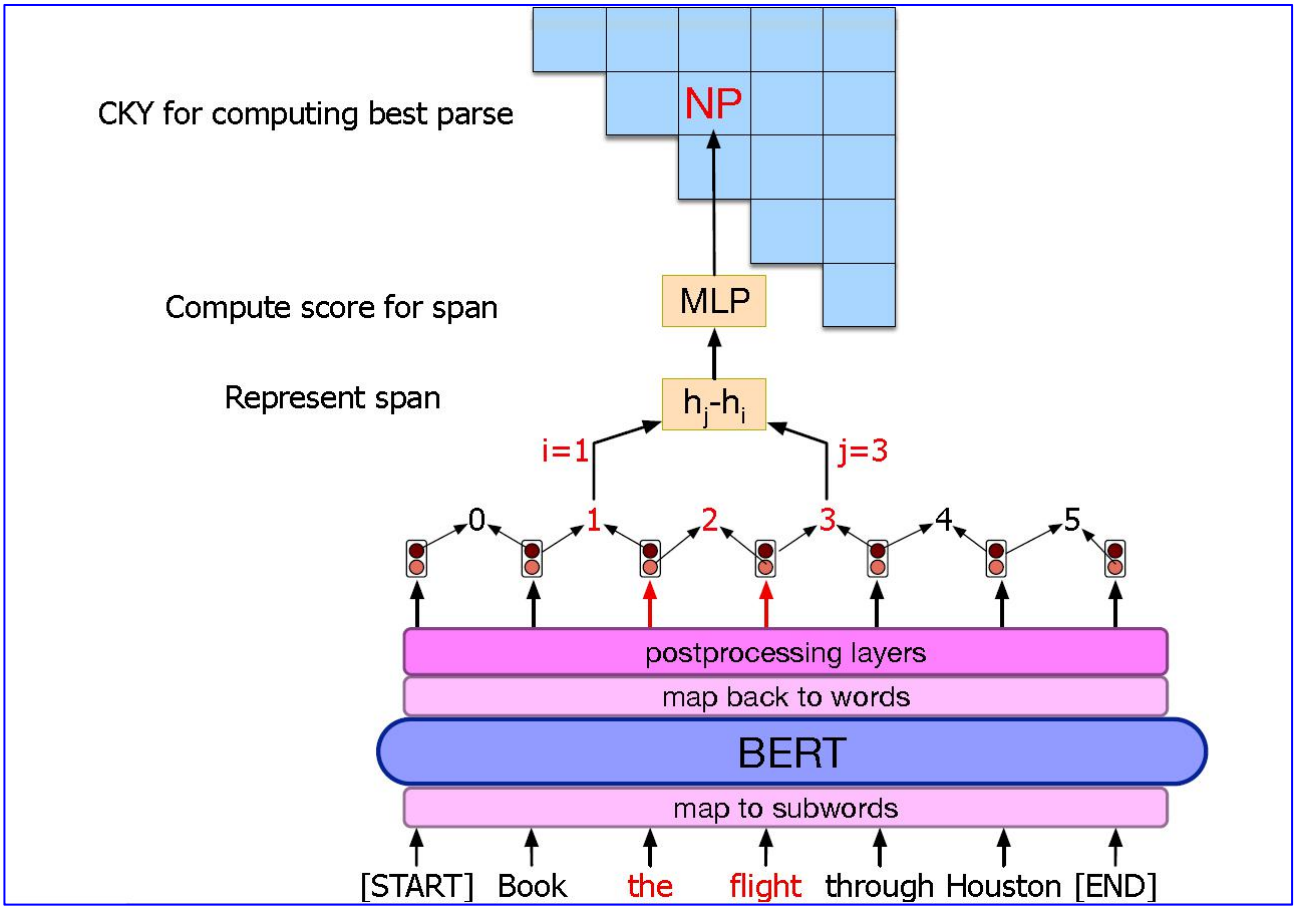
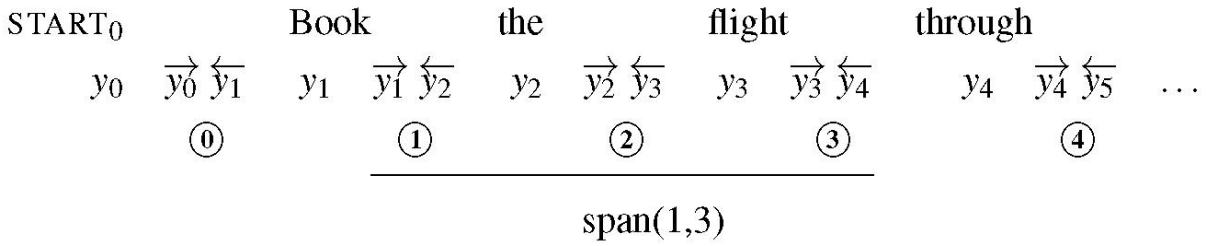


图 13-8：计算“the flight”跨度分数的简化大纲

通过将输入的单词符记传递给像 BERT 这样的预训练语言模型来嵌入它们。由于 BERT 是在子词(词片 wordpiece)符记而不是单词的级别上运行的，因此我们首先需要将 BERT 的输出转换为单词表示形式。一种标准的执行方法是简单地使用最后一个子词单元作为单词的表示形式(使用第一个子词单元似乎效果相当好)。然后可以将嵌入穿过某些后处理层；例如，Kitaev 等人(2019)使用 8 层 Transformer。

然后将所得的单词编码器输出的结果 y_t 用于计算跨度得分。首先，我们必须将单词编码(由单词位置索引)映射到跨度编码(由围栏杆索引)。为此，我们用两个单独的值表示每个围栏杆。直觉是单词右侧的跨度端点表示与单词左侧的跨度端点不同的信息。通过将 y_t 分为两半，我们将每个单词输出 y_t 转换为在此围栏杆结束的跨度值 $\overrightarrow{y}_t$，并为在此围栏杆开始的跨度的转换为跨度值 $\overleftarrow{y}_t$。然后，每一个跨度从一个双向量围栏杆延伸到另一个，如下面的 the flight 表示为跨度(1,3):



一个传统的表示跨度的方法，最初是为基于 RNN 的模型开发的(Wang 和 Chang，2016)，但也扩展到了 Transformers，它采用了开始和结束嵌入之间的差异，即表示跨度(i,j)通过从 j 的嵌入中减去 i 的嵌入。在这里，我们通过拼接每个围栏组件的差异来表示跨度：

$$v(i,j) = [\overrightarrow{y_j} - \overrightarrow{y_i} \; ; \; \overleftarrow{y_{j+1}} - \overleftarrow{y_{i+1}}] \tag{13.4}$$

然后将跨度向量 v 通过一个 MLP 跨度分类器，该分类器有两个全连接层和一个 ReLU 激活函数，其输出维数为可能的非终端标签数:

$$s(i, j, \bullet) = W_2 \, \text{ReLU}(\text{LayerNorm}(W_1 v(i,j))) \tag{13.5}$$

然后 MLP 为每个可能的非终端输出一个分数。

13.3.2. 集成跨度分数到解析中

现在，我们对每个标签的成分跨度 s(i,j,l)都有一个分数。但是我们需要为整个解析树评分。形式上，树 T 表示为|T|这样的标签集合。第 t^{th} 个跨度从位置 i_t 开始，到位置 j_t 结束，标签为 l_t:

$$T = \{(i_t, j_t, l_t) : t = 1, : : : , |T|\} \tag{13.6}$$

因此，一旦我们对每个跨度都有了一个分数，解析器就可以通过对其成分跨度的分数求和来计算整个树 s(T)的分数:

$$s(T) = \sum_{(i,j,l) \in T} s(i,j,l) \tag{13.7}$$

我们可以选择最终的解析树作为得分最高的树:

$$\hat{T} = \underset{T}{\text{argmax}} \, s(T) \tag{13.8}$$

产生最有可能的解析的最简单方法是为每个跨度贪心地选择得分最高的标签。这个贪心的方法不能保证产生一棵树，因为一个跨度的最佳标签可能不适合一个完整的树。然而，在实践中，贪心的方法往往会找到树;Gaddy 等人(2018)在他们的实验中发现，95%的预测包围(predicted bracketings)形成有效的树。

尽管如此，更常见的做法是使用 CKY 算法的变体来找到完整的解析。Gaddy 等人(2018)定义的变量如下。让我们将 $s_{best}(i,j)$定义为最佳子树跨度(i,j)的分数。对于长度为 1 的跨度，我们选择最佳标签：

$$s_{\text{best}}(i, i+1) = \max_{l} s(i, i+1, l) \tag{13.9}$$

对于其他跨度(i, j)，递归为:

$$s_{\text{best}}(i,j) = \max_{l} s(i,j,l) + \max_{k}[s_{\text{best}}(i,k) + s_{\text{best}}(k,j)] \tag{13.10}$$

对更详细的关于基于跨度的句法分析，包括基于边界的训练算法，请参见 Stern 等人(2017)、Gaddy 等人(2018)、Kitaev 和 Klein(2018)、Kitaev 等人(2019)。

13.4. 评估解析器

有一个标准的工具可以用来评估为一个句子分配单个解析树的解析器，这个工具就是**解析值(PARSEVAL)**指标(Black 等人，1991)。PARSEVAL 指标用来衡量两种成分的相像程度，这两种成分是：**假设解析**，手工标签的**参考解析**。因此 PARSEVAL 要求测试集中的每个句子都有一个人工标签的参考(或“黄金标准”)解析，我们通常从诸如宾州树库中提取这些参考解。

如果在参考解析 C_r 中的一个成分具有相同的起点、终点和非终端符，那么在一个句子 s 的假设解析 C_h 中的一个成分将被标记为正确的。然后我们可以测量精度和召回率，就像我们已经看到的任务(如命名实体标记):

$$\textbf{labeled recall: } = \frac{\textbf{\# of correct constituents in hypothesis parse of } s}{\textbf{\# of correct constituents in reference parse of } s}$$

$$\textbf{labeled precision: } = \frac{\textbf{\# of correct constituents in hypothesis parse of } s}{\textbf{\# of total constituents in hypothesis parse of } s}$$

像往常一样，我们经常报道两者的结合，F_1:

$$F_1 = \frac{2PR}{P+R} \tag{13.11}$$

我们还为每个句子 s 使用了一个新的度量，即交叉括号数：

交叉括号数(cross-brackets):参考解析具有括号诸如((A B) C)、而假设解析具有括号诸如(A(B C))的成分的数量。

为了比较使用不同语法的解析器，PARSEVAL 指标包括一个规范化算法，用于删除可能是语法特定的信息(辅助，前不定式“to”等)并计算简化分数(Black 等人，1991)。PARSEVAL 指标的规范实现称为 **evalb**(Sekine 和 Collins，1997)。

13.5. 部分解析

许多语言处理任务不需要所有输入的复杂、完整的解析树。对于这些任务，输入句子的**部分解析(partial parse)**或**浅解析(shallow parse)**可能就足够了。例如，信息提取系统通常不会从文本中提取所有可能的信息:它们只是对文本中可能包含有价值信息的部分进行识别和分类。

部分解析的一种被称为**组块分析(chunking)**。组块是对句子中扁平的、不重叠的片段进行识别和分类的过程，这些片段构成了基本的非递归短语，这些短语对应于主要内容词的词类：名词短语、动词短语、形容词短语和介词短语。在一篇文章中找到所有的基本名词短语的任务是特别常见的。由于组块文本缺乏层次结构，在给定的例子中，一个简单的括号标记法就足以表示组块的位置和类型:

(13.12) [NP The morning flight] [PP from] [NP Denver] [VP has arrived.]

这种括号标记法明确了组块中涉及的两个基本任务:分段(寻找组块的不重叠区域)和标签(为发现的组块分配正确的标记)。有些输入词可能不是任何组块的一部分，特别是在基本 NP 这样的任务中:

(13.13) [NP The morning flight] from [NP Denver] has arrived.

构成语法基本短语的内容取决于应用程序(以及短语是否来自树库)。然而，大多数系统都遵循一些标准准则。首先也是最重要的是，给定类型的基本短语不会递归地包含相同类型的任何成分。消除这种递归留给我们的问题是确定非递归短语的边界。在大多数方法中，基本短语包括短语的标题词，以及成分内的任何中心词之前的材料，而至关重要的是排除任何中心词之后的材料。消除中心词之后的修饰符也就排除

了解决附加歧义的需要。这种排斥确实导致了某些奇怪的现象，比如 PP 和 VP 通常都是由中心词组成的。因此，从 a flight from Indianapolis to Houston 将减少到如下:

(13.14) [NP a flight] [PP from] [NP Indianapolis][PP to][NP Houston]

组块算法通常是通过监督学习来完成的，从带注释的训练数据中训练 BIO 序列标记器，我们在第 8 章中见过。回想一下，在 BIO 标记中，我们有一个标记用于每个组块类型的开始(B)和内部(I)，还有一个符记用于任何组块的外部(O)。下面的例子显示了(13.12)的括号标记法被重新框定为标记任务:

(13.15) *The morning flight from Denver has arrived*
B_NP I_NP I_NP B_PP B_NP B_VP I_VP

带有基本 NP 标记的相同句子说明了 O 标记的作用。

(13.16) *The morning flight from Denver has arrived.*
B_NP I_NP I_NP O B_NP O O

由于注释工作既昂贵又耗时，组块通常依赖于现有的树库，如宾州树库，从句子的完整解析成分中提取语法短语，找到合适的中心词，然后包括中心词左侧的材料，忽略右侧的文本。这有点容易出错，因为它依赖于第 12 章中描述的中心词发现规则的准确度。

给定一个训练集，任何序列模型都可以用来进行组块：CRF、RNN、Transformer 等。与对命名实体标记符的评估一样，组块的评估通过将组块输出与人工注释器提供的黄金标准答案进行比较，使用精度、召回率和 F1。

13.6. CCG 解析

CCG 之类的词汇化语法框架提出了一些问题，而我们一直在讨论的基于短语的方法并不是特别适合。为了快速回顾，CCG 由三个主要部分组成：一组**范畴**，将单词与范畴相关联的**词表**，以及一组用于控制范畴如何在上下文中组合的**规则**。范畴可以是诸如 S 和 NP 之类的原子元素，也可以是诸如(S\NP)/NP 之类的用于指定及物动词范畴的函数。规则指定函数、论元和其他函数的组合方式。例如，以下规则模板(前向和后向函数应用程序)指定函数将函数应用于其论元的方式。

$$X/Y\ Y \Rightarrow X$$
$$Y\ X\backslash Y \Rightarrow X$$

第一个规则将一个函数应用到它右边的论元，而第二个规则从左边查找它的论元。应用其中任何一个规则的结果都是指定为应用函数值的范畴。出于讨论的目的，我们将依赖这两个规则，以及第 12 章中描述的前向和后向组合规则和类型提升。

13.6.1. CCG 中的歧义

在解析中，处理歧义是成功的 CCG 解析的关键。由于大量复杂的词汇范畴与语法规则的普遍性相结合，导致了歧义，从而导致了 CCG 句法分析的困难。为了了解在范畴框架中出现的歧义，我们可以考虑以下例子。

(13.17) United diverted the flight to Reno.

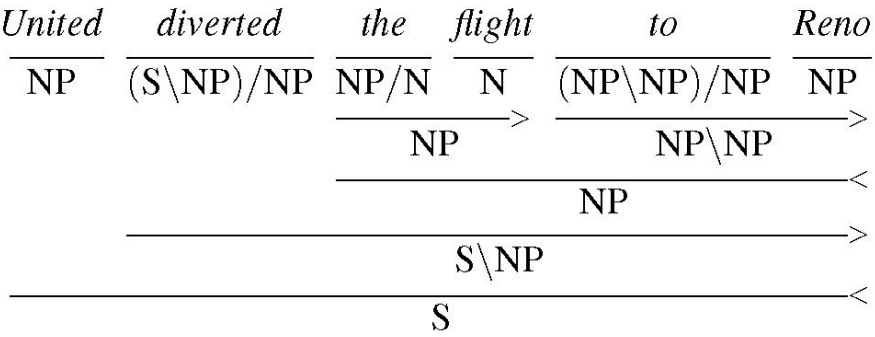
在这个例子中，我们对 the flight 的作用的理解取决于介词短语 to Reno 是作为 the flight 的修饰语，还是作为整个动词短语的修饰语，还是作为动词 divert 的潜在第二个论元。在上下文无关的语法方法中，这种歧义将表现为在语法中的下列规则中进行选择。

$$Nominal \rightarrow Nominal\ PP$$
$$VP \rightarrow VP\ PP$$
$$VP \rightarrow Verb\ NP\ PP$$

在短语结构方法中，我们将简单地将单词 to 分配到类别 P 中，允许它与 Reno 结合形成一个介词短语。随后语法规则的选择将决定最终的推导。在范畴方法中，我们可以将 to 与不同的范畴联系起来，以反映它可能与句子中的其他元素相互作用的方式。然后，相当抽象的组合规则将整理出哪些推导是可能的。因此，

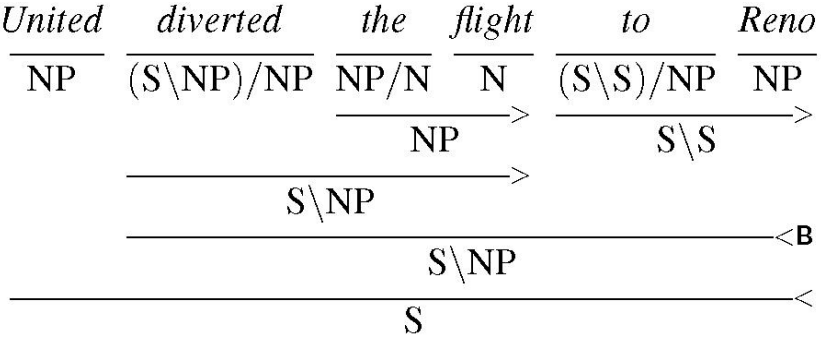
歧义的来源不是来自语法，而是来自词汇。

让我们通过考虑这个例子的几种可能的推导来看看它是如何工作的。为了捕捉介词短语 to Reno 修饰 the flight 的情况，我们将介词 to 指定为范畴(NP\NP)/NP，这就产生了以下推导。

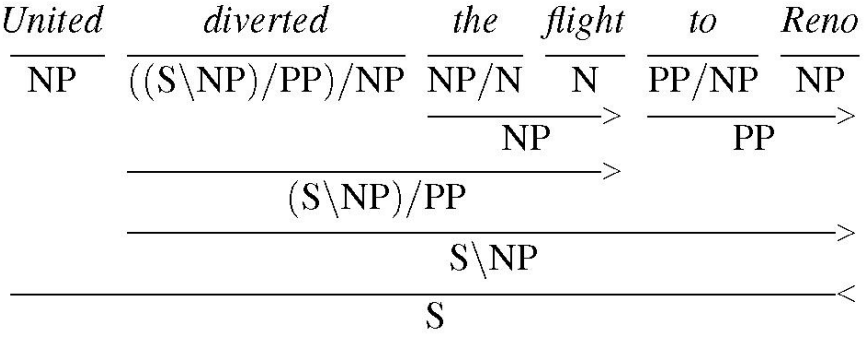


这里，分配给 to 的范畴期望找到两个论元:一个在传统介词的右边，另一个在左边，与要修改的 NP 相对应。

或者，我们可将 to 分配到范畴(S\S)/NP，这允许下面的推导，其中 to Reno 修改了前面的动词短语。



第三种可能是把 divert 看作一个双及物动词，将其归入((S\NP)/PP)/NP 范畴，而把 to Reno 看作一个简单的介词短语。



尽管 CCG 解析器仍然会因语法规则的选择而产生歧义，包括第 12 章中讨论的那种伪歧义，但应该清楚的是，词汇范畴的选择是 CCG 解析中要解决的主要问题。

13.6.2. CCG 解析框架

由于成分语法中的规则要么是二元的，要么是一元的，基于 CKY 算法的自底向上的表格方法应该直接适用于 CCG 解析。不幸的是，每个单词都有大量可用的词汇范畴，再加上 CCG 组合规则的混乱性，导致添加到解析表中的成分数量激增(大多数无用)。管理这种僵尸成分爆炸的关键是准确评估和利用每个单词最可能的词汇范畴——这个过程被称为超级标记。

以下各节描述了使用超级标记的两种 CCG 解析方法。第 13.6.4 节介绍了一种通过使用 A*算法将解析过程构造为启发式搜索的方法。接下来的部分简要介绍了一种更为传统的基于分类器的方法，该方法通过使用自适应超级标记来管理搜索空间的复杂性，该过程会反复考虑越来越多的标记，直到找到解析为止。

13.6.3. 超级标记

第八章介绍了词类标记的任务，即为句子中的每个词指定正确的词汇范畴的过程。**超级标记(supertagging)**是高度词汇化语法框架的相应任务，在这种框架中，指定的标记通常指示一个句子的大部分推导过程。

CCG 超级标记器依赖于树库(如 CCGbank)来提供词汇范畴的整体集合，以及词汇中每个单词的允许范畴分配。CCGbank 包含超过 1000 个词汇范畴，然而，在实践中，大多数超级标记器将其标记集限制为在训练语料库中出现至少 10 次的标记。这使得词典中总共有大约 425 个词汇范畴可供使用。请注意，即使这个较小的数字与宾州树库标记集使用的 45 个 POS 类型相比也很大。

与传统的词类标记一样，构建 CCG 超级标记器的标准方法是使用监督机器学习从手工注释的训练数据构建序列标记器。要找到给定句子中最有可能的标记序列，最常见的方法是使用神经序列模型，RNN 或 Transformer。

但是，也可以使用第 8 章中所述的 CRF 标记模型，并使用类似的功能：当前单词 w_i、其周围 l 个单词内的单词、本地 POS 标记和字符后缀、以及前一时间步长的超级标签，通过最大化训练语料的对数似然性进行训练，并通过第 8 章中所述的 Viterbi 算法进行解码。

不幸的是，大量可能的超级标记与较高的单词歧义相结合，导致朴素的 CRF 算法的错误率对于解析器中的实际应用而言太高了。单个最佳标记序列t̂通常包含太多不正确的标记，无法进行有效的解析。为了克服这个问题，我们取而代之的是在输入中每个单词的可能超级标记上返回概率分布。下表说明了一个简单句子的示例分布，其中每一列代表给定单词“在输入句子的上下文中(in the context of the input sentence)”每个超级标记的概率。“ ...”代表每个单词可能剩余的所有超级标记。

United	serves	Denver
N/N: 0.4	$(S\backslash NP)/NP$: 0.8	NP: 0.9
NP: 0.3	N: 0.1	N/N: 0.05
S/S: 0.1	...	...
$S\backslash S$: .05		
...		

要获得每个可能的单词/标签成对的概率，我们需要将该位置包含该标记的所有超级标记序列的概率相加。这可以通过附录 A 中描述的用于训练 CRF 的前向-后向算法来实现。

13.6.4. 使用 A*算法的 CCG 解析

A*算法是一种启发式搜索方法，利用议程来寻找最优解。代表部分解决方案的搜索状态将基于成本函数添加到议程中，并在每次迭代中选择成本最低的选项进行进一步探索。当代表一个完整解的状态第一次从议程中被选择时，它保证是最优的，搜索结束。

A*成本函数 f(n)用于有效引导搜索到解决方案。f-成本包含两个部分：

g(n)，由状态 n 表示的局部解的精确成本；

h(n)，使用状态 n 的解的启发式近似成本。

当 h(n)满足不会高估实际成本的条件时，A*会找到最优解。毫不奇怪，启发式方法越接近实际成本，A*就越能有效地找到解决方案，而无需探索解决方案空间的大部分。

当应用于解析时，搜索状态对应于代表完成成分的边。每条边都指定一个成分的开始和结束位置、其语法范畴以及其 f-成本。此处，g 分量代表边的当前成本，h 分量代表完成利用该边的推导的成本估算。使用 A*进行短语结构解析源自 Klein 和 Manning(2003)，而此处介绍的 CCG 方法基于 Lewis 和 Steedman(2014)的工作。

使用来自超级标记的信息，一个议程和一个分析表被状态初始化了，这些状态代表了输入中每个单词的所有可能词汇类范畴及其 f-成本。主循环从议程中删除了成本最低的边，并进行测试以查看其是否完整

的推导。如果它反映了一个完整的推导，则将其选择为最佳解决方案，然后循环终止；否则，将根据适用的 CCG 规则生成新状态，分配成本，并将其输入议程以等待进一步处理。循环继续进行，直到发现完整的推导；或议程已结束，表明解析失败。该算法在图 13.9 中给出。

启发式函数

在为 A*搜索定义启发式函数之前，我们需要确定如何评估 CCG 推导的质量。我们将简化假设，即 CCG 推导的概率只是在推导中分配给单词的超级标记的概率的乘积，而忽略了推导中使用的规则。更正式地说，给定包含超级标记序列 T 的句子 S 和推导 D，我们有：

$$P(D,S) = P(T,S) \tag{13.18}$$

$$= \prod_{i=1}^{n} P(t_i|s_i) \tag{13.19}$$

为了更好地适应传统的 A*方法，我们更倾向于使用成本函数给状态评分，越低越好(即，我们试图最小化推导的成本)。为了达到这个目的，我们将使用负对数概率进行推导;这将得到以下方程式，我们将使用该方程式为完整的 CCG 推导打分。

$$P(D,S) = P(T,S) \tag{13.20}$$

$$= \sum_{i=1}^{n} -\log P(t_i|s_i) \tag{13.21}$$

```
function CCG-ASTAR-PARSE(words) returns table or failure

  supertags ← SUPERTAGGER(words)
  for i ← from 1 to LENGTH(words) do
      for all {A | (words[i], A, score) ∈ supertags}
          edge ← MAKEEDGE(i − 1, i, A, score)
          table ← INSERTEDGE(table, edge)
          agenda ← INSERTEDGE(agenda, edge)
  loop do
      if EMPTY?(agenda) return failure
      current ← POP(agenda)
      if COMPLETEDPARSE?(current) return table
      table ← INSERTEDGE(chart, edge)
      for each rule in APPLICABLERULES(edge) do
          successor ← APPLY(rule, edge)
          if successor not ∈ in agenda or chart
              agenda ← INSERTEDGE(agenda, successor)
          else if successor ∈ agenda with higher cost
              agenda ← REPLACEEDGE(agenda, successor)
```

图 13-9：基于 A*算法的 CCG 解析

根据这个模型，我们可以定义 f-成本如下所示。一条边的 f-成本是两个分量的总和：

g(n)，即由这条边表示的跨度的成本；

h(n)，即完成包含这条边的推导所需成本的估计

(这些通常被称为内部成本和外部成本)。

我们将用公式 13.21 定义一条边的 g(n)。也就是说，它只是构成跨度的超级标记成本的总和。

对于 h(n)，我们需要一个接近但从不高估最终推导的实际成本的分数。满足这一要求的一个简单的启发式方法是假定外部跨度内的每个单词都将被分配其最可能的超级标记。如果这些是在最终推导中使用的标记，那么它的分数将等于启发式的分数。如果在最终的推导中使用任何其他的标签，f-成本会更高，因为新的标记必须有更高的成本，从而保证我们不会高估。

综上所述，我们得出以下关于一条边的合适 f-成本的定义。

$$
\begin{aligned}
f(w_{i,j},t_{i,j}) &= g(w_{i,j})+h(w_{i,j}) \qquad (13.22)\\
&= \sum_{k=i}^{j} -\log P(t_k|w_k)+\\
&\quad \sum_{k=1}^{i-1} \min_{t\in tags}(-\log P(t|w_k))+\sum_{k=j+1}^{N} \min_{t\in tags}(-\log P(t|w_k))
\end{aligned}
$$

举个例子，在下面的例子中，有一条边表示带有超级标记 N 的单词 serve。

(13.23) United serves Denver.

这条边的 g-成本就是这个标记的负的对数概率，-$\log_{10}$(0.1)或者是 1。外部的 h-成本由 United 和 Denver 最乐观的超级标记分配组成，分别为 N/N 和 NP。因此，这条边的 f-成本为 1.443。

一个例子

图 13.10 显示了此示例的初始议程和完整解析的进度。使用来自超级标记的信息初始化议程和分析表后，它将从议程中选择最佳的边—United 条目带有标记 N/N 和 f-成本 0.591。该条边不构成完整的解析，因此可通过应用所有相关的语法规则来生成新的状态。在这种情况下，采用前向应用程序于 United：N/N 和 serves：N，会导致创建边 United serves：N [0,2]，1.795 加到议程中。

向前跳过，在第三次迭代中，将代表曼联为丹佛服务的完整推导的一条边 United serves Denver, S[0,3], 0.716 添加到议程中。但是，该算法不会在这一点上终止，因为此边的成本(0.716)并未将其置于议程的顶部。相反，代表 Denver 的边与范畴 NP 被弹出。这导致在议程上增加了另一条边(类型提升 Denver)。只有在弹出并处理此边之后，表示完整推导的早期状态才会上升到议程的顶部，在那里弹出、测试目标并作为解决方案返回。

A *方法的有效性反映在图 13.10 中的状态着色以及最终的解析表中。以蓝色显示的边(包括未明确显示的所有初始词汇范畴分配)反映了搜索空间中从未进入议程顶部的状态，因此，从不对最终表做出任何贡献。这与 PCKY 方法相反，在 PCKY 方法中，解析器针对输入中所有可能的跨度，系统地使用所有可能的成分来填充解析表，并使用无助于最终分析的大量成分填充表。

13.7. 总结

本章介绍成分解析。以下是主要观点的总结:

- **结构歧义(structural ambiguity)**是解析器面临的重大问题。结构歧义的常见来源包括 **PP-附着(PP-attachment)**，**并列连接歧义(coordination ambiguity)** 和 **名词短语括号歧义(noun-phrase bracketing ambiguity)**。
- **动态规划(dynamic programming)**解析算法(例如 CKY)使用局部解析表来有效解析歧义语句。
- **CKY** 将语法形式限制为 Chomsky 范式(CNF)。
- 解析器使用以下三个指标进行评估：**标签召回率(labeled recall)**，**标签精度(labeled precision)**和 **交叉括号数(cross-brackets)**。
- **部分解析(Partial parsing)**和**组块(chunking)**是用于识别文本中浅层句法成分的方法。它们通过在语法注释的数据上训练的序列模型来解决。

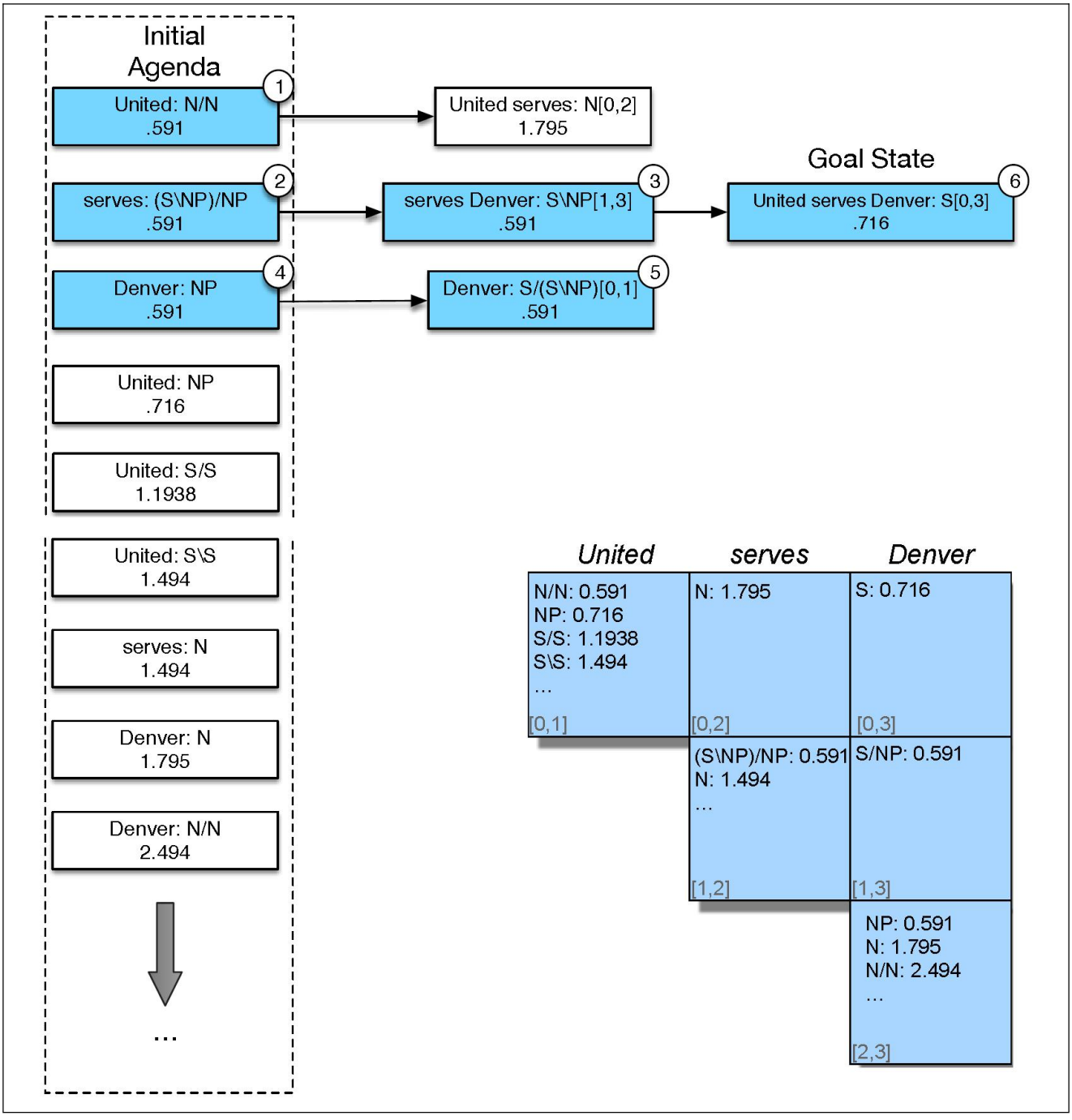


图 13-10：A*搜索“United serves Denver”的示例

图注： 蓝色框上带圆圈的数字表示从议程中弹出状态的顺序。每个状态的成本均基于使用负 log10 概率的 f-成本。

13.8. 文献和历史说明

关于编译器的历史，Knuth 写道:

In this field there has been an unusual amount of parallel discovery of the same technique by people working independently.

(在这一领域中，独立工作的人们对同一技术的并行发现数量之多是不寻常的。)

好吧，也许并不罕见，因为多重发现是科学的常态。但是，肯定有足够多的并行出版物，使这一历史错误地出于简洁的目的，即只对每种算法进行了有特色的早期提及。有兴趣的读者可以参阅 Aho 和 Ullman(1972)。

自底向上解析似乎是由 Yngve(1955)首先描述的，他给出了一个广度优先的自底向上解析算法，作为机器翻译过程说明的一部分。至少 Glennie(1960)、Irons(1961)、Kuno 和 Oettinger(1963)描述了(大概是独立的)自上而下的解析和翻译方法。动态规划解析同样具有独立发现的历史。根据 Martin Kay (个人通讯)的说法，John Cocke 在 1960 年首次实现了一个包含 CKY 算法的根(roots)的动态规划解析器。后来的工作扩展和形式化了算法，并证明了它的时间复杂度(Kay 1967, Younger 1967, Kasami 1965)。相关的格式良好的子字符串表(**WFST**)似乎是由 Kuno(1965)独立提出的，它是一种数据结构，用于存储解析过程中所有以前计算的结果。基于对 Cocke 工作的概括，Kay(1967)(和 Kay 1973)独立地描述了类似的数据结构。Earley 的博士论文(Earley 1968, Earley 1970)描述了动态规划在句法分析中的自顶向下应用。Sheil(1976)证明了 WFST 和 Earley 算法的等价性。Norvig(1991)表明，在任何带有存储化函数的语言(如 LISP)中，只要将存储化操作包装在一个简单的自顶向下解析器周围，就可以捕获动态规划所提供的效率。

虽然在解析的早期历史中，通过有限状态自动机的级联进行解析是很常见的(Harris, 1962)，但不久之后，焦点就转移到了完全 CFG 解析。Church(1980)主张回归有限状态语法作为自然语言理解的处理模型;其他早期的有限状态解析模型包括 Ejerhed(1988)。

解析算法的经典参考文献是 Aho 和 Ullman (1972);虽然这本书的重点是计算机语言，但大多数算法都已应用于自然语言。一本好的编程语言教科书，如 Aho 等人(1986)也很有用。

13.9. 练习

13.1 Implement the algorithm to convert arbitrary context-free grammars to CNF. Apply your program to the L_1 grammar.

13.2 Implement the CKY algorithm and test it with your converted L_1 grammar.

13.3 Rewrite the CKY algorithm given in Fig. 13.5 on page 264 so that it can accept grammars that contain unit productions.

13.4 Discuss the relative advantages and disadvantages of partial versus full parsing.

13.5 Discuss how to augment a parser to deal with input that may be incorrect, for example, containing spelling errors or mistakes arising from automatic speech recognition.

13.6 Implement the PARSEVAL metrics described in Section 13.4. Next, use a parser and a treebank, compare your metrics against a standard implementation. Analyze the errors in your approach.