EISU

E-Institutes of Shanghai Universities

EASTLING

东方语言学

第八辑

《东方语言学》编委会
上海高校比较语言学E－研究院

上海教育出版社

第八辑

EASTLING

东方语言学

《东方语言学》编委会
上海高校比较语言学E－研究院

上海教育出版社

图书在版编目(CIP)数据
东方语言学. 第八辑 / 潘悟云,陆丙甫主编. —上海:
上海教育出版社,2010.12
ISBN 978-7-5444-3168-2

Ⅰ. ①东… Ⅱ. ①潘… ②陆… Ⅲ. ①汉语—语言学—期刊
Ⅳ. ①H1-55

中国版本图书馆CIP数据核字(2011)第015193号

东方语言学
第八辑
《东方语言学》编委会

出版发行 上海世纪出版股份有限公司
上 海 教 育 出 版 社
易文网 www.ewen.cc
地 址 上海永福路 123 号
邮 编 200031
经 销 各地新华书店
印 刷 江苏启东人民印刷有限公司
开 本 787×1092 1/16 印张 11.5 插页 2
版 次 2010 年 12 月第 1 版
印 次 2010 年 12 月第 1 次印刷
书 号 ISBN 978-7-5444-3168-2/H·0177
定 价 26.00 元

目 录

从自然语言处理的角度看二分法

教育部语言文字应用研究所　冯志伟

内容提要 本文从自然语言处理的角度分析了二分法的优点和不足，指出二分法是多分法的一种特殊情况，主张采用多分法。最后讨论了自然语言的计算复杂性。

关键词 二分法　多分法　上下文无关文法

吴道平在《"两分法"浅析》(《东方语言学》3：26-41）中介绍了语言学中二分法的起源及其在生成转换语法发展中的作用和地位，并分析了 Richard S. Kayne 提出严格二分（Binary Branching）的动因和内在逻辑[①]。这篇文章对于我们进一步了解二分法很有帮助。

我是搞自然语言处理的，近年来对于理论语言学的问题思考得不多，吴道平的文章很深奥，很多地方我还没有完全领会。这里，我只想根据我在自然语言处理研究中的感受，谈一谈自己对于二分法的一些不成熟的想法，就教于方家。

在 N．Chomsky 早期的形式语言理论的研究中，他把文法[②]定义为四元组

$$G=(V_N,\ V_T,\ S,\ P)$$

其中，V_N 是非终极符号的集合，这些符号不能处于生成的终点；V_T 是终极符号的集合，这些符号能处于生成的终点；显然，V_N 与 V_T 的并构成了 V，V_N 与 V_T 不相交，因而有

$$V=V_N \cup V_T$$

$$V_N \cap V_T=\phi\ (\phi\text{表示空集})$$

V_N 中的符号用大写拉丁字母表示，V_T 中的符号用小写拉丁字母表示。符号串用希腊字母表示(有时也可以用拉丁字母表中排在较后面的如 ω 之类的小写拉丁字母来表示符号串)。S 是 V_N 中的初始符号。

P 是重写规则，其一般形式为：

$$\varphi \rightarrow \psi\ 。$$

这里，φ 是 V^+ 中的符号串，ψ 是 V 中的符号串，也就是说，$\varphi \neq \phi$[③]。

---

① Kayne, Richard S. (1984) *Connectedness and Binary Branching*, Foris.

② 本文把 grammar 一律翻译成"文法"，这是遵照计算机科学中的习惯用法，"文法"中的"文"指"规则"之意（如"吏乱法曰舞文"中的"文"就是"规则"），而不是指"文章"之意。程序语言也有 grammar，但程序语言是不能"讲"的，因此，计算机科学中一般不把 grammar 翻译为"语法"。

③ V^* 表示由 V 中的符号构成的全部符号串（包括空符号串 ϕ）的集合，V^+ 表示 V^* 中除 ϕ 之外的一切符号串的集合。例如，如果 $V=\{a,\ b\}$，则有

$$V^*=\{\phi,\ a,\ b,\ aa,\ ab,\ ba,\ bb,\ aaa,\text{L L}\},$$

$$V^+=\{a,\ b,\ aa,\ ab,\ ba,\ bb,\ aaa,\text{L L}\}。$$

在文法 $G=\{V_N,\ V_T,\ S,\ P\}$ 中，其重写规则为 $\varphi\to\psi$，并且要求 $\varphi\neq\phi$。为了给文法进行分类，Chomsky 对文法的重写规则加上如下的限制：

限制 1：如果 $\varphi\to\psi$，那么，存在 $A,\ \omega_1,\ \omega_2,\ \omega$，使得 $\varphi=\varphi_1A\varphi_2$，$\psi=\ \varphi_1\omega\varphi_2$。

限制 2：如果 $\varphi\to\psi$，那么，存在 $A,\ \omega_1,\ \omega_2,\ \omega$，使得 $\varphi=\varphi_1A\varphi_2$，$\psi=\ \varphi_1\omega\varphi_2$，并且 $A\to\omega$。

限制 3：如果 $\varphi\to\psi$，那么，存在 $A,\ \omega_1,\ \omega_2,\ \omega,\ a,\ Q$，使得 $\varphi=\varphi_1A\varphi_2$，$\psi=\varphi_1\omega\varphi_2$，$A\to\omega$，并且 $\omega=aQ$，$\omega\to a$，因而 $A\to aQ$，$A\to a$。

限制 1 要求文法的重写规则全都具有形式 $\varphi_1A\varphi_2\to\varphi_1\omega\varphi_2$，这样的重写规则在上下文 $\varphi_1-\varphi_2$ 中给出 $A\to\omega$。显然，在这种情况下，ψ 这个符号串的长度(即 ψ 中的符号数)至少等于或者大于 φ 这个符号串的长度(即 φ 中的符号数)，如果用 $|\psi|$ 和 $|\varphi|$ 分别表示符号串 ψ 和 φ 的长度，则有 $|\psi|\geqslant|\varphi|$。由于在重写规则 $\varphi_1A\varphi_2\to\varphi_1\omega\varphi_2$ 中，每当 A 出现于上下文 $\varphi_1-\varphi_2$ 中的时候，可以用 ω 来替换 A，因此，把加上了限制 1 的文法叫做上下文有关文法[①](context-sensitive grammar)或 1 型文法(type 1 grammar)。

限制 2 要求文法的重写规则全都具有形式 $A\to\omega$，这时上下文 $\varphi_1-\varphi_2$ 是空的，在运用重写规则时不依赖于单个的非终极符号 A 所出现的上下文，因此，把加上了限制 2 的文法叫做上下文无关文法[②](context-free grammar)或 2 型文法(type 2 grammar)。

限制 3 要求文法的重写规则全都具有形式 $A\to aQ$ 或 $A\to a$，其中，A 和 Q 是非终极符号，a 是终极符号，这种文法叫做有限状态文法(finite state grammar)或 3 型文法(type 3 grammar)，有时也可叫做正则文法(regular grammar)。

没有上述限制的文法，叫做 0 型文法(type 0 grammar)。

显而易见，每一个有限状态文法都是上下文无关的，每一个上下文无关文法都是上下文有关的，每一个上下文有关文法都是 0 型的。这样，Chomsky 把由 0 型文法生成的语言叫 0 型语言(type 0 language)，把由上下文有关文法、上下文无关文法、有限状态文法生成的语言分别叫做上下文有关语言(context-sensitive language)、上下文无关语言(context-free language)、有限状态语言(finite state language)，也可以分别叫做 1 型语言(type 1 language)、2 型语言(type 2 language)、3 型语言(type 3 language)。由于从限制 1 到限制 3 的限制条件是逐渐增加的，因此，不论对于文法或对于语言来说，都有

$$0\text{ 型}\supseteq 1\text{ 型}\supseteq 2\text{ 型}\supseteq 3\text{ 型}。$$

这种包含关系叫做“Chomsky 层级”（Chomsky hierarchy）。

例如，例如，有文法 $G=\{V_N,\ V_T,\ S,\ P\}$

$$V_N=\{S,\ A,\ B,\ C\}$$

$$V_T=\{a,\ b,\ c\}$$

$$S=\{S\}$$

① 有的学者把“上下文有关文法”称为“上下文敏感文法”。

② 有的学者把“上下文无关文法”称为“上下文自由文法”。

P：

$S \to ABC$ (i)
$A \to aA$ (ii)
$A \to a$ (iii)
$B \to Bb$ (iv)
$B \to b$ (v)
$BC \to Bcc$ (vi)
$ab \to ba$ (vii)

这个文法可生成终极符号串 $b^n a^m cc\,(n \geqslant 1,\ m \geqslant 1)$。

不难看出，规则(vii)是0型规则，因此，这个文法是0型文法。如果去掉规则(vii)，那么，就得到一个1型文法，因为规则(vi) $BC \to Bcc$ 是1型规则。如果去掉规则(vii)和(vi)，就得到一个2型文法，因为规则(i) $S \to ABC$ 及规则(iv) $B \to Bb$ 是2型规则。如果去掉规则(vii)、(vi)、(i)、(iv)，就得到一个3型文法，因为剩下的规则(ii) $A \to aA$，规则(iii) $A \to a$，规则(v) $B \to b$ 都是3型规则。

可见，任何的3型文法，一定包含在2型、1型、0型文法中，任何的2型文法，一定包含在1型、0型文法中，任何的1型文法，一定包含在0型文法中。

Chomsky还分析了上述这些文法的长处和不足。

他指出，有一些由非常简单的符号串构成的形式语言，不能由有限状态文法生成，它们是：

（i）ab，$aabb$，$aaabbb$，……，它的全部句子都是由若干个 a 后面跟着同样数目的 b 组成的。这种形式的语言可表示为 $L_1 = \{a^n b^n\}$，其中，$n \geqslant 1$。

（ii）aa，bb，$abba$，$baab$，$aabbaa$，$abbbba$，……，这种形式语言是镜象结构语言，如果用 α 表示集合 $\{a,\ b\}$ 上的任意非空符号串，用 α^* 表示 α 的镜象，那么，这种语言可表示为 $L_2 = \{\alpha\alpha^*\}$。

（iv）aa，bb，$abab$，$aaaa$，$bbbb$，$aabaab$，$abbabb$，……，它的全部句子是由若干个 a 或者若干个 b 构成的符号串 α 后面跟着而且仅只跟着完全相同的符号串 α 而组成的。如果用 α 表示集合 $\{a,\ b\}$ 上的任意非空符号串，那么，这种语言可表示为 $L_2 = \{\alpha\alpha\}$。

L_1，L_2，L_3 都不能由有限状态文法生成，可见，有限状态文法的生成能力不强。

Chomsky认为，上下文无关文法比较适合于描述自然语言。为此，他提出了“Chomsky范式”（Chomsky Normal Form）。

Chomsky证明了，任何的上下文无关语言，均可由重写规则为

$A \to BC$

或 $A \to a$

的文法生成。其中，A，B，$C \in V_N$，$a \in V_T$。

从Chomsky范式的重写规则的形式 $A \to BC$ 不难看出，Chomsky范式是严格实行“二分法”的；$A \to a$ 不过是当规则左部的两个非终极符号BC蜕化成一个单独的终极符号a时的一种特殊情况，从实质上说也是二分的。

利用Chomsky范式，可把任何的上下文无关文法的推导树简化为二元形式。

例如，上下文无关语言 $\{a^n cb^{2n}\}$ 的文法的重写规则为

$S \to aCbb$

$C \to aCbb$

$C \to c$

如果要生成符号串 *aacbbbb*，其推导树为：

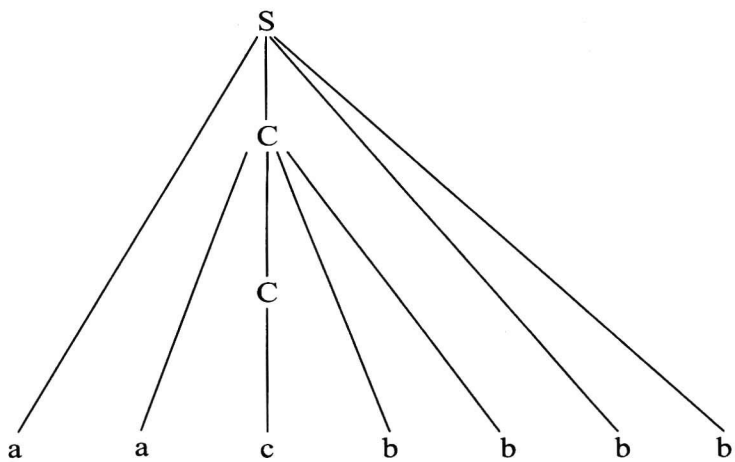

图 1 生成符号串 *aacbbbb* 的推导树

现在我们把这个文法的三个重写规则改写为 Chomsky 范式。

在这三个重写规则中，$C \to c$ 是符合 Chomsky 范式要求的，不必再变换。我们先把 $S \to aCbb$ 及 $C \to aCbb$ 的右边换为非终极符号，用 $S \to ACBB$ 及 $A \to a$，$B \to b$ 替换 $S \to aCbb$，用 $C \to ACBB$ 及 $A \to a$，$B \to b$ 替换 $C \to aCbb$。然后，再把 $S \to ACBB$，$C \to ACBB$ 的右边换成二元形式，用 $S \to DE$，$D \to AC$ 及 $E \to BB$ 替换 $S \to ACBB$，用 $C \to DE$，$D \to AC$ 及 $E \to BB$ 替换 $C \to ACBB$。这样，便得到了符合 Chomsky 范式要求的文法的重写规则：

$S \to DE$

$D \to AC$

$E \to BB$

$C \to DE$

$A \to a$

$B \to b$

$C \to c$

用 Chomsky 范式，可将符号串 *aacbbbb* 的推导树简化为二元形式，这种二元形式的推导树叫“二叉树”（binary tree），如下图所示：

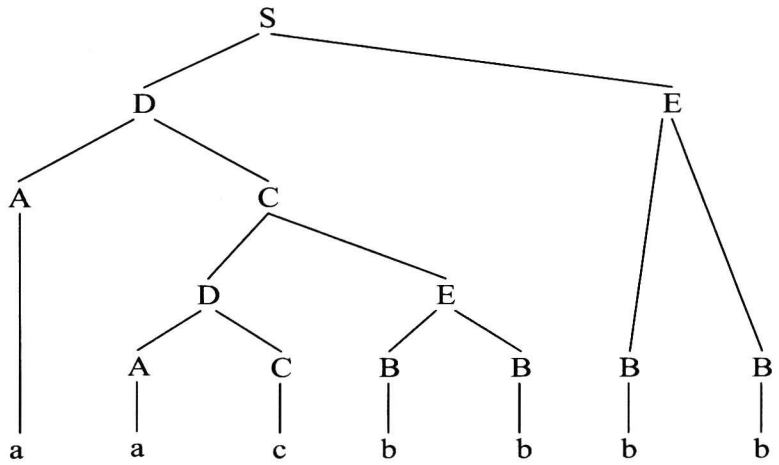

图 2 二叉树

在 Chomsky 范式中，重写规则及推导树都有二元形式，这就为自然语言的形式描写提供了数学模型。

自然语言中的句法结构一般都是二分的，因而一般都具有二元形式（binary form）。例如，在汉语中，除了联合结构、双宾语结构和递系结构(即“兼语式”)之外，具有二元形式的句法结构占大多数：

述宾结构：　思考 问题

主谓结构：　他 走了

偏正结构：　汉语 语法

动补结构：　洗 干净

事实上，语言学中正是采用二分法来分析句子的。二分法就是所谓的层次分析法，这就是说，一个复杂的语言形式，不能一下子就把它分析为若干个词，而要按下面的步骤分析：

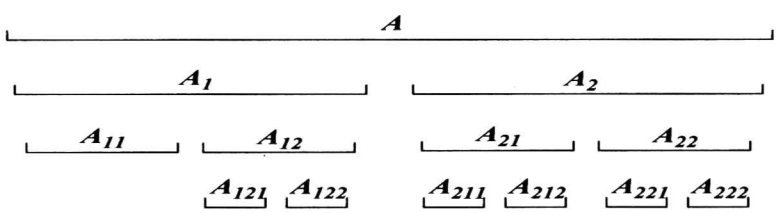

图 3 层次分析法示意图

我们不是把 A 一下子就分成 A_{11}，A_{121}，A_{122}，A_{211}，A_{212}，A_{221}，A_{222}，而是先把 A 分成 A_1 和 A_2 两部分，然后把 A_1 分成 A_{11} 和 A_{12} 两部分，把 A_2 分成 A_{21} 和 A_{22} 两部分，A_{12} 又可再分为 A_{121}，A_{122} 两部分，……，这样分析下去，一直分析到词为止。人们通常把 A_1 和 A_2 叫做 A 的直接成分，把 A_{11} 和 A_{12} 叫做 A_1 的直接成分，把 A_{121}，A_{122} 叫做 A_{12} 的直接成分。这种顺次找出语言格式的直接成分的方法，叫做直接成分分析法或层次分析法。因此，在语言学界，又有人把 Chomsky 的上下文无关短语结构语言模型叫做直接组成成分模型，把用直接组成成分模型分析语言的方法叫做直接成分分析法（immediate constituent analysis，简称 IC 分析法）。

可见，Chomsky 范式反映了自然语言结构的这种二分特性，因而通过 Chomsky 范式这一重要工具，上下文无关文法可以在自然语言研究中得到广泛的应用。

不少语言学家在他们描写自然语言的工作中，已经认识到了自然语言结构的这种二分特性。

吴道平在他的文章中举出了 Wilhelm Wundt 等人使用二分法描述自然语言的例子。我们这里再补充一下面的材料。

——我国语言学家马建忠在《马氏文通》中提出了两端两语说，指出："盖意非两端不明，而句非两语不成。"

——美国语言学家 E. A. Nida(奈达)在 1949 年出版的《形态学》中指出："根据经验，我们发现语言结构倾向于二分。"①

——美国语言学家 C. C. Fries(福里斯)在《英语结构》一书中，更是明确地提出了二分的观点，他指出："在英语里，一个结构层次通常只有两个成分。当然，每一个成分都可以由好几个单位组成，不过在同一层次上，结构的直接成分通常只有两个。"②

Fries 还把这一观点具体地应用于英语句子的分析中。例如，他对"The recommending committee approved his promotion"("推荐委员会批准了他的提升")这个句子的分析是：

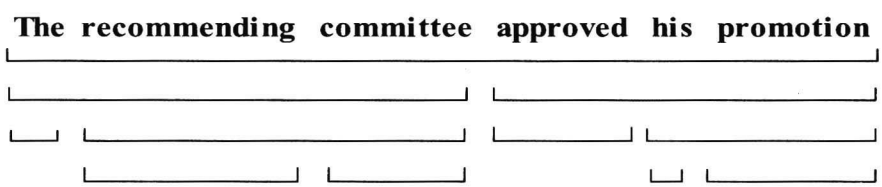

图 4　英语句子的层次分析图示

完全采用了二分法。

在自然语言处理中，二分法也得到了广泛的应用。CYK 算法（CYK algorithm）就是严格遵循二分法的。

CYK算法是Cocke-Younger-Kasami算法的首字母缩写。这是一种并行的句法分析算法。这种算法是以Chomsky范式(Chomsky normal form)为描述对象的句法分析算法。

例如，如果我们有如下的上下文无关文法的规则：

① E. A. Nida，《Morphology》，University of Michigan Press, 1949，P. 91～93.

② C. C. Fries，《英语结构》(中译本)，264 页，商务印书馆，英文本原名《English Structures》.

S → NP VP
VP → V NP
NP → Det N

这些规则都是二分的，满足 Chomsky 范式的要求。

根据这种满足 Chomsky 范式要求的上下文无关文法，对于英语句子“the boy hits a dog”（那个男孩儿打狗），使用 CYK 分析法，我们可以得到图 5 中的表：

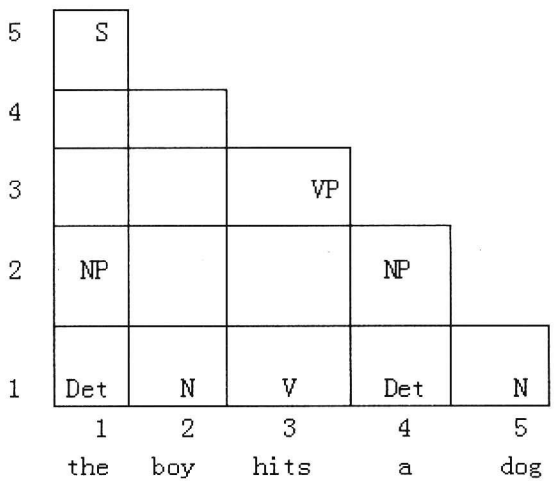

图 5 CYK 算法中的表

在这个表中，行方向（横向）的数字表示单词在句子中的位置，列方向（纵向）的数字表示该语言成分所包含的单词数。语言成分都装在框子(box)内，我们用 b_{ij} 来表示处于第 i 列第 j 行的框子的位置。这样，每一个语言成分的位置就可以确定下来。例如，

$Det \in b_{11}$ 表示 Det 处于第 1 列第 1 行，
$N \in b_{21}$ 表示 N 处于第 2 列第 1 行，
$V \in b_{31}$ 表示 V 处于第 3 列第 1 行，
$Det \in b_{41}$ 表示 Det 处于第 4 列第 1 行，
$N \in b_{51}$ 表示 N 处于第 5 列第 1 行

这样一来，处于第 1 列第 2 行的 NP 的位置可用 b_{12} 表示（$NP \in b_{12}$），这种记法说明，这个 NP 处于句首，包含 2 个单词(the 和 boy)，也就是说，这个 NP 是由 Det 和 N 组成的；处于第 4 列第 2 行的 NP 的位置可用 b_{42} 表示($NP \in b_{42}$)，这种记法说明，这个 NP 处于第 4 个词的位置，包含 2 个单词（a 和 dog），也就是说，这个 NP 是由 det 和 N 组成的；处于第 3 列第 3 行的 VP 的位置可用 b_{33} 表示($VP \in b_{33}$)，这种记法说明，这个 VP 处于第 3 个词的位置，包含 3 个单词（hits, a 和 dog），也就是说，这个 VP 是由 V（包含 1 个词）和 NP（包含 2 个词）组成的；处于第 1 列第 5 行的 S 的位置可用 b_{15} 表示($S \in b_{15}$)，这种记法说

明，这个 S 处于句首，包含 5 个单词(the, boy，hits, a 和 dog)，也就是说，这个 S 是由 NP（包含 2 个单词）和 VP（包含 3 个单词）组成的。这些框子里的标记，明确地说明了这个句子中的句法结构关系，因此，如果我们能够通过有限步骤造出这样的表，就等于完成了句子的句法结构分析。

由于文法规则都用 Chomsky 范式表示，因此，在文法规则 A→BC 中，对于某个 k(1≤k<j) 来说，如果 $b_{i\,k}$ 中包含 B，$b_{i+k\,j-k}$ 中包含 C，则 $b_{i\,j}$ 中必定包含 A。也就是说，如果从输入句子中的第 i 个单词开始，造成了表示由 k 个单词组成的成分 B 的子树（这时，B 的长度为 k,其首词标号为第 i 列，末词标号第 i+k-1 列，例如，如果 B 的长度为 4，如首词标号为 3，则末词标号为 i+k-1=3+4-1=6，即这 4 个词的标号分别为 3，4，5，6），从第 i+k 个单词开始，造成了表示由 j-k 个单词组成的成分 C 的子树（这时，C 的长度为 j-k，其首词标号为第 i+k 列，末词标号为第 i+j-1 列，例如，如果 A 的长度 j=6，C 的长度为 j-k=6-4=2，则其首词标号为 i+k=3+4=7,末词标号为 i+j-1=3+6-1=8），那么，就可以作出如下的表示 A 的树形图（图 6）：

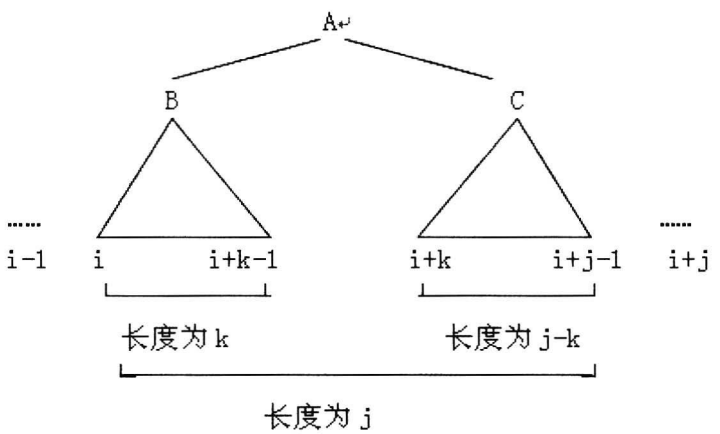

图 6 CYK 算法中的标号

例如，在上表的 $b_{1\,2}$ 中包含 NP，$b_{1\,1}$ 中包含 Det，$b_{2\,1}$ 中包含 N，这反映了文法规则 NP→Det N 的情况。这时，k=1, i=1, j=2。

CYK 算法就是顺次构造上述表的算法，当输入句子的长度为 n 时，CYK 算法可分为如下两步。

第一步：从 i=1 开始，对于长度为 n 的输入句子中的每一个单词 W_i，显然都有重写规则 A→W_i，因此，顺次给每一个单词 W_i 相应的非终极符号 A 记入框子 $b_{i\,1}$ 中。在我们的例句“the boy hits a dog”中，根据相应的重写规则，顺次把 Det 记入 $b_{1\,1}$ 中，把 N 记入 $b_{2\,1}$ 中，把 V 记入 $b_{3\,1}$ 中，把 Det 记入 $b_{4\,1}$ 中，把 N 记入 $b_{5\,1}$ 中。

第一步相当于确定输入句子中各个单词所属的词类，如果一个单词属于若干个词类，可以把它所属的词类都记入表中。

第二步：对于 1≤h<j 以及所有的 i，造出 $b_{i\,h}$，这时，包含 $B_{i\,j}$ 的非终极符号的集合定义

如下：

$b_{i\,j}$ = {A|对于 $1\le k<j$，B 包含在 $b_{i\,k}$ 中，C 包含在 $b_{i+k\ j-k}$ 中，并且，存在文法规则 A→BC}。

第二步相当于构造句子的句法结构。根据文法的重写规则，从句首开始，顺次由 1 到 n 取词构造框子 $b_{i\,j}$，如果框子 $b_{1\,n}$ 中包含开始符号 S，也就是说，$S\in b_{1\,n}$，那么，就说明输入句子是可以接受的。

例如，根据规则 NP→Det N 以及 $det\in b_{1\,1}$ 和 $N\in b_{21}$，可知此时 i=1,k=1,j=2,因此，NP 的框子的编号应为 $b_{1\,2}$；根据规则 NP→Det N 以及 $Det\in b_{4\,1}$ 和 $N\in b_{5\,1}$，可知此时 i=4,k=1,j=2,因此，这个 NP 的框子的编号应为 $b_{4\,2}$；根据规则 VP→V NP 以及 $V\in b_{3\,1}$ 和 $NP\in b_{4\,2}$,可知此时 i=3,k=1,j=3,因此，VP 的框子的编号应为 $b_{3\,3}$；根据规则 S→NP VP 以及 $NP\in b_{1\,2}$ 和 $VP\in b_{3\,3}$，可知此时 I=1,k=2,j=5,因此，S 的框子的编号 $b_{5\,1}$。由于句子长度 n=5，因此，有 $S\in b_{n\,1}$，所以输入句子被接受，分析成功。

由于基于Chomsky范式的CYK算法严格遵循了二分法，算法简单明快，效率较高。在自然语言处理中强制性地使用二分法，采用二叉树来描述自然语言的层次结构和线性顺序，可以提高自然语言处理系统的性能。基于Chomsky范式的二分法，在自然语言处理中受到了欢迎。

然而，自然语言处理的实践表明，这种二分法还是有缺陷的。我们认为，主要的缺陷有如下三方面：

第一，上下文无关文法的规则是 $A\rightarrow\omega$，其中规则右部的 ω 可以是终极符号，也可以是非终极符号，还可以是由终极符号和非终极符号混合组成的符号串，所以，ω 不一定总是二分的。如果上下文无关文法的规则不是严格二分的，那么，在 CYK 算法中，就必须首先把这些规则都转写成严格二分的 Chomsky 范式，才有可能用 CYK 算法进行自动分析。这样，在很多情况下，我们需要对于规则进行转换，CYK 算法使用起来就不是很方便，自动分析的效率也会降低。

例如，如果上下文无关语法具有如下的规则：

S → NP VP

NP → PrN

NP → DET N

NP → N WH VP

NP → DET N WH VP

VP → V

VP → V NP

VP → V that S

我们用这个语法来剖析句子‘the table that lacks a leg hits Jack”。

由于这些规则不全是二分的，首先我们需要把重写规则转换为 Chomsky 范式。这样，就需要做如下的转换工作，逐一地审查每一条规则是不是二分的：

S → NP VP 这个规则是二分的，是 Chomsky 范式，不需要转换。

NP → PrN 这个规则不是 Chomsky 范式，因此转换为如下的 Chomsky 范式：

NP → Jack

NP → John

NP → Maria

NP → DET N 这个规则是二分的，是 Chomsky 范式，不需要转换。

NP → N WH VP 这个规则不是 Chomsky 范式，因此转换为如下的 Chomsky 范式：

NP → N CL

CL → WH VP

NP → DET N WH VP 这个规则不是 Chomsky 范式，因此转换为如下的 Chomsky 范式：

NP → NP CL

NP → DET N

CL → WH VP

这里 CL 是一个 WH 从句（WH clause），它由 that 和 VP 组成。

VP → V 这个规则不是 Chomsky 范式，因此转换为如下的 Chomsky 范式：

VP → cough

VP → walk

VP → …

VP → V NP 这个规则是二分的，是 Chomsky 范式，不需要转换。

VP → V that S 这个规则不是 Chomsky 范式，因此转换为如下的 Chomsky 范式：

VP → V TH

TH → WH S

这里 TH 是一个 that 从句，它由 that 和 S 组成。

我们必须把全部规则转换成 Chomsky 范式之后，才能够根据 CYK 算法来计算非终极符号 b_{ij} 的列号和行号。例如，我们可以按照句子中的词序排列表示词类（POS）的非终极符号 b_{ij}，逐一地计算它们的列号和行号：

“The	table	that	lacks	a	leg	hits	Jack”
DET	N	WH	V	DET	N	V	NP
b_{11}	b_{21}	b_{31}	b_{41}	b_{51}	b_{61}	b_{71}	b_{81}

计算出表示短语的非终极符号 b_{ij} 的列号和行号之后，为我们得到如下的方框和表：

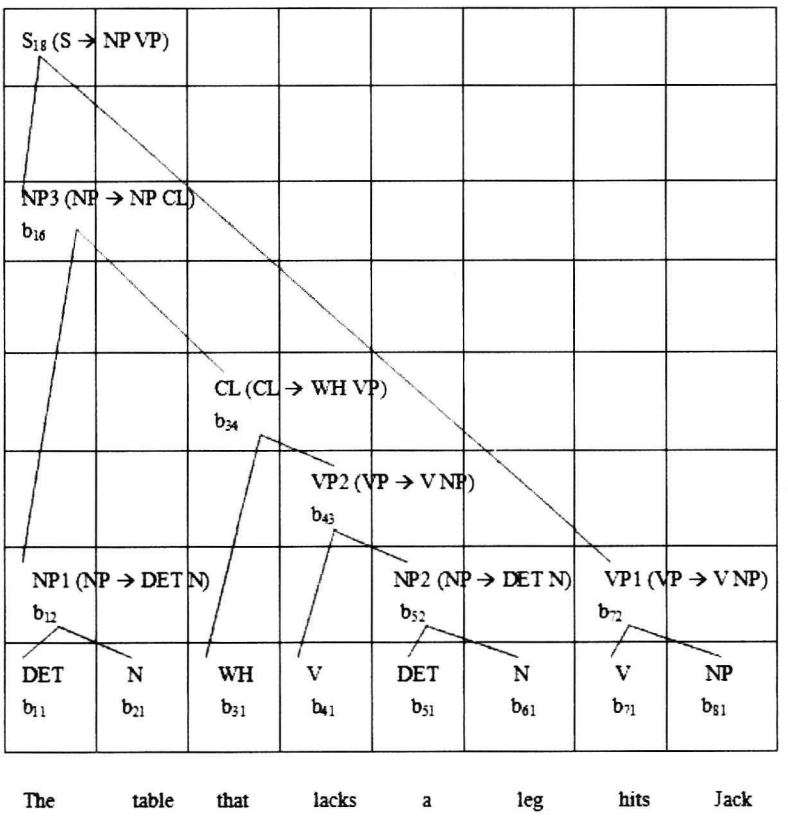

图 7 句子的方框和表

其中，各个方框中的 b_{ij} 计算详情如下：

b_{ij} (NP1): i=1. j=1+1=2

b_{ij} (NP2): i=5, j=1+1=2

b_{ij} (VP1): i=7, j=1+1=2

b_{ij} (VP2): i=4, j=1+2=3

b_{ij} (CL): i=3, j=1+3=4

b_{ij} (NP3): i=1, j=2+4=6

b_{ij} (S): i=1, j=2+6=8

这个句子的长度为 8，我们得到的 S 的方框中的行号也为 8，因此句子剖析成功。

我们使用 CYK 算法构造出图 7 中的表中的各个结点可以系连起来形成一个金字塔（pyramid），这个金字塔也就是一个树形图，它可以表示句子的结构。为了得到这样的金字塔，我们首先必须把全部的规则转换为 Chomsky 范式，影响了自动分析的效率。

为了严格遵守二分的原则，在规则的转换的时候，有时要进行分解，有时要进行合并。在上面的例子中，我们是将规则分解为 Chomsky 范式；有时我们还需要把一些规则合并成二分的 Chomsky 范式。

如果我们使用 CYK 算法来剖析句子“book that flight”，就必须对规则进行合并。

用于分析这个句子的上下文无关文法规则是：

S → VP

VP → Verb NP

NP → Det Nominal

Nominal → Noun

由于第一条规则 S → VP 的右手边只包含一个单独的非终极符号 VP，这不是 Chomsky 范式，但第二条规则 VP → Verb NP 是 Chomsky 范式，因此，我们把第一条规则和第二条规则合并，形成如下的符合 Chomsky 范式要求的规则：

S → Verb NP

第四条规则 Nominal → Noun 的右手边也只包含一个单独的非终极符号，也不是 Chomsky 范式，但第三条规则 NP → Det Nominal 是 Chomsky 范式，因此，我们把第四条规则和第三条规则合并，形成如下的符合 Chomsky 范式要求的规则：

NP → Det Noun

经过规则合并之后的上下文无关语法的规则如下：

S → Verb NP

NP → Det Noun

这些规则都符合 Chomsky 范式的要求了。只有在规则合并之后，我们才可以根据这样符合 Chomsky 范式要求的规则，使用 CYK 算法分析上述句子。结果如下：

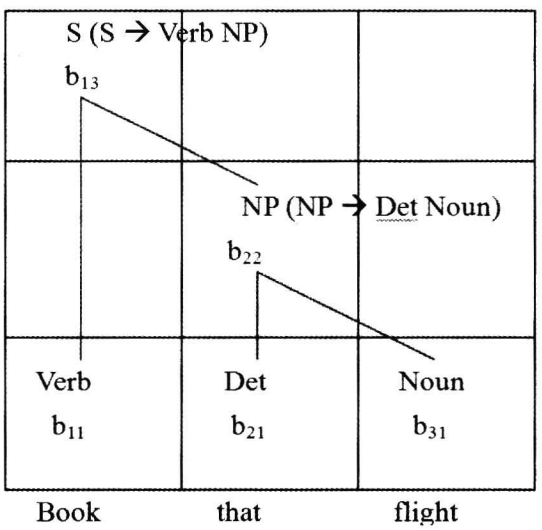

图 8 句子的方框和表

其中，各个方框中的 b_{ij} 计算详情如下：

b_{ij} (NP): i=2,. j=1+1=2

b_{ij} (S): i=1, j=1+2=3

用 CYK 算法造出的金字塔也就是表示句子结构的树形图。