一种无回溯的自然语言分析算法

冯志伟

(教育部语言文字应用研究所 北京 100010)

[摘要]在自然语言的自动分析中,如果回溯过多会严重地降低分析的效率。依尔利算法,可以完全避免回溯。本文介绍了依尔利算法的基本原理,它的三种基本操作,并以实例详细地描述了依尔利算法分析句子的过程。

[关键词]自底向上分析法;自顶向下分析法;依尔利算法;回溯;左角表;线图;点规则;非成圈有向图

[中图分类号]H08[文献标识码]A[文章编号]1003-5397(2003)01-0063-12

An Algorithm Without Backtracking for Natural Language Parsing

Feng Zhiwei

Abstract: In natural language parsing, too much backtracking seriously decreases the parsing effect. The Earley algorithm, however, can completely avoid backtracking. In this paper, the author introduces the basic principles of Earley algorithm and its 3 fundamental operators. The parsing process of a sentence is described in detail as an example.

Key words: bottom-up parser; top-down parser; Earley algorithm; backtracking; left-corner table; chart; dotted rule; Directed Acyclic Graph (DAG)

一 传统分析算法中的回溯问题

乔姆斯基的形式语法 G 是一个四元组

$$G = (Vn, Vt, S, P)$$

其中,Vn 是非终极符号的集合,Vt 是终极符号的集合,S 表示句子,P 是重写规则,P 的形式为:

$$A \rightarrow \omega。$$

[收稿日期] 2002 - 07 - 23

[作者简介] 冯志伟,男,教育部语言文字应用研究所研究员。

这里,A 是单个的非终极符号,ω 是符号串,它可以由终极符号组成,也可以由非终极符号组成,或者由终极符号和非终极符号共同组成。这样的重写规则是上下文无关的,具有这样的重写规则的语法叫做上下文无关语法(context-free grammar,简称 CFG),也叫做上下文无关的短语结构语法(context-free phrase structure grammar),或者叫做短语结构语法(phrase structure grammar,简称 PSG)。在本文中,这三个术语的含义是一样的。

本文中,我们提出如下简化了的英语的 CFG 文法来描写最常见的英语句子,其中各种符号的含义,熟悉英语的读者一看便知,不再解释。这个简单的 CFG 文法有一定的代表性和概括性,本文中的所有算法,都使用这同一个 CFG 文法,以便读者比较它们的优劣。

G = {Vn, Vt, S, P}

Vn = {S, NP, VP, Aux, Wh-NP, Pron, PrN, predet, Det, Card, Ord, Quant, AP, Nom. N. V, Adj, Adv, Wh-Pron, PP, GdVP, GdV, RelCL, who, that, and}

Vt = {the, a, that, table, leg, Jack, lacks, hits, …}

S = S

P: :1. S→NP VP;2. S→VP;3. S→Aux NP VP;4. S→Wh-NP VP;5. NP→Pron;6. NP→PrN;7. NP→(Det) (Card) (Ord) (Quant) (AP) Nominal;8. NP→Nom N;9. Wh-NP→Wh-Pron;10. Nom→N Nom;11. Nom→N;12. Nom→N PP (PP) (PP);13. Nom→Nom GdVP;14. GdVP→GdV NP (GdVP 表示 Gerund verb phrase, GdV 表示 Gerund verb);15. GdVP→GdV PP;16. GdVP→GdV;17. GdVP→GdV NP PP;18. Nom→Nom RelCl (RelCL 表示关系从句);19. RelCl→who VP (这个 who 是关系代词);20. RelCl→that VP(这个 that 是关系代词);21. AP→Adv Adj;22. AP→Adj;23. VP→V;24. VP→V NP;25. VP→V NP PP;26. VP→V PP;27. VP→V S;28. VP→V that S (这个 that 是关系代词);29. PP→Prep NP;30. NP→NP and NP;31. VP→VP and VP;32. S→S and S;33. Pron→{I, you, he, …};34. N→{table, leg, …};35. V→{lacks, hits, …};36. PrN→{Jack, John, …};37. P→{on, of, at, …};38. Aux→{does, has, …};39. Ord→{first, second, …};40. Det→{the, a, that, …};41. Card→{one, two, …}。

基于短语结构语法的自动句法分析方法,主要有自底向上分析法和自顶向下分析法两种。下面,我们用前面的 CFG 文法来分析同一个稍微复杂的英语句子"The table that lacks a leg hits Jack"(缺少一条腿的那个椅子打着了杰克),分别观察这两种分析方法中的回溯问题。

1.1 自底向上分析法(bottom-top parser)中的回溯

自底向上分析法从输入句子的第一个单词开始分析,顺次取词向前移进(shift),并根据文法的重写规则逐级向上归约(reduce),直到构造出表示句子结构的整个推导树为止。

自底向上分析法实际上是一种"移进-归约算法"(shift-reduce algorithm),它对于句子中的单词处理是顺次"移进",而利用文法中的重写规则是按条件"归约"。这种算法类似于计算机编译技术中的 LR 算法[自左(Left)向右(Right)算法]。移进-归约算法的信息存放方式主要是"栈"(stack),信息操作方式主要有移进、归约、拒绝、接受等方式。这种算法利用一个栈来存放分析过程中的有关"历史"信息(即关于已经走过的过程的信息),并且根据这种历史信息和当前正在处理的符号串来决定究竟是移进还是归约。所谓"移进",就是把一个尚未处理过的符号移入栈顶,并等待更多的信息到来之后再做决定;所谓"归约",就是把栈顶部分的一些符号,用文法的某个重写规则的左边的符号来替代,这时,这个重写规则的右边部分必须与栈顶的那些符号相匹配。采用这样的办法对栈中的符号以及输入符号串进行移进和归约两种操作,直

到输入的符号串处理完毕并且栈中仅仅剩下初始符号 S 的时候,就认为输入符号串被接受。如果在当前状态,既无法进行移进,也无法进行归约,并且栈中并非只有唯一的初始符号 S,或者输入符号串中还有符号未处理完毕,那么,输入符号串就被拒绝。

英语句子"The table that lacks a leg hits Jack"自底向上分析过程如下:

输入句子:"The table that lacks a leg hits Jack".

栈	操作	输入句子中的剩余部分
开始		The table that lacks a leg hits jack
The	移进	table that lacks a leg hits Jack
Det	用规则 40 归约	table that lacks a leg hits Jack
Det table	移进	that lacks a leg hits Jack
Det N	用规则 34 归约	that lacks a leg hits Jack
Det Nom	用规则 11 归约	that lacks a leg hits Jack
+ Det Nom that	移进	lacks a leg hits Jack
Det Nom Det	用规则 40 归约	lacks a leg hits Jack
NP Det	用规则 7 归约	lacks a leg hits Jack
	[回溯到 +]	
Det Nom that lacks	移进	a leg hits Jack
Det Nom that V	用规则 35 归约	a leg hits Jack
Det Nom that V a	移进	leg hits Jack
Det Nom that V Det	用规则 40 归约	leg hits Jack
Det Nom that V Det leg	移进	hits Jack
Det Nom that V Det N	用规则 34 归约	hits Jack
Det Nom that V Det Nom	用规则 11 归约	hits Jack
Det Nom that V NP	用规则 7 归约	hits Jack
Det Nom that VP	用规则 24 归约	hits Jack
Det Nom RelCL	用规则 20 归约	hits Jack
Det Nom	用规则 18 归约	hits Jack
NP	用规则 7 归约	hits Jack
NP hits	移进	Jack
+ + NP V	用规则 35 归约	Jack
NP VP	用规则 23 归约	Jack
S	用规则 1 归约	Jack
	[回溯到 + +]	
NP V Jack	移进	ϕ
NP V PrN	用规则 36 归约	ϕ
NP V NP	用规则 6 归约	ϕ
NP VP	用规则 24 归约	ϕ
S	用规则 1 归约	ϕ
	[分析成功 !]	

在上面的分析过程中,出现两次回溯。第一次回溯(回溯到 +)是由于把 that 过早地用规则 40 归约为 Det,而 that 后面没有名词,使得系统不能利用任何规则继续分析,因此只好回溯到 + ,把 that 作为关系代词来处理,这样系统就可以继续进行分析。第二次回溯(回溯到 + +)是由于把 NP + VP 过早地用规则 1 归约为 S,而在输入的句子中还剩下 Jack 没有处理,因此只好回溯到 + + ,移进 Jack,把 Jack 归约为 PrN,再进一步归约为 NP,从而使分析得到成功。

1.2 自顶向下分析法(Top-down parser)中的回溯

使用自顶向下分析法时,首先根据重写规则,从初始符号开始,自顶向下地进行搜索,构造推导树,一直分析到句子的结尾为止。

在搜索过程中,搜索目标首先是初始符号 S,从 S 开始,选择文法中适用的规则来替换搜索目标,并用文法规则的右边部分同句子中的单词相匹配,如果匹配成功,则抹去这个单词,在搜索目标中,记录下有关规则,然后,继续对输入句子中的遗留部分进行搜索,如果分析到句子的结尾,搜索目标为空,则分析成功。

当 CFG 文法的规则比较多的时候,从文法规则左部的同一个非终极符号出发来进行自顶向下搜索的可能性是很多的,常常会出现搜索落空的情况,使得自顶向下分析的效率很低。

为了提高自顶向下分析的效率,减少搜索的盲目性,我们给上面的 CFG 文法建立一个左角表(left-corner table),把与 CFG 文法规则左部的同一个非终极符号相配的规则右部的最左的非终极符号一一列举出来,搜索时,对于与文法规则左部的同一个非终极符号相配的规则右部的符号串,只搜索左角表中的成分,不必再搜索左角表中没有的成分,这样可以避免空搜索,大大提高自顶向下分析的效率。

例如,对于前面的 CFG 文法,我们建立如下的左角表(left-corner table):

非终极符号	左角
S	Det , Pron , PrN , Wh-Pron , Aux , Verb
NP	Det , Ord , Pron , PrN , N
Nom	N
VP	V
PP	Prep
AP	Adv , Adj
GdVP	GdV
RelCL	that , who

自顶向下分析的搜索过程中,我们根据这个左角表来选择准确的规则,过滤不必要的规则。例如,在我们的 CFG 文法中,以 S 为左部的规则有 4 条:

1. S →NP VP
2. S →VP
3. S →Aux NP VP
4. S →Wh-NP VP

为了分析 S,我们必须逐一地搜索这 4 种不同的情况,但是,我们句子的第一个词是 the,它的范畴是 Det,如果我们使用规则 S →VP , S →Aux NP VP , S →Wh-NP VP 来搜索,这些规则的右

部的左角与 Det 毫无关系,这样,系统就要做很多的空搜索,但是,如果我们利用左角表,可以知道 Det 是 S 的左角,而句子的第一个词 the 的范畴正好是 Det,这样,就可以马上搜索到 Det,下一步就直接转入规则 7,用不着去使用 S→VP, S→Aux NP VP y 以及 S→Wh-NP VP 等规则进行无用的空搜索了。

现在,我们根据前面 CFG 文法和这个左角表,采用自顶向下分析法来分析句子"The table that lacks a leg hits Jack",分析过程如下:

搜索对象	所用规则的号码	输入句子中的剩余部分
S		The table that lacks a leg hits Jack
NP VP	1	The table that lacks a leg hits Jack
Det Nom VP	7	The. table that lacks a leg hits Jack
+ Nom VP		table that lacks a leg hits Jack
N Nom VP	10	table that lacks a leg hits Jack
Nom VP		that lacks a leg hits Jack
	[回溯到 +]	
N VP	11	table that lacks a leg hits Jack
VP		that lacks a leg hits Jack
	[回溯到 +]	
N PP VP	12	table that lacks a leg hits Jack
PP VP		that lacks a leg hits Jack
	[回溯到 +]	
Nom GdV NP VP	14	table that lacks a leg hits Jack
N Nom GdV NP VP	10	table that lacks a leg hits Jack
Nom GdV NP VP		that lacks a leg hits Jack
	[回溯到 +]	
Nom RelCL VP	18	table that lacks a leg hits Jack
N RelCL VP	11	table that lacks a leg hits Jack
RelCL VP		that lacks a leg hits Jack
that VP VP	20	that lacks a leg hits Jack
++ VP VP		lacks a leg hits Jack
V VP	23	lacks a leg hits Jack
VP		a leg hits Jack
	[回溯到 ++]	
V NP VP	24	lacks a leg hits Jack
NP VP		a leg hits Jack
Det Nom VP	7	a leg hits Jack
+++ Nom VP		leg hits Jack
N Nom VP	10	leg hits Jack
Nom VP		hits Jack
	[回溯到 +++]	

	N VP	11	leg hits Jack
++++	VP		hits Jack
	V	23	hits Jack
	ϕ		Jack
		[回溯到 ++++]	
	V NP	24	hits Jack
	NP		Jack
	PrN	6	Jack
	ϕ		ϕ

[分析成功 !]

从分析过程可以看出,尽管我们使用了左角表来过滤,避免了大量的空搜索,但是,在分析过程中,仍然出现 7 次回溯。第一次回溯(回溯到 +),是由于使用规则 10 把 Nom 重写为 N + Nom,而在 N 匹配成功被抹去后,Nom 与输入的剩余部分的第一个词 that 不匹配。第二次回溯(回溯到 +),是由于使用规则 11 把 Nom 重写为 N,而在 N 匹配成功被抹去后,VP 与输入的剩余部分的第一个词 that 不匹配。第三次回溯(回溯到 +),是由于使用规则 12 把 Nom 重写为 N + PP,而在 N 匹配成功被抹去后,PP 与输入的剩余部分的第一个词 that 不匹配。第四次回溯(回溯到 +),是由于把 Nom + VP 中的后项 VP 使用规则 14 重写为 GdV + NP + VP,并且使用规则 10 把 Nom + GdV + NP + VP 中的前项重写为 N + Nom,而在 N 匹配成功被抹去后,Nom 与输入的剩余部分的第一个词 that 不匹配。第五次回溯(回溯到 ++),是由于使用规则 23 把 VP + VP 的前项 VP 重写为 V,而在 V 匹配成功被抹去后,后项 VP 与输入的剩余部分的第一个词 a 不匹配。第六次回溯(回溯到 +++),是由于使用规则 10 把 Nom + VP 中的前项 Nom 重写为 N + Nom,而在 N 匹配成功被抹去后,Nom 与输入的剩余部分的第一个词 hits 不匹配。第七次回溯(回溯到 ++++),是由于使用规则 23 把 VP 重写为 V,而在 V 与输入的剩余部分的第一个词 hits 相匹配被抹去后,搜索对象已经变空,但在输入中尚有剩余的词 Jack,两者无法匹配。在第七此回溯到 ++++ 之后,使用规则 24 把 VP 重写为 V + NP,最后分析到句子的结尾,搜索目标为空,导致分析成功。可见,在自顶向下的分析过程中,尽管我们使用左角表来过滤,回溯仍然还是非常严重的。如果不使用左角表来过滤,回溯就更加严重了。

不论采用自底向上分析法还是采用自顶向下分析法来分析句子“The table that lacks a leg hits Jack”,得到的分析结果都是一样的,可以用树形图表示如图 1。

自底向上和自顶向下分析法中普遍存在回溯严重地影响到分析算法的效率。为了避免回溯,提高分析算法的效率,美国计算机学者依尔利 (J. Earley) 在他的博士论文中,把自底向上和自顶向下的方法结合起来,有效地避免了回溯,他的博士论文中提出的这种算法,后来在计算语言学中被称为依尔利算法(Earley algorithm),这是一种高效的自然语言分析算法,他的算法对于自然语言的计算机处理,做出了重大的贡献。依尔利算法虽然已经提出多年,但是我国计算语言学界对此知之甚少。本文在此稍微详细地介绍,并以实例来说明其基本原理,以飨读者。

二 依尔利分析算法 (Earley algorithm)

2.1 线图和点规则(Chart and dotted rule)

依尔利算法的核心是线图(chart)。在表示语言信息方面,线图比树形图更为优越。在线

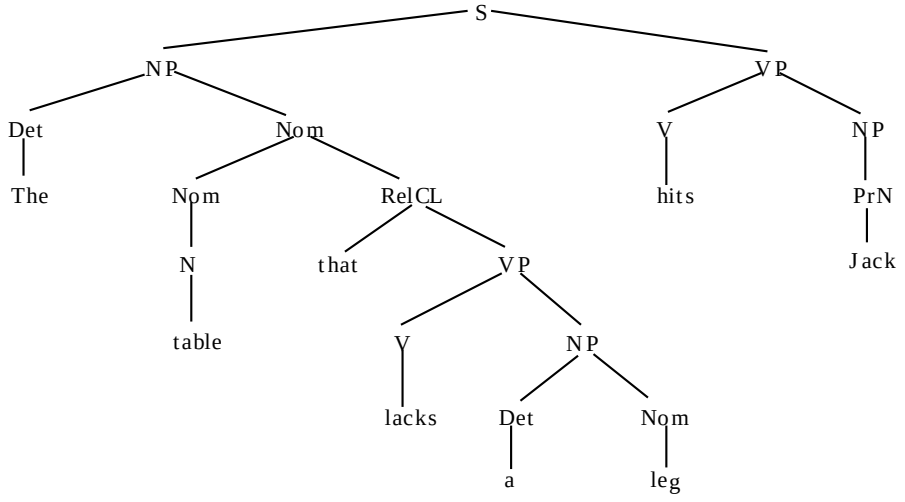

图 1　分析结果的树形图表示

图中,对于输入句子中每一个词,都可以用一个状态表来表示分析进程对于这个词所达到的位置。在句子分析结束时,线图可以把所有可能的分析结果都精练地表示出来。

线图中的状态包括三方面的信息:关于与语法中的某一规则相对应的子树(sub-tree)的信息;关于在这个子树的分析进程中已经完成了的分析结果的信息;关于这个子树当前所处位置与输入句子中相应的位置的信息。

这些信息采用点规则(dotted rule)来表示。所谓点规则,就是加了点的上下文无关文法的规则,它与一般的上下文无关文法规则的不同之处就在于规则中加了点,用这个点来表示这个上下文无关规则在分析进程中所处的状态。除了在上下文无关文法的规则中加点之外,在点规则中,还用两个数字来表示相应状态开始的位置和这个点在当前所处的位置。

例如,为了分析英语句子"book that flight"(请订这个航班),我们可以使用如下三条上下文无关文法的规则:S →VP;NP →Det Nom;VP →V NP。在分析的某一时刻,我们可以用如下三条点规则来表示分析的进程:S →. VP, [0,0];NP →Det. Nom, [1,2];VP →V NP. , [0,3]。

这时,分析的状态可以用在边(edge)上标明了点规则的如下线图来表示:

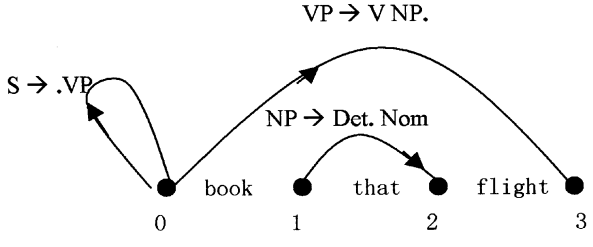

图 2　带点规则的线图

这个线图是一个非成圈的有向图(Directed Acyclic Graph,简称 DAG)。点规则"S →. VP, [0,0]"表示,计算机试图判断 S 是否是一个 VP,该规则所处的位置是:从结点 0 出发,到结点 0 结束,由于分析还没有完成,点处于规则右部的开头,这说明 S 的分析刚刚开始,这个点规则所在的边是活性边(active edge)。点规则"NP →Det. Nom, [1,2]"表示,在分析 NP 时,计算机已经分析了 Det(它处于点的左面),并且判明了它是输入句子中 that 的词类范畴,但是,计算机还没

有分析 Nom(它处于点的右面)。该规则所处的位置是:从结点 1 出发,到结点 2 结束,这恰恰是 that 在线图中的位置,这说明,计算机已经分析了 that,但是,还没有分析它后面的 flight,计算机还需要继续进行分析,这个点规则所在的边,也是活性边。如果计算机分析了 flight,并且判断它的范畴是 Nom,那么,这时的状态就可以用点规则"VP→V NP. , [0 ,3] "来表示。这个点规则在线图中的位置,是从 0 开始,到 3 结束;点的位置处于规则的末尾,表示计算机已经判明 VP 是由 V 和 NP 组成的,VP 的分析已经完成。这个点规则所在的边,已经分析结束了,已经不需要再分析了,这样的边是非活性边(non-active edge)。

2.2 依尔利算法的三种基本操作(Three operators in Early algorithm)

依尔利在他的算法中,提出了三种不同的基本操作:Predictor(预示)、Scanner(扫描)、Completer(完成)。它们的功能分述如下:

Predictor:它的功能是预示。在自顶向下的搜索过程中,Predictor 的作用是生成新的状态。来预示下一步可以做什么。Predictor 用于点规则中点的右边为非终极符号的那些状态,对于每一个这样的非终极符号,根据文法规则进行进一步的扩展。这些新生成的状态可加入到线图中去。Predictor 从所生成的新状态的位置出发,再回到同一个位置。例如,应用 Predictor 于状态"S→. VP , [0 ,0] ",可以生成新的状态"VP→. V , [0 ,0] "和"VP→. V NP , [0 ,0] ",并把它们加入到线图中去。

Scanner:它的功能是扫描。当状态中有一个词类范畴符号处于点的右边,Scanner 就检查输入句子,判断将要分析的单词的词类是否与这个词类范畴相匹配,如果匹配,就把点向右移动一个位置,并把新的状态加入到线图中。例如,在状态"VP→. V NP , [0 ,0] "中,点的右边是词类范畴 V,而在输入句子中恰恰分析到单词 book,而且根据规则"V→book. , [0 ,1] ",book 的词类范畴也是 V,两者相互匹配,这时,就把点向右移动一个位置,状态改变为"VP→V. NP , [0 ,1] ",并且把这个新的状态加入到线图中去。

Completer:它的功能是完成某一种分析。当状态中的点的右边是非终极符号,而在输入句子中,这个非终极符号所跨越的输入符号串已经分析结束,这时,就把该状态中点的位置向右移动到这个非终极符号的右边,并把新的状态加入到线图中。例如,如果经过 Scanner,计算机处于状态"VP→Verb. NP , [0 ,1] ",这时,输入句子中已经把跨越在结点 1 和 3 之间的符号串处理完毕,状态为"NP→Det Nom. , [1 ,3] ",其中的非终极符号 NP 与状态"VP→Verb. NP , [0 ,1] "中点的右边的 NP 相匹配,这时,就把状态"VP→Verb. NP , [0 ,1] "中的点向右移动到结点 3 的位置,得到新的状态"VP→Verb NP. , [0 ,3] "。

美国著名计算语言学家马丁. 凯依(Martin Kay)提出了"线图分析的基本规则"(fundamental rule of chart parser)。这个基本规则可以帮助我们进一步理解 Eraley 算法中的上述三种操作。

线图分析的基本规则可以稍微严格地表述如下:如果在线图中含有活性边 $(A \rightarrow W_1 . BW_2 , [i ,j])$ 和非活性边 $(B \rightarrow W_3 . , [j ,k])$,其中,A 和 B 是范畴,$W_1$, W_2 和 W_3 (可能为空)是范畴序列或词,那么,在线图中加一条新的边 $(A \rightarrow W_1 B . W_2 , [i ,k])$。

线图分析基本规则中没有明确地说明新的边是活性边还是非活性边,因为这完全取决于 W_2。如果 W_2 不空,那么,新的边就是活性边;如果 W_2 为空,那么,新的边就是非活性边。线图分析的基本规则可以用 DAG 表示如图 3:

例如,在分析"that flight "这个 NP 的过程中,当 that 已经判定为 Det,而 flight 只判定为 N 但还没有判定为 NOM 的时候,如果我们有活性边"NP→Det. Nom , [1 ,2] "和非活性边"Nom→N. ,

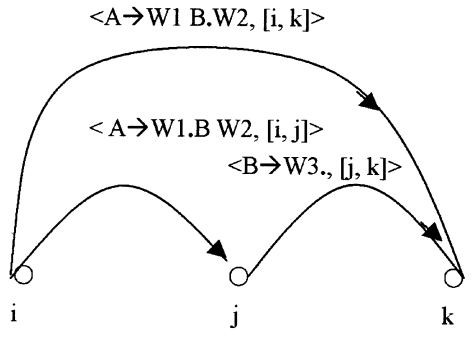

图 3　线图分析基本规则的 DAG 表示

[2 ,3] ",根据线图分析基本规则,我们就可以推出:"NP →Det Nom. , [1 ,3] "。

这样的操作把活性边"NP →Det. Nom , [1 ,2] "中的规则右部的点,从 Nom 之前推进到 Nom 之后,得到新的边"NP →Det Nom. , [1 ,3] ",从而可判定 flight 为 Nom,而且"that flight "为 NP。由于在新的边中,点的后面是空的,所以,这条新的边是非活性边。这里,A = NP ,B = Nom ,W1 = Det , W2 =φ,W3 = N. 。这个操作的 DAG 表示如下:

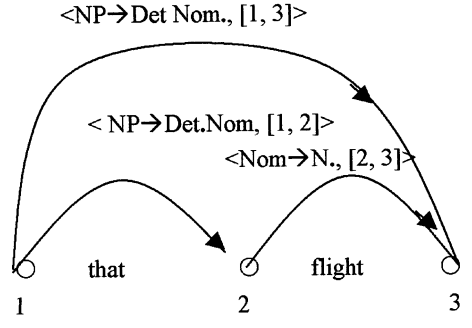

图 4　that flight 分析过程的 DAG 表示

下面,我们用依尔利算法来分析例句"The table that lacks a leg hits Jack "。这个句子中共有 8 个单词,因此,我们用如下的 9 个状态来表示它们:

●the　●table　●that　●lacks　●a　●leg　●hits　●jack　●
0　1　2　3　4　5　6　7　8

把状态作了这样的划分之后,我们就可以根据依尔利算法一步一步地写出线图中各个状态的分析情况了:

	Chanrt[0]	
γ→. S	[0 ,0]	Dummy start state
S→. NP VP	[0 ,0]	Predictor
NP→. Det Nom	[0 ,0]	Predictor
NP→. PrN	[0 ,0]	Predictor
S→. Aux NP VP	[0 ,0]	Predictor
S→. Wh-NP VP	[0 ,0]	Predicator
Wh-NP→. Wh-Pron	[0 ,0]	Predictor
S→. VP	[0 ,0]	Predicator
VP→. V	[0 ,0]	Predicator

VP →. V NP	[0 ,0]	Predicator
VP →. V PP	[0 ,0]	Predicator
	Chanrt[1]	
Det →the.	[0 ,1]	Scanner
NP →Det. Nom	[0 ,1]	Completer
Nom →. N Nom	[1 ,1]	Predictor
Nom →. N	[1 ,1]	Predictor
	Chanrt[2]	
N →table.	[1 ,2]	Scanner
Nom →N.	[1 ,2]	Completer
NP →Det Nom.	[0 ,2]	Completer
Nom →Nom. RelCl	[1 ,2]	Completer
RelCl →. who VP	[2 ,2]	Predictor
RelCl →. that VP	[2 ,2]	Predictor
	Chanrt[3]	
That →that. .	[2 ,3]	Scanner
RelCl →that. VP	[2 ,3]	Completer
VP →. V	[3 ,3]	Predictor
VP →. V NP	[3 ,3]	Predictor
	Chanrt[4]	
V →lacks.	[3 ,4]	Scanner
VP →V.	[3 ,4]	Completer
RelCl →that VP.	[2 ,4]	Completer
Nom →Nom RelCl.	[1 ,4]	Completer
NP →Det Nom.	[0 ,4]	Completer
(在 VP 之后还有 NP ,因此 ,在分析到 VP 是不行的)		
VP-V. NP	[3 ,4]	Completer
NP →. Det Nom	[4 ,4]	Predictor
	Chanrt[5]	
Det →a.	[4 ,5]	Scanner
NP →Det. Nom	[4 ,5]	Completer
Nom →. N	[5 ,5]	Predictor
	Chanrt[6]	
N →leg.	[5 ,6]	Scanner
Nom →N.	[5 ,6]	Completer
NP →Det Nom.	[4 ,6]	Completer
VP →V NP.	[3 ,6]	Completer
RelCl →that VP.	[2 ,6]	Completer
Nom →Nom RelCl.	[1 ,6]	Completer

NP →Det Nom.	[0 ,6]	Completer
S →NP. . VP	[0 ,6]	Completer
VP →. V.	[6 ,6]	Predictor
VP →. V NP	[6 ,6]	Predictor
	Chanrt[7]	
V →hits.	[6 ,7]	Scanner
VP →V.	[6 ,7]	Completer
S →NP VP.	[0 ,7]	Completer (S 的跨度为 7 < 8)
VP →V. NP	[6 ,7]	Completer
NP →. PrN	[7 ,7]	Predictor
	Chanrt[8]	
PrN →Jack.	[7 ,8]	Scanner
NP →PrN.	[7 ,8]	Completer
VP →V NP.	[6 ,8]	Completer
S →NP VP.	[0 ,8]	Completr
	[分析成功 !]	

上面只描述了各个状态内的识别过程。事实上,Predictor 并不参与句子的实际分析过程,因此,采用依尔利算法的句子分析过程要比上面的识别过程简单。下面,我们一步一步地把这个分析过程写下来:

Det →the.	[0 ,1]	Scanner
NP →Det. Nom	[0 ,1]	Completer
N →table.	[1 ,2]	Scanner
Nom →N.	[1 ,2]	Completer
NP →Det Nom.	[0 ,2]	Completer
Nom →Nom. RelCl	[1 ,2]	Completer
That →that. .	[2 ,3]	Scanner
RelCl →that. VP	[2 ,3]	Completer
V →lacks.	[3 ,4]	Scanner
VP-V. NP	[3 ,4]	Completer
Det →a.	[4 ,5]	Scanner
NP →Det. Nom	[4 ,5]	Completer
N →leg.	[5 ,6]	Scanner
Nom →N.	[5 ,6]	Completer
NP →Det Nom.	[4 ,6]	Completer
VP →V NP.	[3 ,6]	Completer
RelCl →that VP.	[2 ,6]	Completer
Nom →Nom RelCl.	[1 ,6]	Completer
NP →Det Nom.	[0 ,6]	Completer
S →NP. . VP	[0 ,6]	Completer

V →hits.	[6 ,7]	Scanner
VP →V. NP	[6 ,7]	Completer
PrN →Jack.	[7 ,8]	Scanner
NP →PrN.	[7 ,8]	Completer
VP →V NP.	[6 ,8]	Completer
S →NP VP.	[0 ,8]	Completer

[分析成功 !]

我们把分析结果用线图表示,见图 5。线图中的边都是有方向的,为简明起见,我们在线图中没有标明各个边的方向,读者可以自行补出。

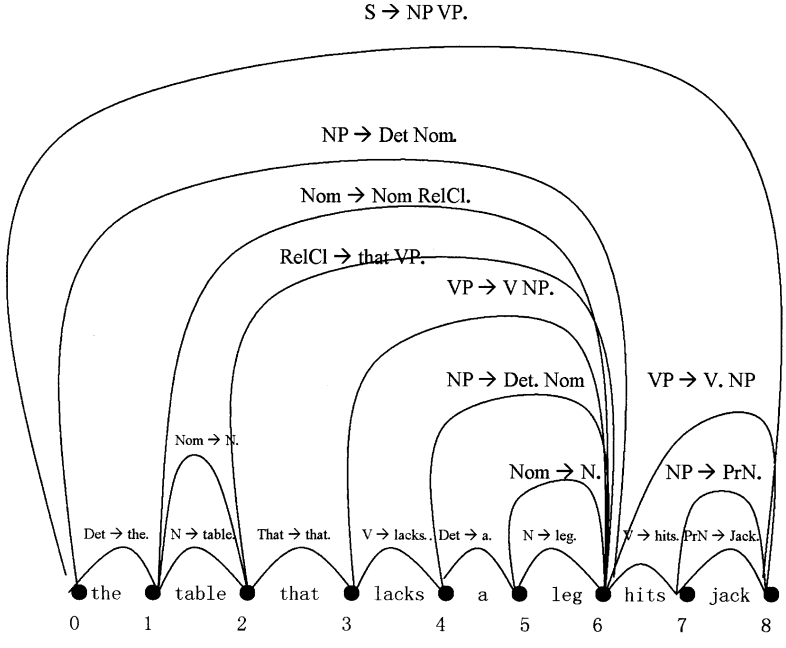

图 5 分析结果的线图表示

在使用依尔利算法分析句子的整个过程中,没有出现任何回溯,这有力地说明了依尔利算法确实比自底向上分析法和自顶向下分析法都优越。我们可以断言,依尔利算法完全避免了回溯,它大大提高了自然语言的分析效率,这是自然语言处理研究中的一个重要成果。近年来,依尔利算法也被引进到概率上下文无关文法的研究中,在基于统计的自然语言处理中,也发挥了它的积极作用。

[参考文献]

[1] J. Earley, An efficient context = free parsing algorithm, Communications of the ACM, 6(8), 451-455,1970.

[2] M. Kay, Algorithm Schemata and Data Structures in Syntactic Processing, Technical Report CSL. 80. 12., Xerox PARC, Oct, 1980.

[3] G. Gazdar, Ch. Mellish, Natural Language Processing in Prolog, Addison-Wesley Publishing Company, 1989.

[4] 冯志伟. 计算语言学基础[M]. 北京:商务印书馆,2001.

[5] 冯志伟. 基于短语结构语法的自动句法分析方法[J]. 当代语言学,2000,(2).

[6] 冯志伟. 通用句法处理器和线图分析法[J]. 当代语言学,2002,(4).