Tomita 增強型 LR 剖析器在機器翻譯系統的實際應用

唐安慶　　黃祖誠

工業技術研究院電子工業研究所

前言

電子所的英譯中機器翻譯系統，自1987年 7月開始發展，當時採用 ATN (Augmented Transition Network) 剖析器作為文法剖析用。經過一年的研究發展後，雖然它確有很強的功能，但在評估比較後，決定試用另一種方法，參考 Tomita 所設計的剖析器演算法 [1, 2] ，經過一年的設計及使用 ， 目前(1988年 8月)已經有了更新的面貌，並解決了一些難題。

本文共分六節，第一節是對剖析器系統的整體概略介紹，並提及剖析原理和剖析表 (parsing table) 的建構方式，第二節討論所使用的資料結構，第三節介紹文法中的限制條件 (restriction) 及評分 (scoring)功能，第四節介紹語意的處理，第五節介紹軟式失敗救援 (fail-soft) 原理，第六節則是實驗結果，評估及簡單的結論。

此套剖析系統，已成功應用在電子所的英中翻譯系統上，而對其他的自然語言應用領域，也能提供良好的應用功能支援。

1. 系統描述

自然語言分析在實際應用上有許多難題 ，為了提供有力的文法剖析方式，我們採用 Tomita 剖析演算法，並增加許多新功能，使得正確率與執行效率得以提高。

Tomita 剖析器是由日本學者 Tomita 所提出的觀念，為一種自然語言的文法剖析程序，其方法基本上是以由下而上 (Bottom-up) 的 LR 剖析器為基礎，並加強部份功能。我們為了提高剖析器的效率，已對其建構剖析表 (parsing table) 的方式加以修改，得以提高其效率。為了避免產生過多的剖析樹，提高執行效率，並保持正確率，我們加入了限制條件及評分功能；另外還有語意檢查(semantic check)以彌補只做語法分析的效能不足。對於不合系統文法而無法成功地分析出文法結構的句子，我們也提供了軟式失敗救援的功能，可保留部份分析的成果，以供機器翻譯系統的運用。

1.1 增強型 LR 剖析器的結構

增強型 LR 剖析器的主要構成份子如下:

a. 剖析器 (程式之主體)。
b. 剖析表
   動作表 (action table)
   指示讀取 (Shift) 和化簡 (reduce) 的動作。
   指示表 (goto table)
   指示轉移到新狀態 (state)。
c. 輸入字串, 以語形單元(lexical token), 含語形變化和詞類。
d. 圖形堆疊 (graph stack)
   存放符號 (symbol) 和狀態。
e. 剖析林 (parsing forest)

圖 1.1 為增強型 LR 剖析器的方塊圖。

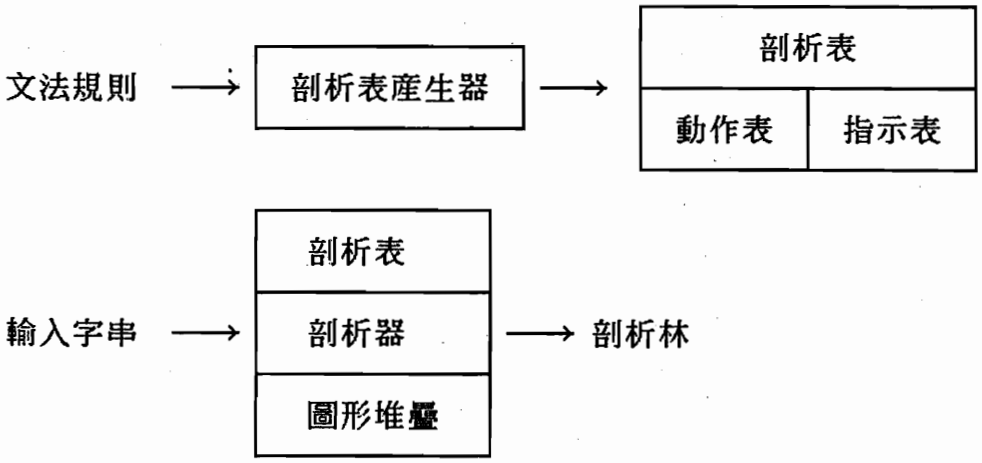


圖 1.1 增強型 LR 剖析器的基本模式

1.2 增強型 LR 剖析器的功能

為了增強剖析器的功能，我們作了幾方面的修正，以改善效率及功能。這些修正包括了下列幾項：

(1) 文法功能的擴充

為了解決語法混淆 (syntactic ambiguity)問題，我們參考Sager 的限制條件觀念，來限制剖析過程的進行。因此，我們在文法規則內，加入限制條件及動作 (action) 部份。另外，我們還將文法加以分層次，以提高剖析過程的效率與正確性。而評分更使文法的功能可以提昇到極致。

目前約有 600條不同的文法規則。為了加快速度及正確的剖析、易於維護等原因，我們將文法規則加以分層次，目前有 4個階層 (level)，其中階層 0是最常用且標準的句型，階層 1-3則是較不常用或是較特殊的文法。

在剖析一個平常的句子時，使用階層 0的文法就能容易地剖析成功。較複雜或較特殊的句子，則可能在階層 0階段無法剖析成功，而必須引入階層 1的文法。如此依序進行，把文法一層一層地放大，直到剖析成功為止。

(2) 剖析器功能的擴充

針對文法的改進，故而也要對剖析器加以修正。除了處理文法中的新功能外，還加上追蹤(Tracing) 文法功能及計時功能。

(3) 屬性 (attribute)

規則中的非終端節點 (non-terminal node) 可以有屬性，來表示其特性。

1.3 剖析原理

LR剖析演算法原先是為程式語言而設計的，是一個讀取化簡(shift-reduce) 剖析演算法，而由一個剖析表來引導進行以指引每一步驟所該採取的動作。LR剖析演算法有很高的效率，因為它是完全可決定的 (deterministic)，而且不作回溯追蹤(backtracking)及搜尋(search)。因此，它只適用於 LR 文法，但是任何實際的自然語言文法均不是 LR 文法，故而 LR 剖析技巧不能使用在自然語言上，必須再加以修改。

為了處理非 LR 的文法，因此剖析表須有多重欄項(multiple entry)，而動作表中的欄項也常需是多重定義(multiple defined)。增強型 LR 剖析器允許動作表有多重的讀取和化簡動作，剖析器部份也可以產生許多不同結構的剖析樹。其演算過程，請參考 Tomita 的論文 [2] 。

1.4 剖析表的建構

在進行由下而上剖析之前，首先必須建出剖析表。剖析表可以自動由剖析表產生器產生，這些剖析表描述各個狀態，以使剖析器可進行剖析。本系統之程式除了採用原有 LR(0)剖析表之外(其演算過程，請參考 Tomita 的論文[2])，實際製作時並利用編碼化(encoding)及雜湊表(hash tabe) 等技巧，提高執行效率十倍以上。

2. 資料結構

2.1 圖形堆疊

在程式中用一個堆疊 *GA* 來存放句子剖析的過程。在*GA*中所存的結構有兩種：

a. 一種節點代表狀態，則在*GA*中所存的結構如
   (26 23 24.....)
   其中26代表狀態編號，而 23, 24 則為此節點往回指向它前面的符號節點 (symbol node)。
b. 另一種節點代表符號，其在*GA*中所存的結構如
   (18 16.....)
   18 代表剖析林 *FA* 的指標(index)，指向符號放在 *FA* 的位置，16 則為符號節點往回指向它前面的狀態節點。

指標	內容
0	(0)
1	(0 0)
2	(4 1)
3	(1 0)
4	(2 3)
5	(2 4)
6	(7 5)
..	...

圖 2.1 圖形堆疊 *GA* 結構

2.2 剖析林的表示法

本系統的剖析演算法採用全路徑剖析 (all-path parsing) 方式，我們在程式中用一個陣列 *FA* 來存放剖析器所產生的剖析林，即可根據它來建立各個剖析樹。

在*FA*中，存放的格式如下：

(<node> (<index>...) <Rscore> <Lscore>)

其中，<node> 為節點名稱，<index> 指向其子節點，<Rscore> 表示經由規則評分函數 (Scoring Function) 計算出來的分數值，而 <Lscore> 則是經由詞類解模程式(Categorial Disambiguator)所分析出來的分數值。例子見圖 2.2。

指標	內容
0	(*N "I" 0 0)
1	(NP (0) 0 0)
2	(*V "saw-V" 0 90)
3	(*DET "a" 0 0)
4-9	....
10	(NP (3 5) 60 0)
11-19	....
20	(NP (6 8) 80 60)
21	(S (10 20) 240 150)
22-42	....
43	(S (10 42) 160 130)

圖 2.2 剖析林結構

2.3 文法之資料結構

此機器翻譯系統的文法，格式如下：

```
( A -> ( B1 ... Bn)
          :test <Fadvice>
          :action <Faction>
          :level <level>
          :score <Fscore> )
```

說明各個成份如下：

<Fadvice> 為文法的限制函數，預設值為 T，即不做測試。當剖析器要化簡(reduce)一條文法規則時，須先通過測試，符合條件才能化簡。<Faction> 存放一些動作函數，預設值為 T，即不做動作。在文法化簡之後，對所產生的樹進行某些動作。<level> 存放文法的階層，為一數值（範圍在 0-3），預設值為 0。當文法規則要被化簡時，剖析器會檢查此 <level>，合於正在進行的階層值，才准許化簡。<Fscore> 存放這條文法規則 的分數函數。

當剖析過程結束，若有多棵語意不清 (ambiguity) 而難以判斷正確性的剖析樹產生，則可藉由 score 之總分來挑選出最好的樹。

```
( S -> (SDEC) :score 20)
( S -> (ADVP "," SUBCL "," SDEC)
       :test (and (not_exist_node_p x1 ",")
                  (not_exist_node_p x3 ",")))
( S -> (PP "," SUBCL "," SDEC)
       :level 1)
( S -> (S *C S)
       :test (or (exist_node_p x1 SUBCL)
                 (exist_node_p x3 SUBCL) ))
( S -> (S "," SEMICOLON)
       :action (no_build_son_structure x3))
```

圖 2.3 文法規則舉例

3. 限制條件及評分功能

無疑地，在現有的文法系統下，任何文法撰作者都能定義出任意可能的語言結構，其中包含某些不合自然語言的結構。如果不加以限制，則會產生許多不合理的結構，也會增加挑選正確剖析樹的困難度。另一方面，如果文法撰作者希望能分析出一棵正確的剖析樹的話，則必須利用限制條件，刪除不合理的結構。有了限制條件後，就比較容易建構一個大而正確的自然語言文法。

為避免混淆 (ambiguity)的問題，很多文法都必須經由一些限制條件函數 (restriction function) 的測試，來限制不正確樹的成長。但由於限制規則有絕對性的影響，一旦測試不成立，即無法挽救。因此，當限制條件因某些因素而無法發揮功能時，便須評分功能來加以協助。這些因素包括有文法的例外情形，或語法混淆等等。使用評分並非最理想的方式，但在限制條件無法發揮功能時，便成了最佳的選擇。在無法確定之前，為了防止例外情況的發生，不得不採用評分方式。

3.1 限制條件 (Restriction)

在剖析器要化簡某一條文法規則之前，就先把規則限制條件加以運算，若得到真值 (true value) ，則進行化簡，若得到偽值 (false)，就放棄。在文法結構中的 <test> 這一欄，只能有一個函數，若有多於一個測試條件存在，可用 AND,OR,IF 等函數把測試條件整合起來。

如：

```
( S -> (S "," S)
       :test (or (exist_node_p x1  SUBCL)
                 (exist_node_p x3  SUBCL) ))
```

為免文法過於龐大，測試函數 (test function) 與文法分開定義，但並未完全分開，而是將限制條件涵蓋在文法內部，但是它使用到的測試函數則獨立定義，而由文法內的限制條件呼叫測試函數來達成限制文法作合理成長的目的。

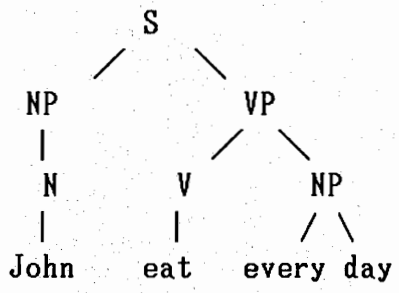
測試函數的定義及呼叫方式，與一般的 LISP 函數完全相同，這些測試函數不但適用於限制條件，也適用於評分。

以下面這條文法為例：

```
(VP -> (V NP)
       :test (VP_test1 x2)
       :act  (VP_act)  )
```

在測試函數 (test packet) 中，另外定義 VP_test1 這個函數，以減輕文法負擔，如此即可將文法與測試功能結合起來，發揮限制條件的功能。

舉例來說："John eats every day."
在字典中， "eats" 的屬性為 (class=V，sub=(1 21))
"day" 的屬性為 (class=N，sem_mark=(UNIT TIMP))
當剖析器要化簡下列的文法規則時，
(VP -> (V NP))
此時 V="eat"，而NP="every day"，便會產生如下的不合理剖析樹



因為"day"不能當"eat"的受詞，所以為解決此一問題，必須對文法加上如下限制條件，以使不合理的剖析結構無法產生。

```
(VP -> (V NP)
       :test (if (verb_type_is x1 1)
                 then (sem_mark_not x2 TIMP)
                 else <other_test>) )
```

在加入了限制條件後，就能檢查出 "eat"與"day" 不能構組成VP (動詞片語)，也就不會化簡成功。如此一來，就不會產生如上的不合理樹狀結構了。

限制條件的構成要素有三類：

(1) 主體 (subject)
用來指向節點或某個字，一律以 xi (i=1..6) 來表示。

```
(VP -> (NP *C NP)
       :test (or (NP_conj_test1 x1)
                 (NP_conj_test2 x3) ))
```

上例中，x1與x3分別指 (NP *C NP) 中的第一個NP與第二個NP。

(2) 測試函數

用來測試主體是否其有所限定的特性。

可分為下列五類：

a. 屬性的測試

檢查子結構 (component) 是否具有所限定的屬性及其值。

```
(is_plural_verb x1)          ; 複數動詞
(is_plural_noun x1)          ; 複數名詞
(next_word_class_is (Z P))   ; 下個字詞類
(eq_verb_form x1 x3)         ; 時態相同
(voice_is_passive x1)        ; 被動語氣
```

b. 字的測試

檢查子結構是否含有某個字，或是檢查文法規則的下一個字。

```
(next_word_is you)           ; 下個字
(word_is x1 to)              ; 這個字
(subsumed_word x1 (to for))  ; 含此字
```

c. 結構的測試

檢查某條子結構是否具有所限定的結構

```
(single_branch_to x2 PP)     ; 獨子
(exist_node  x2 PP)          ; 含此節點
```

d. 選擇性 (selection) 的測試

檢查各子結構間，是否具有所限定的關係 (relation) 存在。

```
(match_verb_prep x1 (VP18) x2 (P)) ; 一致性
(semantic_match x1 x2)             ; 語意一致
(number_agreement x1 x2)           ; 單複數一致
```

e. Common Lisp 函數

(3) 測試函數的連結

```
(not <test-function>)
(and <test-function1>
     <test-function2>)
(or  <test-function1>
     <test-function2>)
```

3.2 評分功能

評分功能的計算原理有下列幾項原則 :

a. 規則評分 (rule score) 由規則中的評分函數計算出，可由實驗中得出。
b. 所算出的分數，其範圍在 -100 到 +100 之間。正數表示可能性較高，分數越高者，可能性越高。零分表示正常情況。而負數則表示可能性較低。
c. 整個結構的部份評分 (local score) 為各個子結構的部份評分之和，再加上其規則評分。
d. 總評分最高的樹，即是剖析器的輸出結果，但並不表示 "最正確"，而只是依據現有的語法及語意知識作判斷的情形下，所得出的最可能的結果。

評分功能的計算方式如下：

a. 語彙總評分 = 句中各語彙評分的總合
b. 結構之規則評分 = 各子結構之規則評分總合
b. 規則總評分 = 句結構之規則評分
c. 剖析樹之總評分 = 語彙總評分 * 規則總評分

評分功能有如下的功能 :

a. 處理混淆問題
若剖析句子時，發現有兩棵以上的樹狀結構，而不易判斷何者較為正確合理時，便可對每一棵樹評分，再依評分高低加以區分之。

b. 解決連接詞 (conjunction) 問題
以下面句子為例，組成 NP 有兩個可能性。

"The man with a pen and a woman"

結構-1

" man (modifier = with pen) and woman "

結構-2

" man (modifier = with (pen and woman)) "

很難選擇任一種結構，亦即是兩種都有可能性，需視全文涵意才能判斷，這是剖析器很不容易做到的。但結構-2 的可能性較低，故而可加以扣分，以使剖析器能選擇結構-1。因此，文法中需加入一些評分函數 (如下頁) 。

```
( NP -> (NP *C NP)
        :test <NP_test>
        :action <NP_action>
  :score
    (+
      ;;; test for structure
      (if (same_structure_p x1 x3) then +30
       elseif (same_structure_type_p x1 x3) then +10
       elseif (similiar_structure_p x1 x3) then 0
       else -30)
      ;;; test for nominal head word
      (let (nomhd1 nomhd2)
         (setq nomhd1 (get_nomhd_word x1)
               nomhd2 (get_nomhd_word x3) )
         (if (eq nomhd1 nomhd2) then +50
          elseif (same_semantic_p nomhd1 nomhd2)
             then +20
          elseif (same_semantic_class_p nomhd1 nomhd2)
             then 0
          else -30 ))) )
```

再如下例，組成 NP 亦有兩個可能性。

"The man with a pen and the ink"

結構-1

" man (modifier = with pen) and ink "

結構-2

" man (modifier = with (pen and ink)) "

由上面的文法評分結果，可判斷出結構-2 的可能性較高。

c. 解決PP（介系詞片語）的修飾問題
請參見第 4 節的討論。

4. 語意的處理

由於自然語言的特性，只考慮語法結構（syntactic structure） 的話，並不能解決問題。要解決這種語意混淆（semantic ambiguity）問題，有兩種解決方式，一是由電腦與使用者溝通，以找出正確的剖析樹，但此種方式很耗時間，另一種方式是由電腦根據知識庫（knowledge base）所提供的知識來判斷。大部份的機器翻譯系統都採用後者，一般是使用語意分析（semantic Analysis）技巧來解決混淆問題。我們採用的方式則是，同時使用語意檢驗（semantic checking）及格位分析（case analysis)兩種技巧，以使翻譯過程

能達全自動化，更希望能提升剖析能力，以提高翻譯正確率。有關格位分析的討論請參考本所的另一篇論文。

在剖析過程中，語法分析及語意分析同時進行。每次做語法分析時，即呼叫語意檢驗常式（semantic check routine）進行處理並得到一個評分。

4.1 介系詞片語修飾（PP Attachment）問題

因為介系詞片語修飾問題，在剖析過程中佔了很重的份量，所以我們把它列為第一優先的處理對象，集中全力加以解決。

我們處理介系詞片語修飾的策略如下：

1. 發現有右向修飾（Right Attachment）問題，立刻檢查是否可把介系詞片語（PP）接在正確的名詞片語（NP）上。
   檢查結果可能有下列三種情形：

   ① 只有某一個 NP-PP 配對(pair) 符合修飾規則（Attachment rule），就將該 PP 修飾該 NP 上，並予以加分。
   ② 有兩個以上的 NP-PP 配對是對的，便使用修飾規則，加以評分。
   ③ 找不到任何成功的 NP-PP 配對，不做任何修飾。

2. 檢查是否可將介系詞片語接在某一個動詞上。
   檢查情形有下列兩種：

   ① 有一個 VP-PP 配對不成立，不做任何修飾。
   ② 針對 VP-PP 配對加以扣分，但仍做修飾。

3. 完成修飾。

以下面句子為例，介系詞片語修飾性有兩個可能性。

" I hit the man with the hammer ."

可能性 1：

" I hit man (modifier = with hammer) "

可能性 2：

" I hit (modifier = with hammer) man "

為了解決此一問題，就必須先建立語意知識，其中包含了名詞及動詞的語意分類，即語意標記（semantic marker）。因為句中的關鍵字配對（hit, man, hammer）可以滿足規則模板（Preplate）中的一條規則（*STRIK,*HUMAN, *INST），其中的 *STRIK等表示語意標記，因此就可以從下面這條規則得到合理的分數。

```
(VP --> (V NP PP)
    :score (if (match_verb_preplate_p x1 x2 x3) then +50
               else -20 ))
```

4.2 連接詞問題

對於 " B1 and B2 and B3 ... " 等連接片段的問題，是令很多系統頭痛的。我們處理的方式如下：

a. 採用限制條件殺掉不合理的結構
b. 採用評分功能來判斷可能的結構
c. 判斷方式
    c1. 欲結合的成份 (constituent)，結構越接近，分數越高
    c2. 欲結合的成份，語意越接近，分數越高

5. 軟式失敗救援 (Fail-soft) 原理

當一個句子經過剖析過程後，並不表示一定可以剖析成功。如果剖析失敗，連一棵剖析樹也無法產生，若不加以補救，則所有剖析過程做過的努力將付之東流，而且也無法針對目前狀況加以了解問題的失敗點出在何處。為了解決這個問題，就必須採用軟式失敗救援技術，以達成救援的目的。

剖析失敗的可能有下列四種：

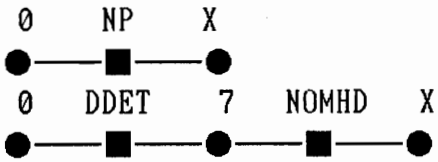
a. 文法知識庫所建的文法不齊全。
b. 限制條件尺度過嚴，以致誤殺正確的樹。
c. 詞庫記載的資料不正確。
d. 輸入的句子本身就有錯誤。

軟式失敗救援可以讓每一個句子都能順利進行剖析，都能產生剖析樹，都能得出翻譯結果。軟式失敗救援的原理是，當所有可能剖析的路徑都中斷，而無法繼續進行化簡時，就從中斷的路徑之中，找出一條最完整的路徑，再和尚未剖析過的剩餘字串銜接起來，構成一棵剖析樹，再由剖析器送出結果。

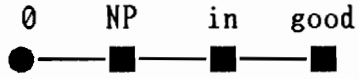
以下面句子為例：

The book in good.

當剖析到 "in" 時，假設已經全部中斷而不能執行下去，假設此時中斷的路徑有兩條：(借用 Tomita [2]的表示法)



則軟式失敗救援模組選擇 NP 這條路徑，並直接串上 "in good" 這兩個字，形成如下的結構。



至此，軟式失敗救援模組便將這棵經過特殊處理的樹送出，再由生成轉換模組據以完成翻譯的功能。

6. 實驗結果及評估

此系統所用改良過的剖析器，是以 Common Lisp 製作，在 Lambda Lisp Machine 或 PC/AT+Humming Board 上執行。執行結果，發現效率上改進很多。實驗測試採用一本有關計算機系統的書，書名是 The Design of UNIX Operating system [Maurice J. Bach]。 我們從第一章中，整理出 285 個句子作為測試用。

6.1 效率上的改進

以一般不超過 10 個字長的句子（ 而且不含連結性 (conjunction)），若文法不加限制條件等功能的話，會產生約 200 到 2000 棵的剖析樹，所花費的剖析時間（以所耗用的時間計）約在 2 分到 15 分之間。但在增強後，剖析樹可減至 1 到 4 棵，而且剖析時間只需 10 到 30 秒左右。如果再以10個字長以上的句子來測試的話，大部分的句子都會有 2000 棵以上的樹，甚至有高達上萬棵樹的情形，不但需執行幾小時，而且佔記憶體(memory)過多。

6.2 正確率的提高

根據測試結果顯示（在 285個句子的範圍內）正確率高達 99 % 以上。而經以更多句子(使用 4500 個句子，來自 7 篇科技性書籍文件）測試後，成果也十分良好。以 10 個字長以內的句子來說，正確率可提高到 80 % 以上。而 10 個字長以上的句子，其正確率也有 50 到 60 % 的表現。

6.3 結論

由以上情況可見，在全路徑剖析 (all-path parsing) 的演算法下，必須加以限制條件，否則效率及正確率無法提高。

本文說明 Tomita 增強型 LR 剖析器在一個英中機器翻譯系統的成功應用，並描述目前此系統所增加限制條件、評分功能、語意處理、及軟式失敗救援等功能的目前狀態。雖已獲致相當成果，仍有未盡如意之處，有待進一步研究。

致謝

此機器翻譯系統承蒙經濟部的電腦與通訊技術發展計劃(計劃編號: *ITRI-048-P001 (77))支持，特此誌謝。而在電子所張照煌、黎偉權、李炳煌三位先生之指導下，方得以順利進行，在此致以誠摯的謝意。

參考文獻

[1] Tomita, M.

An Efficient Augmented-Context-Free Parsing Algorithm
Computational Linguistics, vol:13:1-2, p.31-46
Jan-Jun, 1987.

[2] Tomita, M.
Efficient Parsing for Natural Language.
Kluwer Academic Publishers, 1986

[3] Naomi Sager

Natural Language Information Processing
-- A Computer Grammar of English and Its Applications
Addison-Wesley, 1981

[4] Dahlgren, K. and J. McDowell

Using commonsense knowledge to disambiguate prepositional phrase modifiers.
proceedings of AAAI-86, August 1986.

[5] Aho,A.V. and Ullman,J.D.

Compilers, principles, techniques, and tools