

TR-267

最近の構文解析の話題－
並列構文解析について

松 本 裕 治

June, 1987

©1987, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

最近の構文解析の話題

— 並列構文解析について —

松本 裕治

(新世代コンピュータ技術開発機構)

コネクショニストモデルなどの大規模並列の話題と共に、並列計算や並列アルゴリズムが最近見直されている。I C O Tでも並列論理型言語を実行させるべく並列論理マシンの研究を行っている。

ここでは、構文解析の最近の話題として、少々おこがましいけれども、著者が開発した(並列論理型言語による)並列構文解析アルゴリズム、および、コネクショニストモデルに基づく並列構文解析の方法をいくつか紹介しようと思う。

1. はじめに

自然言語の文法の記述について、最近の文法理論(LFG、HPSG等)では文法のクラスとして文脈自由文法程度を設定しておいても十分な記述が可能であることが示されている。このような文法形式に基づく構文解析について考えると、文脈自由文法は本質的に曖昧な文法を含んでおり、一般の文脈自由文法を対象とする構文解析には決定的な解析法がなく典型的な探索問題となっている。

最近、コネクショニズムや論理型言語の並列実行等の話題と相俟って、並列処理に関する研究が盛んである。構文解析はこのような並列処理によって飛躍的な効率改善が期待できる恰好の問題であり、いくつかの研究が見受けられる。

ここでは主に文脈自由文法を想定し、これを対象とする並列構文解析法を紹介しようと思う。よく知られているように、文脈自由文法の構文解析に要する計算時間は、逐次的な計算モデルでは入力文の長さを n とすると $O(n^3)$ 必要である[Aho 72]。つまり、入力文の長さの増加にしたがって、その3乗に比例する解析時間が必要な誤である。一方、並列計算については計算時間を評価するための共通に認められた土俵(並列計算モデル)がないのが問題ではあるけれども、例えば、P-RAM(Parallel Random Access Machine)を想定すると、文脈自由文法の認識問題には $O(\log n)$ しか必要でないことがわかっている[Rytter 85]。ただし、P-RAMというのはメモリを共有する多数のプロセッサからメモリへの同時アクセスを仮定していたり、あるデータの状態に対するプロセッサの割り当てが一定の時間でできることを仮定していたりして、並列計算モデルとしては現実的ではない。また、時間的には効率のよい $O(\log n)$ のアルゴリズムでは、かなりの量の無駄な計算が起こってしまう。ここでは、効率上の観点からも無駄な処理が少なく、しかも並列実行が期待できる構文解析法のいくつかを

紹介しようと思う。

まず、次章では、並列論理型言語での実現を目的とした著者らの並列構文解析アルゴリズムを説明する[Matsumoto 86]。これはもともと文脈自由文法用の並列構文解析法として考えられたものであるが、最近、文脈依存文法や更にそれ以上の記述力をもつ文法形式についても同じアルゴリズムが適用できることがわかった[Matsumoto 87]。また、解析中に同一処理の繰り返しを行わないので、逐次型のアルゴリズムとしてもEarleyのアルゴリズム[Earley 70]やチャートパーズング[Kay 80]と比較しても劣らないので、Prolog上での実用的な構文解析システムとしても利用できる[松本 86][杉村 86]。

第3章では、文脈自由文法による構文解析をコネクショニストモデルに基づいて実現した例として、Fantyのシステムを紹介する[Fanty 85]。基本的には、CYKアルゴリズム[Aho 72]で用いられているテーブルを縦直な形でコネクショニストネットワークとして実現したものである。

第4章では、文脈自由文法に基づく構文解析とはかなり観点が外れるが、最近の並列構文解析の話題として有名なWaitz & PollackのMassively Parallel Parsing[Waitz & Pollack 85]を紹介する。構文解析の観点から外れると言ったのは、彼らの対象が文の構文にあるのではなくどちらかというと意味解析まで含めた曖昧性の除去のための枠組みをコネクショニズムの立場から提案しているからである。

2. 並列論理型言語による並列構文解析

GHCやPARLOGなどの並列論理型言語のプログラムは、通信を交わし合うプロセスの集まりによって記述される。ここでは、文脈自由文法を対象にして、どのような考え方によって自然な並列構文解析が実現できるかを説明する。

次のような簡単な文脈自由文法を考えてみよう。釣

括弧（ '['、']' ）内は単語（終端記号）であり、それ以外は文法カテゴリ（非終端記号）である。

【文法1】

- (1) sentence --> np, vp.
- (2) np --> det, noun.
- (3) noun --> adj, noun.
- (4) noun --> noun, relc.
- (5) relc --> [that], vp.
- (6) vp --> verb.
- (7) vp --> verb, np.
- (8) det --> [the].
- (9) adj --> [beautiful].
- (10) noun --> [man].
- (11) noun --> [woman].
- (12) verb --> [walks].
- (13) verb --> [loves].

この文法に含まれるどの文法規則も右辺に三つ以上の文法カテゴリを含んでいない。以下では、このような所謂チョムスキー標準形の文脈自由文法に限って説明する。これは議論を簡単にするためであって、このような制限をもたない文法に対するアルゴリズムの拡張については最後に概略を述べることにする。

さて、単語や文法カテゴリなど文法規則を構成する基本要素をプロセスと考えるようなパーシングモデルを仮定して見よう。入力文として与えられた単語列のそれぞれの単語が自立的にプロセスとして動き、隣り合うプロセス同志が通信をし合いながら文を作っていくようなモデルである。

その場合、単語は何をすればよいだろうか。例えば、'man' という単語は文法規則(10)によれば名詞 (noun) であるから、これはすぐに 'noun' に生まれ変わればよい。これは(10)の文法規則をボトムアップに使用することに相当する。このように、右辺にただ一つの要素しか含まない文法規則に相当する処理は非常に単純に実現できる。例えば、文法規則(6)によれば、動詞 (verb) は直ちに動詞句 (vp) を生めばよい。それではそれ以外の形の文法規則に含まれる文法カテゴリはどう処理すればよいだろうか。

文法規則(2)について考えてみよう。これを用いてボトムアップに解析するというのはつまり、冠詞 (det) と名詞が隣り合わせて存在するときにこれらから名詞句 (np) を生み出すことに相当する。det も noun もプロセスとして表現することを考えているから、この場合はどちらかのプロセスが自分の存在を一方のプロセスに伝えてやればよい。今、このような文法規則を扱うときに、右辺に現れる文法カテゴリのうち常に左の要素が右の要素に対して従属的な立場にあると決



図1. 文法規則(2)の使用例

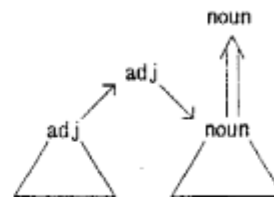
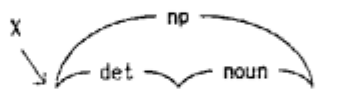
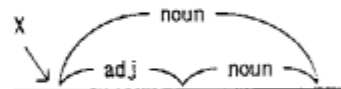


図2. 文法規則(3)の使用例



(a)図1の文法規則の接続関係



(b)図2の文法規則の接続関係

図3. 文法カテゴリの接続関係

めてしまおう。従って、det は自分の存在を示すために自分の名前を右のプロセスに送り、右のプロセス noun はそれを受けて np というプロセスを生み出せばよい訳である（図1）。その他の文法規則についても同様であり、特に noun を右辺に含む文法規則(3)についても図2のように noun は同様の役割を果たす。

従って、det や adj が行う処理というのはこの場合非常に単純である。それぞれ自分の名前を右のプロセスに送るだけでよい。プログラムで書けば次のようになる。

```
(14) mode det(?, ^).
      det(X, [det(X)]).
```

```
(15) mode adj(?, ^).
      adj(X, [adj(X)]).
```

このプログラムは PARLOG による記述であり、mode は述語の引数の入出力の宣言である。det(?, ^) は、第一引数が入力引数、第二引数が出力引数であることを宣言している。これらはそれぞれ自分の左のプロセスからの入力と右のプロセスへの出力に相当している。

よって、(14)の意味は、det は右からの入力に関係なく左側へ det(X) というデータを送るということである。det(X) がリストに含まれた形になっているのは、一般にプロセスが授受するデータは複数個あるのでそれらを集合として表現するためにリストを使うからである。この例では送るデータがたまたま一つだけであつたに過ぎない。adj についての定義(15)についても同様である。さて、これら二つが自分の名前だけでなく入力引数 X も同時に送っていることに注意してほしい。図3を見ると、det が送ったデータを noun

n が受け取ったときにできる np はもともと det の右にあった要素と接するのであるから、det が受け取っていたのと同じデータ（すなわち X）を受け取る必要がある。X 送られているのはこのようにいずれ np ができた時にその入力としてこの入力引数を渡すためである。

noun の定義も同様にプログラムによって記述してみよう。noun は文法規則(4)の右辺にも含まれているので、図1、図2に対応する処理以外にこれに対応する処理も行なわねばならない。

```
(16) mode noun(?, ^).
    noun(X, [noun(X) | Tail]) :- noun2(X, Tail).

mode noun2(?, ^).
noun2([], []).
noun2([det(X) | Tail], Y) :-
    np(X, Y1), noun2(Tail, Y2),
    merge(Y1, Y2, Y).
noun2([adj(X) | Tail], Y) :-
    noun(X, Y1), noun2(Tail, Y2),
    merge(Y1, Y2, Y).
noun2([_ | Tail], Y) :- otherwise !
    noun2(Tail, Y).
```

noun(X) というデータが出力されている部分が文法規則(4)に含まれている noun に相当する。noun2 の定義が図1、図2の下辺の noun の働きに対応する。

noun2 の第2番目のPARLOG節は図1の noun に相当し、det(X) というデータを受け取ると np というプロセスを生成している。np が第一引数として X を受け取っているのは、上で説明し図3に示されているように、この新しく得られた np の受け取るべき入力データが X に他ならないからである。また、noun2 は、他にも入力データを受け取るかも知れないので、残りのデータ（Tail部分）に対して再び noun2 が呼ばれている。このPARLOG節の本体部の np と noun2 の出力が merge（第一引数と第二引数のリストの和集合を第三引数に返す述語）によって一つのリストにまとめ上げられ、全体の出力となっている。これは、図3の(4)において noun と np の右端が一致しており、これらの右隣にある文法要素が受け取るのがこれら両方の出力である必要があるからである。noun2 の第三のPARLOG節についても同様である。第一のPARLOG節は、入力データを処理し終わったときにプロセスを終了させるための定義であり、第四の節は、入力データが det(X) でも adj(X) でもなかったとき（すなわち、名詞にとって関係ないデータであったとき）にそれを読み飛ばすための定義である。

その他の文法カテゴリについても同じようにして、

並列論理型言語による定義が可能である。図5に【文法1】を PARLOG プログラムに変換した結果を示す。このプログラムを用いると、例えば(17)の文の構文解析を行うには、(18)のようなゴール列を初期ゴールとして呼べばよい。

(17) The man loves the beautiful woman.

```
(18) the([begin], D1), man(D1, D3), loves(D3, D4),
    the(D4, D5), beautiful(D5, D6), woman(D6, D7),
    fin(D7).
```

この入力文が解析される様子を図6に示す。図中、太い矢印はプロセスの生成に相当し、細い矢印はデータの送受に対応する。あるプロセスは図4の(1)のように入力データの値とは無関係にデータを出力し、また、あるものは図4の(2)のように入力データに依存して新しいプロセスを生成する。構文解析の手続きにおいて逐次性が生じるのは図4の(2)のような場合だけであって、その他の部分がすべて並列に動作できる。特に、図4の(1)の場合に、入力引数がたとえ変数のままであっても、データを出力できることに注意してほしい。

プログラムの実行の結果知りたいのは入力文から sentence が構成できるかどうかであるので、図5のプログラムで sentence の定義だけが少し特殊になっている。[begin]というのは、(18)からわかるように入力文の先頭の単語に与えられる値である。もしこれが sentence に受け取られるとすると、その意味は、得られた sentence の左端が文頭に一致しているということである。この sentence の右端が文末に一致していることが解析が成功する条件である。それを調べるために、sentence は、[begin]を受け取ると end という新たなデータを右のプロセスに送る。fin というプロセスが(18)の文末に置かれているが、これが文の完成を調べるためのプロセスであり、単に end というデータを待っているだけのプロセスである。上の説

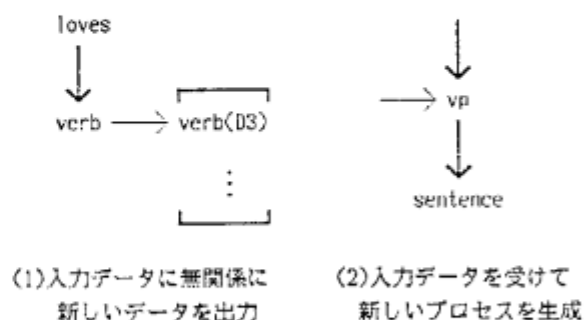


図4. プロセスのデータの授受

```

np(X,[np(X)!Tail]) :- np2(X,Tail).
np2([],[]).
np2([verb(X)!Tail],Y) :-
    vp(X,Y1),np2(Tail,Y2),merge(Y1,Y2,Y).
np2([_!Tail],Y) :- otherwise |
    np2(Tail,Y).

vp([],[]).
vp2([np(X)!Tail],Y) :-
    sentence(X,Y1),vp2(Tail,Y2),merge(Y1,Y2,Y).
vp2([_!Tail],Y) :- otherwise |
    vp2(Tail,Y).

det(X,[det(X)]).
adj(X,[adj(X)]).

noun(X,[noun(X)!Tail]) :- noun2(X,Tail).
noun2([],[]).
noun2([det(X)!Tail],Y) :-
    np(X,Y1),noun2(Tail,Y2),merge(Y1,Y2,Y).
noun2([adj(X)!Tail],Y) :-
    noun(X,Y1),noun2(Tail,Y2),merge(Y1,Y2,Y).
noun2([_!Tail],Y) :- otherwise |
    noun2(Tail,Y).

```

```

recl([],[]).
recl([noun(X)!Tail],Y) :-
    noun(X,Y1),recl(Tail,Y2),merge(Y1,Y2,Y).
recl([_!Tail],Y) :- otherwise |
    recl(Tail,Y).

that(X,[that(X)]).

verb(X,[verb(X)!Y]) :- vp(X,Y).

sentence([],[]).
sentence([begin],[end]).
sentence([_!Tail],Y) :- otherwise |
    sentence(Tail,Y).

the(X,Y) :- det(X,Y).

beautiful(X,Y) :- adj(X,Y).

man(X,Y) :- noun(X,Y).

woman(X,Y) :- noun(X,Y).

walks(X,Y) :- verb(X,Y).

loves(X,Y) :- verb(X,Y).

```

図5. 文法1の並列構文解析プログラム

入力文: the man loves the beautiful woman.

初期ゴール: the([begin],D2),man(D2,D3),loves(D3,D4),the(D4,D5),beautiful(D5,D6),woman(D6,D7),fin(D7).

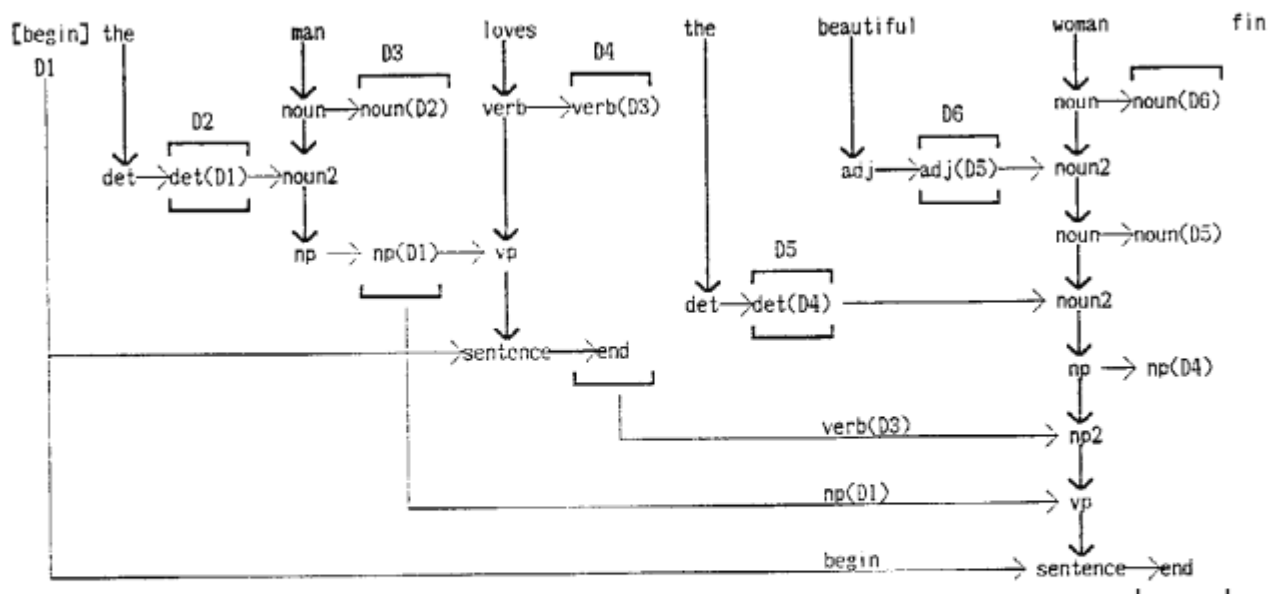


図6. 並列構文解析の実行例

明からわかるように、end というデータは文頭から完成した sentence だけが発生するデータであり、これが文末に位置する fin に受け付けられたときに初めて文の解析が成功したことがわかるのである。

図6を最終的に得られた sentence から逆向きに眺めてみると、これが構文解析木に対応していることがわかるだろうか。これにより、この並列構文解析が与えられた文法に基づくボトムアップ構文解析を実現していることがわかって頂けると思う。

構文解析のアルゴリズムの計算の複雑さとして、注意しておきたいのは、一旦ある文法カテゴリを根にもつ構文解析木が得られるとそれはプロセスとして生まれ、その左に隣接するすべてのプロセスからの出力を受け取り、そのデータを調べ尽くすまで生き続けることである。また、まったく同じプロセスは生成されることがなく、このような意味で重複計算がない。

トップダウン予測

ここまで説明した並列構文解析は純粋にボトムアップ構文解析であり、所謂トップダウン予測を行っていない。ここでは、今まで述べた並列構文解析法に簡単にトップダウン予測を取り入れることができることを説明する。

例えば、文頭に動詞が現れたと仮定しよう。求めたいのは sentence であり、文法1からは、動詞を先頭とする文はどのようにしても構成できない。ところが、文頭の単語が冠詞ならば、文を構成する上で問題がない。このようなことがわかるためには解析の各段階でどのような文法カテゴリを求めようとしているのかを利用するようにしなければならない。

図4の(1)のように入力データに関係なく新しいデータを生成している部分があった。トップダウン予測はこのようなデータの出力を一時的に抑制する処理として実現できる。

例えば、冠詞が得られて、det(X) というデータを出力しようとしたと仮定しよう。det(X) を生成する処理は、文法1の文法規則(2)によると、いずれ np を作り上げるための処理である。したがって、np を予測しているデータが入力引数 X の中に含まれていないならば、det(X) というデータを生成するのは無駄である。np を予測しているデータというのは、例えば、verb(_) は np を予測している。なぜなら、文法規則(7)により、このデータを処理できるのが np だからである。また、np を左端の子として持ち得る文法カテゴリを予測しているデータが X に含まれていてもよい。このようなデータも np を予測しているものと考えことにしよう。例えば、begin というデータは文頭の単語に与えられるデータであるからもちろん sentence を予測していると仮定されるが、これはまた、np をも予測している(文法規則(1)からわか

る。また、同様に文法規則(2)により、begin は det も予測している)。

このように、新しいデータを生成するプロセスがどのようなプロセスを作るためにそのデータを出力しようとしているかということと、新しくできるであろうプロセスを予測しているデータがどのようなものであるかということは、文法規則が与えられた時点で事前に計算しておくことが可能である。

細かい処理手順は省略するが、実際のシステムではトップダウン予測をより積極的に利用し、データを発生するプロセスに対してその入力データの中からそのプロセスがいずれ作ろうとしている文法カテゴリを予測しているデータのみを取り出すフィルタを割り当てている。フィルタは他のプロセスと並列に動き、もしフィルタの出力が空ならば、そのフィルタが割り当てられたプロセスはデータを出力しないようにする。ただし、データの出力はフィルタの計算結果を最後まで待つ必要はなくて、フィルタを抜け出して来るデータが一つであれば、プロセスは隣へデータの送り出しを行ってもよい。

チョムスキー標準形以外の文脈自由文法について

右辺に三つ以上の要素を含む文法規則がある場合には、プロセスが送受するデータに一工夫が必要になる。つまり、

a --> b, c, d,

のような文法規則があるときに、文法カテゴリ b は自分の名前を送るだけでよいが、それを受け取ったプロセス c は自分の名前を送るのでは情報が不完全である。このため実際の構文解析システムの実現においては、まさにこの文法規則の c の役割を実現するため、この文法規則の c と d の間の位置を指示する特別な識別子(例えば、idl などという固有の項)を割り当て、この文法規則内の c の役割を次のPARLOG節により実現している。

c2([b(X)|Tail],[idl(X)|Y]) :- c2(Tail,Y).

また、これに伴い d に関するプログラムにも次のPARLOG節を追加する。

d2([idl(X)|Tail],Y) :-
a(X,Y1),d2(Tail,Y2),merge(Y1,Y2,Y).

同様に、三つ以上の要素を持つ文法規則の右辺の二番目以降に現れる文法カテゴリの間にこのような固有の識別子が与えられ、上のようなプログラムが定義される。詳細については省略するが、このような考え方で

最後に一つだけ注意しておく。実際のシステムでは、文法規則が(制限された) Definite Clause Grammar (DCG) [Pereira 80] で記述されることを仮定している。DCGにおいては、例えば、

のように、文法規則中に Prolog のプログラム (この例の場合 prog1, prog2) を挿入することができる。(また、ここでは省略したが文法カテゴリが引数をもつことができる。)そして、この例のように、同じ *b* であっても、prog1 と prog2 の計算の結果が異なるかも知れないので、*b* が得られた場合に単純に *b(X)* というデータを出力するだけでは事が済まない。実際のシステムでは、右辺の先頭の文法カテゴリにも固有の識別子を割り当て、それを出力させるようにしている。

【Fanty 85】によるコネクショニストモデルに基づく並列構文解析について説明する。彼の考え方は基本的に文脈自由文法に対するCYK (Cook-Younger-Kasami) アルゴリズムに従っている。CYKアルゴリズムというのは、文脈自由文法が与えられたときに図7のような表を作ることによって構文解析を行う方法である。この図は、図の右上に書かれた五つの文法規則を用いて入力系列 aabbb を解析したときに得られる表である。この表は、系列の開始点 i 、系列の長さ j の箱（以後 (i, j) と呼ぶ）の中に非終端記号 A があるとき、入力系列の第 i 番目から第 $i + j - 1$ 番目の終端記号からなる部分系列によって非終端記号 A が構成できることを表わしている。 $(1, 5)$ の箱に S があるのはこの入力系列が正しい文であったことを意味している。

Fanty の考え方は、このアルゴリズムに極めて忠実にネットワークを構成することである。ネットワークは基本的な次の3種類の構成要素および特殊な構成要素1種類によって作られる。

-
- Figure 1: A diagram showing the construction of a sequence S from a sequence A . The diagram is a grid where the vertical axis is "Series Length" (1 to 5) and the horizontal axis is "Series Starting Point" (1 to 5). The grid contains symbols: a and A in the first row, S in the second and third rows, and B in the third, fourth, and fifth rows. To the right of the grid, a set of rules is listed: $S \rightarrow AB$, $A \rightarrow aA$, $A \rightarrow a$, $B \rightarrow bB$, and $B \rightarrow b$.

け terminal unit が置かれる。入力文の第 i 番目の入力記号が a であるとき、 $(i, 1)$ の箱の a に対応する terminal unit が発火される (be primed)。

(2) nonterminal unit: 表のすべての箱に対応する位置に非終端記号種類の数だけ nonterminal unit が置かれる。各 nonterminal unit は、対応する非終端記号をもっている。nonterminal unit は、次の matching unit によって発火される。

(3) matching unit: 各 nonterminal unit に対応して、その非終端記号を左辺にもつ文法規則とその規則を用いることが可能な位置の組み合わせに対して matching unit が置かれる。例えば、 $A \rightarrow BC$ という文法規則があるとき、 (i, j) の位置の A に対応する nonterminal unit に対し、 $(i, 1)$ と $(i+1, j-1)$ 、 $(i, 2)$ と $(i+2, j-2) \dots (i, j-1)$ と $(i+j-1, 1)$ の $j-1$ 通りのそれぞれの組み合わせに対して matching unit が置かれる。つまり、一つの matching unit は一つの文法規則の特定の位置での使われ方に対応しており、文法規則の右辺に対応する二つの nonterminal unit が発火しているとき、左辺に対応する nonterminal unit を発火させる。

(4) \$ unit: 文末をしめすための構成要素である terminal unit と同様に、系列の長さ 1 の列の一つずつ置かれる。 $(i+1, 1)$ の箱の \$ unit は、 $(1, i)$ の箱の非終端記号 S (文を表わす非終端記号) につながっている。

-5-

ば、 $1 \leq i \leq n$ の各 i について $(i, 1)$ の位置の入力文の第 i 番目の終端記号に対応する terminal unit を発火させる。また、 $(n+1, 1)$ の位置の \$ unit を発火させる。matching unit によって次々に nonterminal unit が発火して行く。 $(1, n)$ の位置の非終端記号 S に対応する nonterminal unit が発火することが、入力文が正しい文であったことに相当する。

以上の装置によって入力文が正しい文かどうかは決定できるが、実用的には、構文解析木などの構文解析の結果を得る必要がある。そのために、上で述べたユニット間の結合はすべて双方向になっている。また、各ユニットは、二種類の発火状態を持つ。

入力文からのユニットの発火動作をボトムアップ処理と呼ぶとしよう。ボトムアップ処理中に発火したユニットは、活性化状態 (active) になる。 $(1, n)$ の位置にある文記号 S が活性化されると、 $(n+1, 1)$ の位置の \$ unit によってそれは第二の状態である過活性化状態 (hyperactive) になる。これからがトップダウン処理である。過活性化状態の nonterminal unit は、それに結合している matching unit に自分が過活性であることを知らせる。matching unit が既に活性状態ならば、それは過活性化状態になり、さらに自分の下につながっている nonterminal もしくは terminal unit に自分が過活性であることを伝える。この信号を受け取ったユニットは自分が活性である場合に限り過活性化状態になる。このようにして、 S を作るために貢献していたすべてのユニットが過活性化状態になり、ここから構文解析に関する情報を得ることができる。

アルゴリズムおよびネットワークの構成が非常に簡単であるという特徴を持つ反面、この方法にはいくつかの問題点がある。

一つは、入力文の長さの上限がネットワークの大きさによって制限されてしまうことである。図7のような表に対応するネットワークを事前に作っておく訳であるから、用意しておいた表の横幅以上の長さの文はこのネットワークでは解析することができない。

二つめは、解析可能な文の長さを n とすると、ネットワークには n の3乗に比例する数のユニットが必要なことである。ただし、これは文法規則がチョムスキー標準形である場合のことで、文法規則の右辺に現れる要素の数の最大値が m とすると、ネットワークには n の $m+1$ 乗に比例するユニットが必要である。しかし、本当に問題なのは、必要なユニットの数のオーダーではなくその係数の大きさである。例えば、入力文の長さの上限を15、非終端記号の数を10、同一非終端記号を左辺に持つ文法規則の数が5つずつと仮定すると、matching unit の数は、文法がチョムスキー標準形のときは34000個、文法規則の右辺の要素の数がす

べて3とすると153750個必要になる。著者も述べているように、実際にはすべての箱の位置に nonterminal unit が必要な訳ではないので、この数の matching unit がすべて必要とは限らない。例えば、前置詞句を作るのに少なくとも三単語必要であることがわかっているならば、表の第二行までには前置詞句に対応する nonterminal unit が不必要であるし、そうすると matching unit の数も少なくて済む。しかし、入力文の上限を増やすに従って、このために不必要になるユニットの割合は減少すると考えられる。

また、これはCYKアルゴリズム自体がそうなのであるが、この方法では、前節の並列構文解析システムのようなトップダウン予測を解析に活かすことができない。

4. Massive Parallel Parsing

本章では、[Waltz & Pollack 85] のコネクションストモデルに基づく自然言語文の並列解析を紹介する。まえがきでも述べたように、彼らの仕事は構文解析というよりも、それに伴う曖昧性の解消をコネクションストモデルによって説明したことに意味がある。

図8が、彼らのアイデアを示す例である。

John shot some bucks.

という文を解析するときの初期状態が(a)である。入力文の各単語はいくつかの品詞に属し、いくつかの異なる意味概念を持つ。互いに相反する概念が抑制リンク（両端に小さな円を持つリンク）で結ばれ、両立する概念間は活性化リンク（両端が矢印のリンク）によって結ばれている。ある程度の時間経過後のネットワークの状態が(b)に示されている。色の濃いものがより活性化されている概念である。HUNTING という文脈によって、shot の意味として FIRE という動詞概念が選ばれていることがわかる。

彼らの実験では、構文解析にはチャートパーザが用いられ、構文構造上で曖昧性があるときには、構文構造の親子関係のノードには活性化リンクが張られ、同じ子を支配する姻戚関係のノードの間には抑制リンクが張られる。図9にそのような例を示す。最初、(a)のように木の深さが低い解析が優先されるが、意味的な制約が働くと、(b)の解釈が有効になる。

構文解析を自然言語文の解析の過程においてどのようなレベルで取り扱うかについては様々な議論がある。文解析のための指針 (guideline) として文の構文構造を用いる考え方もあるが、文に曖昧性が生じるのは構文処理機構の副作用であり、解析の指針となる原理は「曖昧性を解消しようという普遍的な意識 (universal will to disambiguate)」であるとする彼らの取っている態度は非常に興味深い。

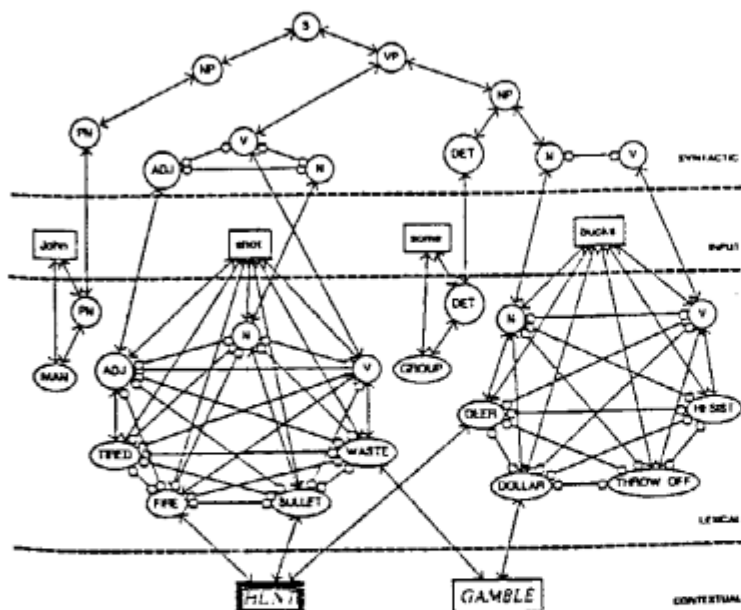


図8(a) 初期状態

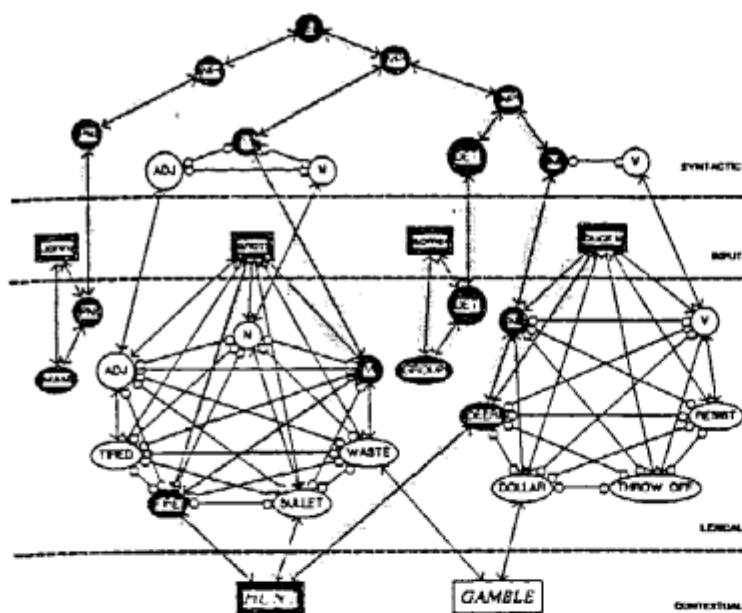


図8(b) 50サイクル後の状態

ここでは省略するが、

The astronomer married a star.

という文の解釈において、star の意味として、最初、astronomer の意味によって「星」という意味が優っていたのに、その後、married が持つ意味的な制約によって、次第に、「映画スター」という意味に活性化

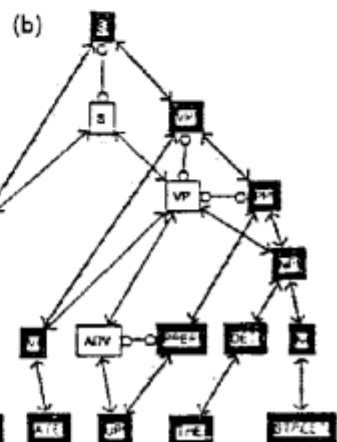
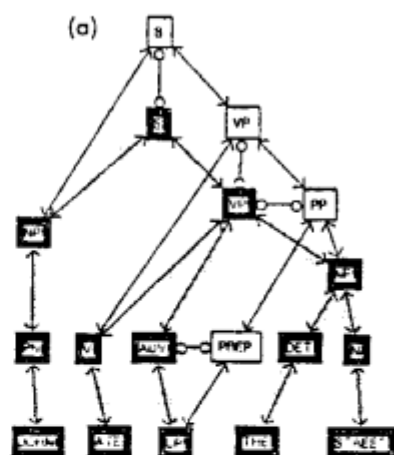


図9. "John ate up the street"の解析

が以降していく例が示されており、この例も非常に興味深い。

5. あとがき

並列構文解析について、いくつかのアプローチを紹介した。それぞれ立場や特徴が異なるので単純に比較することはできない。

最初に示した著者らの並列構文解析法は、並列論理型言語という並列環境に載せるという考えのもので、そういう意味では汎用の環境での実現を目指している。

別のアプローチとしては、専用のマシンなり、環境なりを設定した方法がある。後半のコネクションストモデルによるアプローチはこちらの種類に属する。その中でも二つの比較的両極端なアプローチを紹介した。

Fanly の方法は、構文解析専用の結合マシンを設計しようというもので、コネクションストといっても、人間の行っている構文解析過程とは程遠い。

一方、Waltz らのアプローチは、人間の文理解をあ

る程度模倣したアプローチであり、いかにもコネクションリストモデルという名にふさわしい。また、彼らは個々の概念をさらに細かいプリミティブに分け、概念間の両立や抑制関係の記述に用いており、これによってより広範囲の文脈情報を利用することを実験している。

【参考文献】

- [Aho 72] Aho, A.V.: The Theory of Parsing, Translation, and Compiling, Vol.1 Parsing, Prentice-Hall, 1972.
- [Earley 70] Earley, J., "An Efficient Context-free Parsing Algorithm," C.ACM, Vol.13, 1970.
- [Fanty 85] Fanty, M., "Context-Free Parsing in Connectionist Networks," TR174, Department of Computer Science, University of Rochester, Nov. 1985.
- [Kay 80] Kay, M., "Algorithm Schemata and Data Structures in Syntactic Processing," XEROX PARC, CSL-80-12, Oct. 1980.
- [Matsumoto 86] Matsumoto, Y., "A Parallel Parsing System for Natural Language Analysis," Proc. 3rd ICLP, London, 1986.
- [松本 86] 松本裕治、「論理型言語に基づく構文解析システム S A X」、コンピュータソフトウェア、Vol.3, No.4, pp.308-315, Oct. 1986.
- [Matsumoto 87] Matsumoto, Y., "Parsing Gapping Grammars in Parallel," to appear as ICOT Technical Memo, 1987.
- [Pereira 80] Pereira, F. and Warren, D., "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks," Artificial Intelligence, Vol.13, pp.231-278, 1980.
- [Rytter 85] Rytter, W., "Parallel Time $O(\log N)$ Recognition of Unambiguous CFLs," Lecture Notes in Computer Science 199, pp.380-389, 1985.
- [杉村 86] 杉村領一、松本裕治、「構文解析システム S A X の C I L による実現」、情報処理学会全国大会、Oct. 1986.
- [Waltz & Pollack 85] Waltz, D.L. and Pollack, J.B., "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation," Cognitive Science, Vol.9, No.1, pp.51-74, 1985.