数理解析研究所講究録
591巻 1986年 237-246

並列構文解析法とその評価

（財）新世代コンピュータ技術開発機構
松本裕治（Yuji Matsumoto）

1．はじめに

本稿の目的は、現在開発中の自然言語処理用の並列構文解析システムＡＸ（Analyser for natural language syntaX)の評価を行なうことである。このシステムは、Parlog［Clark 84］、Concurrent Prolog［Shapiro 83］、ＧＨＣ（Guarded Horn Clauses）［Ueda 85］などの並列論理型言語上での実現を目指した構文解析システムである。このシステムは、構文上は、文脈自由文法を扱うが、論理型言語より派生した自然言語用の文法記述形式であるＤＣＧ（Definite Clause Grammars）［Pereira 80］で書かれた文法を処理することができるため、実際には、文脈自由文法以上の文法をも取り扱うことが可能である。

本稿では、構文解析アルゴリズムそのものの評価を行なうため、文法を文脈自由文法に限って、その計算量の複雑さについて、考察する。

まず、最初に、アルゴリズムの記述を行ない、並列計算での複雑さを評価する。次に、アルゴリズムの並列論理型言語による実現法を示す。その結果、本手法は、入力文の長さに対して

線形時間の並列構文解析プログラムであり、しかも、同一計算を繰り返さない最適アルゴリズムであることが分かる。

また、本アルゴリズムは、逐次処理マシンにとっても効率よく、Prologによって実用的な構文解析システムとして、利用することができる。アルゴリズム自体は、ＡＸと呼ばれるが、Prologで逐次マシン上に実現されたシステムをＳＡＸ（Sequential AX）、ＧＨＣ等の並列型言語によって実現されたシステムをＰＡＸ（Parallel AX）と呼ぶ。

２．アルゴリズムの記述

並列アルゴリズムの記述には、並列計算モデルが必要である。本節では、我々のアルゴリズムの記述に、Ｗ－ＲＡＭ（Write-enabled Parallel Random Access Machine）を用いる。これは、自然数によって、インデクシングされた、共有記憶を持ち、各記憶セルに対して、同時読み出し、および、同時書き込みが許されている並列ＲＡＭである。ただし、同時書き込みが行なわれる場合には、書き込まれる内容が同じでなければならない。

文脈自由文法を次のように定義しよう。$G=(V_N, V_T, P, S)$。ここに、V_Nは非終端記号の有限集合、V_Tは終端記号の有限集合、Ｐは文法記号の有限集合、Ｓは初期記号を表わす非終端記号である。以下、入力文字列を、$a_1a_2\cdots a_n$であらわす。また、一般に、非終端記号を大文字のアルファベットで、終端記号を小文字のアルファベットで、非終端記号と終端記号よりなる記号列をギリシャ文字で表わす。また、文法規則の右辺に現れる任意の連続する二つの記号の間に識別子として、

個別に自然数を割り当て、そのような自然数の集合をＫとする。例えば、Ａ→ＢＣＤ、Ｅ→ＦＧをＰに含まれる文法規則とすると、これには、Ａ→Ｂ　１　Ｃ　２　Ｄ、Ｅ→Ｆ　３　Ｇのように識別用の自然数が割り当てられているとする。

準備として、次の二種類の集合を定義する。これらはアルゴリズムでは、表のような構造をもつ記憶領域として扱われる。一つは、

$$S = \{ (A, i, j) : A \in V_N,\ 0 \leqq i < j \leqq n \}$$

なる集合。もう一つは、$1 \leqq j \leqq n-1$なるｊに対して、

$$T_j = \{ (k, i) : k \in K,\ 0 \leqq i < n \}$$

として定義される一連の集合である。

Ｓに含まれる要素（A,i,j）の意味は、非終端記号Ａが、入力文字列の部分列$a_{i+1} \cdots a_j$を葉（leaves）とする解析木の根（root）になり得ることである。T_j内の要素（k,i）の意味は、ｋが、文法規則$A \rightarrow \alpha\ k\ \beta$に含まれる識別子とすると、この規則の右辺でｋによって区切られた箇所の直前までの記号列、すなわちαが、入力文字列の部分列$a_{i+1} \cdots a_j$によって完成されたということである。これらの集合は各要素を項目として持つ表として取り扱われ、全ての要素は初期値 'false' を持つものとする。

さて、文脈自由文法についての並列構文解析のアルゴリズムを示そう。

アルゴリズムＡＸ

begin

```
1: for each 0≦i<n, A∈V_N such that A→a_{i+1}
     parallel do  (A,i,i+1):=true ;
repeat N times
2: for each (B,i,j)=true and A→B k α
     parallel do  (k,i):=true in Tj ;
2':for each (B,i,j)=true and A→B
     parallel do  (A,i,j):=true ;
3: for each (B,j,m)=true and (k,i)=true in Tj
                  such that A→α k B
     parallel do  (A,i,m):=true ;
3':for each (B,j,m)=true and (k,i)=true in Tj
                  such that A→α k B k+1 β
     parallel do  (k+1,i):=true in Tm
end repeat ;
if (S,0,n)=true then ACCEPT
end ;
```

２および３，３'によって、文法規則の右辺に含まれる記号が順次処理されていくので、文法規則の右辺に現れる記号数の最大値をｃとすると、アルゴリズムの繰り返し回数Ｎは、(c-1)n 回でよく、このアルゴリズムはo(n) の線形アルゴリズムとなる。本アルゴリズムは、チャートパージング［Kay 80］を上昇型に並列に行なうのと同等で、各部分木に対して、文法規則の適用は、重複しては行なわれないから、同一処理の繰り返しは避けられている。

3．並列論理型言語による実現

本節では、上で述べたアルゴリズムを並列論理型プログラミング言語で実現する方法を示す。記述言語としては、ＧＨＣを用いる。

並列論理型言語によるプログラミングは、ストリームを介して通信し合う並列プロセスの記述を基本とする。そこで、(A,i,j) なる項をプロセスとして論理型言語では述語として定義し、Ｔｊをプロセス間を流れるストリームとして取り扱うことを考える。入力文字列 $a_1 a_2 \cdots a_n$ に対して、初期実行を行なう呼び出しを

$a_1(T0,T1), a_2(T1,T2), \cdots, a_n(Tn-1,Tn).$

と定義する。これは、ＧＨＣでａｎｄ並列に動くプロセスである。ここに、T0、Tn は、システムの使用者によって定義され、使用される、定数またはストリームで、その使い方についてはここでは省略する。

前節で示したアルゴリズムの各ステップは、次のようなＧＨＣ節に一対一に変換することができる。ただし、論理型言語の慣習により、述語名は英小文字で、変数は英大文字を先頭に持つ項として記述されている。

```
1:  a_i(Ti-1,Ti) :- a(Ti-1,Ti).   (for each  A→a_i ∈ P)
2:  b(Ti,Tj) :- Tj=[(k,Ti)].   (for each  A→B k α ∈ P)
2': b(Ti,Tj) :- a(Ti,Tj).   (for each  A→B ∈ P)
3:  b([(k,Ti)|Tj],Tm) :- a(Ti,Tm'),b(Tj,Tm")
                          merge(Tm',Tm",Tm).
```

(for each A→ α k B∈ P)

3': b([(k,Ti)|Tj'],Tm) :- Tm=[(k+1,Ti)|Tm'],b(Tj',Tm').

(for each A→ α k B k+1 β ∈ P)

このようにして定義されたそれぞれのＧＨＣ節は、独立に動くことができ、同一名をもつ述語の出力（第２引数）は、マージされて、次のプロセスへ渡すことのよってこれらをａｎｄ並列に実行することが可能である。これらのＧＨＣ節の定義を見て分かるように、文法規則の右辺に現れる記号（終端記号、非終端記号を問わず）一つずつに対して、対応するＧＨＣ節が生成される。同一名をもつ述語は、全てａｎｄ並列に動くプロセスとして定義できる。ただし、３と３'は、互いに排他的であるので、実際には、これらの節は、ｏｒ並列に実行することも可能であり、現システムでは、そのように定義されている［松本 86］。具体的な文法規則について、ＧＨＣ節への変換の例を付録に示す。

４．論理型言語による実現についての問題点

本節では、ＡＸの実現に関して述べておかねばならない問題点について考察する。

同時書き込みについての問題は、論理型言語による実現では部分解析木がプロセスとして定義されているため、同一プロセスの生成の問題に置き換わっている。資源が無尽蔵と考えてよいならば、これは計算時間には影響を与えない。ただし、現実には、プロセッサ数は有限と考えねばならないので、その場合には、同一プロセスの生成を阻止するような、メタな機構を考

えねばならない。ただし、本システムが単なる文脈自由文法ではなく、ＤＣＧで書かれた自然言語文法を扱うことを考えると、構文解析だけではなく、それに伴う意味解析等も考慮に入れなければならない。従って、入力文の同一部分列からたとえ同じ非終端記号を根とする解析木が生成されようとも、文法規則に同一のものが含まれていない限り、それらの内容はどこかが違うのであるから、同じ非終端記号が構成されてしまっても構わないという考え方もある。この問題は、資源の有効利用と重複処理のオーバーヘッドとが絡み、今後解決しなければならない問題である。

また、いろいろなプロセスから生み出される出力が、ストリームによって、一本にまとめられてしまうため、並列性がある程度失われてしまうように見えるが、これは、ストリームを並列論理型言語の中でどのように実現するかに依存する問題であり、ストリームがランダムアクセス可能な記憶構造として実現されれば、効率に対する影響は軽減される。この問題に関してだけでなく、ストリームに関する処理の効率は並列論理型言語にとって極めて重要な問題であることを指摘しておきたい。

本アルゴリズムによる文の解析は、完全に上昇型（bottom-up）に行なわれるが、下降型（top-down）の戦略も部分的に利用することができる。実際のシステムでは、上からの予測を、ストリームの中から無駄な要素を取り除くフィルターのようなプロセスとして実現されている。

５．結論と考察

文脈自由文法に対する並列構文解析アルゴリズムとして、ＡＸは、線形時間のアルゴリズムであることが分かった。文脈自由文法の認識についての並列アルゴリズムとしては、他に、時間的により優れたものが提案されている。Alternating Turing Machine［Chandra 81］やＰ－ＲＡＭ（Parallel Random Access Machine）［Fortune 78］では、$o(\log^2 n)$のアルゴリズムが報告されている［Ruzzo 80］［Rytter 84］。また、Ｗ－ＲＡＭでは、それらのアルゴリズムを $o(\log n)$でシミュレートできる。

しかし、これらのアルゴリズムでは、無駄なプロセスが相当作られるので、現実の構文解析システムとして使うには難がある。

一方、我々のアルゴリズムでは、無駄なプロセスやデータの生成は、極力おさえられている。逐次的実行を行なった場合でも、その処理の複雑さは、チャートパーザと同等もしくはそれ以下であり、データやプロセスの生成量は最適化されている。このシステムは、Prolog 上でもインプリメントされ、自然言語処理用の構文解析システムとして利用される予定である。

今後の課題としては、無駄な計算を抑えながら、如何に計算時間を短縮するかということと、同一処理を防ぐためのメタな機能をどのように自然に取り込むかを考えなければならない。

【参考文献】

［Chandra 81］ Chandra,A.K., et al., "Alternation,"
JACM, vol.28, no.1, pp.114-133, 1981.

[Clark 84] Clark,K. and Gregory,S., "PARLOG: Parallel Programming in Logic," Imperial College Research Report DOC 81/16, July 1984.

[Kay 80] Kay,M., "Algorithm Schemata and Data Structures in Syntactic Processing," XEROX PARC, CSL-80-12, Oct. 1980.

[Fortune 78] Fortune,S. and Wyllie,J., "Parallelism in Random Access Machines," Proc. 10th ACM Symposium on Theory of Computation, pp.114-118, 1978.

[Pereira 80] Pereira,F. and Warren,D., "Definite Clause Grammars for Language Analysis," Artificial Intelligence 13, pp.231-278, 1980.

[Ruzzo 80] Ruzzo,W.L.,"Tree-size Bounded Alternation" Journal of computer and System Sciences 21, pp.218-235, 1980.

[Rytter 85] Rytter,W., "Parallel Time o(log N) Recognition of Unambiguous CFLs," Lecture Notes in Computer Science 199, pp.380-389, 1985.

[Shapiro 83] Shapiro,E.,"A Subset of Concurrent Prolog and its Interpreter," Technical Report TR-003,ICOT, Feb.1983.

[Ueda 85] Ueda,K.,"Guarded Horn Clauses," Proc. The Logic Programming Conference '85, ICOT, pp.225-236, July 1985.

［松本 86］松本裕治、「並列構文解析」、情報処理学会
自然言語処理研究会、1986年1月

【付録】

文脈自由文法からＧＨＣ節への変換の例を示す。（１）で示すのは、文法規則の一部であり、（２）が、変換後のプログラムの内 noun に関するＧＨＣ節だけを取り出したものである。ただし、文法規則中の右辺の非終端記号間に書かれている自然数は、識別子である。

（１）
```
NP → DET  1  NOUN
NP → DET  2  NOUN  3  REL_CLAUSE
NP → NOUN
NOUN → NOUN  4  PP
NOUN → NOUN  5  PRED
```

（２）
```
noun(X,Y) :- noun1(X,Y1),noun2(X,Y2),
                                merge(Y1,Y2,Y).
noun1(X,Y) :- Y=[(4,X),(5,X)|Y1],np(X,Y1).
noun2([],Y) :- Y=[].
noun2([(1,X)|X1],Y) :- np(X,Y1),noun2(X1,Y2),
                                merge(Y1,Y2,Y).
noun2([(2,X)|X1],Y) :- Y=[(3,X)|Y1],noun2(X1,Y1).
noun2([_|X],Y) :- otherwise | noun2(X,Y).
```