逐次処理系上でのＬＡＸの問題点とその解決法

3T-6

赤坂宏二、杉村領一、松本裕治

（財）新世代コンピュータ技術開発機構

0．はじめに

LAX[1],SAX[2]は，ともに並列論理型言語上での実行を目的とした自然言語の解析手法であり将来は，並列マシンの上でパラレルに実行されることが期待されている．SAXに関しては逐次処理系の上でも高速に動作することが確認されており[3]我々はPrologや ESP[4]上での自然言語解析システムとして実験を進めている．本稿では LAX とSAXについて簡単に紹介した後，この二つを逐次処理系上で実行させる際の問題点とその解決案について報告する．

1．ＬＡＸとＳＡＸ

1.1 LAX

LAXはGHC[5]等並列論理型言語の上で動作するように提案された形態素解析手法であり，基本的にレイヤードストリーム[6]を用いたプログラミングの応用といえる．LAXでは与えられた形態素辞書は，形態素解析を行うプログラムに変換される．以下，簡単にLAXでの形態素解析について説明する．まず，入力文の各文字ごとに一つのプロセスを対応させて起動する．おのおののプロセスは，前方からの結果を受け取るための入力ストリームと後方へ解析結果を渡すための出力ストリームをもつ．個々のプロセスの役割は（a）自分が割り当てられた文字以降の文字列を検索する．(b) 検索に成功した場合，前方からの入力の内その語と連接可能なものをプロローグリストで一纏めにしたものを尾部とし，その語を頭部とするリストにして出力ストリームに流し，次段へ送ることである．先頭のプロセスにはbeginというデータをあたえる．最後の文字に割り当てられたプロセスの出力が入力文字列全体の形態素解析結果となる．入力文字列”くるまで待つ”に対する解析結果を図1に示す．このようにLAXの出力はその語の表層を頭部，その語の前方の構造を表す構造を尾部とする再帰的な構造になっている．

```
[待つ,
 [まで,
   [来る,begin],[繰る,begin] ],
 [で,
   [間,
     [来る,begin],[繰る,begin] ],
   [車,bigin]]]s
```

図1

1.2 SAX

SAXは，DCGで記述された文法をPrologの構文解析プログラムに変換する手法である．変換されたプログラムは決定的に動作しバックトラックなしに全解探索を行なう．

2．両者の結合

ここではLAXの出力をSAXの入力として使うことを考える．

2.1 構造の重複

図1の形態素解析結果は図2のような，文末の形態素を根とする木構造（以下，形態素解析木）になっており根から葉までの各パスが形態素解析候補となっている．

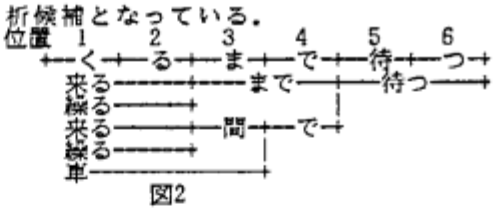

図2

しかしこれは計算機上では図3のような縮退したデータであり，表現上展開されているにすぎない．前方のプロセスからの解析結果が自プロセスの検索語の複数のものに接続可能なとき，このようなデータの重複が発生する．この例では「繰る」，「来る」が共に「まで」，「間」の両方に接続可能である．

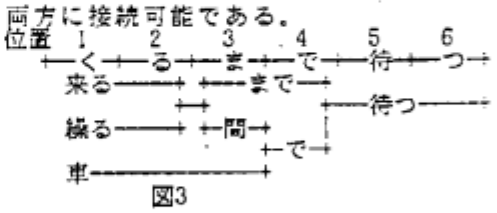

図3

2.2 SAXの起動

SAXでは 例えば"every man walks."のような単語列からなる入力文を解析するために次のようなゴール列を呼ぶ。

every([begin],S0),man(S0,S1),walks(S1,S2).

LAXの場合と同じく,それぞれのゴールは入力ストリーム,出力ストリームと呼ばれる二つの引数を持ち,前からの出力を受け取りそれを処理し後ろに流すというやりかたで解析が行われる。従って,LAXの出力をSAXの入力とするためには先程示した,図のような形態素解析木を後ろからたどって,このようなストリームで結合されたゴール列を生成せねばならない。

3．逐次処理系上での問題点

ここで先程のLAXの出力結果のデータの重複が問題になる。図2の木を後からたどってゴール列を生成した場合，SAXでの処理に重複が生じる。（「来る」，「繰る」の部分)このような重複は,入力文が長くなり解析結果が複雑になると構文解析時間に与える影響が大きい。

3.1 解決案

そこで次のようないくつかの解決案を検討した。

(案1)

形態素解析結果の木をたどる際,Prologの組込述語assert をつかってゴールを記録していき,重複を防ぐ。

(案2)

LAXの出力データ構造を図4のように変更し形態素の連接をプロローグの論理変数を使って表す。図4で「来る」は変数Bを通じて「まで」，「間」に連接する。この変数はSAX のストリームとしてそのまま使う。

```
{
  {[begin],苦,A},
  {[begin],句,B},
  {[begin],繰る,C},
  {[begin],来る,D},
  {[begin],車,E},
  {{C,D},まで,F},
  {{C,D},間,G},
  {{E,G},で,H},
  {{F,H},待つ,J}
}.
```

図4

〔問題点〕

文末まで到達しないものもSAXの入力としてしまう可能性があり無駄な処理が生じうる。

(例: 図4の「苦」，「句」)

(案3)

まず,形態素解析時に後方へ送るデータに論理変数を2つ付与する。すると,形態素解析結果は図5のようになる。「まで」に前接している「来る」も,「間」に前接している「来る」も内部のデータ構造としては,共有された同一の構造であり,同一の変数を持つ。その変数のうちの一つを「ストリーム変数」と呼び,他方を「フラグ変数」と呼ぶことにする。「ストリーム変数」はSAXでの構文解析時に出力ストリームとして使用する。子の節点の「ストリーム変数」をマージしたものが親の節点に対応したゴールの入力となる。そして形態素解析木をたどる際,一度たどった節の「フラグ変数」をバインドしていく。そして「フラグ変数」が既にバインドされている構造からはゴールを生成しない。このようにゴール列を生成していくことにより副作用を使わずに重複なくゴール列を生成し構文解析において無駄な処理を行なわないですむ。現在,我々はこの方向で実験を進めている。

```
〔待つ,FA,SA,
 〔まで,FB,SB,
   〔来る,FC,SC,begin〕,
   〔繰る,FD,SD,begin〕〕,
 〔で,FE,SE,
   〔間,FF,SF,
   〔来る,FC,SC,begin〕,
   〔繰る,FD,SD,begin〕〕,
   〔車,FG,SG,bigin〕〕〕
```

図5

3．おわりに

本稿では形態素解析システムLAXと構文解析システムSAXを逐次処理系上で実行させる場合の問題点について述べた。今後更に実験を重ね効率の良い方法について検討していく予定である。

〔参考文献〕

〔1〕:杉村ほか,並列形態素解析 ソフトウニア科学会 1986

〔2〕:松本ほか,a parsing system on logic programming IJCAI 87

〔3〕:奥西ほか,Comparison of Logic Programing Based Natural Language Parsing System, Proceeding of 2nd International Workshop of Natural Language and Logic Programing

〔4〕:近山,ESP reference Manual,ICOT TR-044

〔5〕:上田,Gurded Horn Clause,ICOT TR-103

〔6〕:奥村ほか,レイヤードストリームを用いた並列プログラミング,Logic Programming Conference '87