並列形態素解析システムＬＡＸの実現

1T-5

杉村領一、赤坂浩二、松本裕治

（財）新世代コンピュータ技術開発機構

１．はじめに

並列実行可能な日本語の形態素解析手法[6]についてのべる。本手法では、形態素辞書をＧＨＣ[7]等の並列論理型言語で記述されたプログラムへ変換する。形態素辞書には、形態素の①見出し、②カテゴリー、形態素間の連接可能性を調べるための③接続素性情報[1]、を記述する。接続素性情報は、④接続マトリクス、と共に曖昧さ削減に用いる。解釈に曖昧さがあれば、解析終了時に結果を縮退して一度に出力する。解析結果は、構文解析プログラムＳＡＸ[5]への入力に用いる。本方式は未登録語処理をリニアオーダで行なう。また、逐次論理型言語上でも、高速な解析が可能である。

２．形態素解析アルゴリズム

本アルゴリズムはEarleyのアルゴリズム[2]や、チャートパージング[4]などで用いられているアルゴリズムに基づいている。

以下、文字列検索プロセスの起動方法とプロセス間の通信方式について、その概要を述べる。

２．１プロセスの起動

以下図１を例に説明を行なう。

入力：　く　る　ま　で　待　つ　。

	1	2	3	4	5	6	7
F 1	dt	p13	p14				
	F 2	dt					
		F 3	dt	p35			
			F 4	dt			
				F 5	dt		
					F 6	dt	
						F 7	dt

−図１−

'dt'は初期状態で起動される検索プロセスで、入力文字各々の位置に対応して起動される。初期状態で、隣接する'dt'はストリームによって一次元に連結される。

'dt'は対応した文字を先頭とする部分文字列についての形態素の検索を次の様に行う。

まず'dt'は対応する１文字を形態素辞書から検索する。これと共に、対応する１文字に始まる長さ２以上の形態素を検索するサブプロセスを起動する。例えば図１で行Ｆ１の'dt'は「く」を検索すると共に、「くる」を検索する'p13'と「くるま」を検索する'p14'を起動する。起動された検索処理は全て並列に実行が可能である。この処理は次節に説明するように、図４のようなＰｒｏｌｏｇまたは、ＧＨＣのプログラムによって実現されている。

'dt'とそのサブプロセスの出力は、d-listによって連結され、１本のストリームとしてまとめられる。

以上の方法により、'dt' の対応する文字を先頭とする部分文字列の検索結果は、'dt'の出力として、その右隣のプロセスへ渡る。

２．２．形態素辞書の検索プロセスへの変換

上記の検索プロセス'dt'とそのサブプロセスは以下の方法で形態素辞書から変換して得られる。例えば、図２のような形態素辞書を考える。

見出し	形態素カテゴリ	接続素性
く	カ変動詞語幹	1
くる	わ行動詞語幹	2
くるま	名詞	3
くろ	形容名詞	4

−図２−

上記の辞書はＴＲＩＥ構造[3]を用いて図３の様に表現出来る。

```
[く カ変動詞語幹 1
    [る わ行動詞語幹 2 [ま 名詞 3]]
[ろ 形容名詞 4]]
```

−図３−

更に図３の構造は図４のようなＰｒｏｌｏｇのプログラムに変換される。

Parallel Lexical Analyzer LAX

Kyoichi Sugimura, Koji Akasaka, Yuji Matsumoto
Institute for New Generation Computer Technology

```
dt([く|R],...)   :- く(R,..).
く([る|R],...)   :- くる(R,...).
く([ろ|R],...)   :- くろ(R,...).
くる([ま|R],...) :-.
```

－図４－

図４で、dt([く|R],...) が呼び出され「く」が検索されると、サブプロセスの'く（R,..)'を起動する。これは、「く」という単語が存在したことに相当し、このプロセスは「く」という単語が持つデータを出力すると共に、第２行目の節を呼び出すことによって、「く」から始まる他の単語の検索を引き続き行う。

以上の様に、形態素辞書を予めコンパイルし、検索に用いる事により、効率の良いプロセス呼出しにより処理を進められる。

２．３．プロセス間の通信

プロセス間の通信方法と、検索データの縮退について述べる。

前記２．１で述べたように、プロセス'dt'は対応する文字に始まる部分文字列の検索結果をストリームを通じて右隣の'dt'へ出力する。このため、'dt'から出力されるデータは種々の部分文字列の検索結果からなり、各々のデータが入力文字のどこまでを解析したかに応じて、次のプロセスの検索開始位置を示す検索開始位置番号Ｎｓが必要となり、これを各々のデータは持つ。

行Ｆ１の'dt'には起動時に次のデータを渡す。

```
  Ｎｓ    文頭の接続素性   検索結果
〔１，        １，          〔〕〕
```

行Ｆ１の'dt'が「く」を検索すると、「く」の接続素性と文頭接続素性より「く」が文頭に連接可能か接続行列を用いて'dt' は調べる。可能ならば、'dt'は「く」に続く文字位置２をＮｓとする次のデータを出力する。

```
  Ｎｓ    「く」の接続素性，   検索結果
〔２，        １０，        〔く,ら行,〔〕〕〕
```

'p13' が文字「くる」を検索すると３をＮｓとする次のデータを出力する。

```
  Ｎｓ    「くる」の接続素性，   検索結果
〔３，        １１，        〔くる,わ行,〔〕〕〕
```

行Ｆ１の'dt'及び'p13','p14'の検索結果は、マージされて、行Ｆ１の'dt'から行Ｆ２の 'dt'へ渡される。

行Ｆ２の'dt'は、入力ストリームからＮｓが２のデータを取り出し、「る」より始まる形態素を検索するプロセスに渡す。

「る」より始まる形態素はＦ２の位置の'dt'により生み出されるプロセスによって切り出され、それぞれのプロセスは受け取ったデータの中から接続情報を満足するものだけを取り出す。

こうして取り出されたデータは唯一つに決まるとは限らず、一般に接続情報を満足するデータは複数個存在する事があり得る。このようなデータは集合として表され、例えば図１の例の出力は、

```
〔〔待つ,た行強変化動詞,
      〔〔で,格助詞,    〔車,名詞〕〕,
       〔まで,格助詞,
            〔る,か変語尾
                 〔来,か変動詞語幹〕
```

のようになり、図式化すると、

```
待つ──┬──で────車
      └──まで──る──来
```

のように「待つ」の部分が共有されたような縮退した形の出力が得られる。

３．おわりに

本アルゴリズムは現在Ｐｒｏｌｏｇ上にインプリメントされ、構文解析システムＳＡＸとの結合されている。プロトタイプによる実験で、逐次実行時にも十分な解析速度を持っていることを確認したが、今後更に大掛かりな実験を経て、評価を行ない改良を加える予定である。

【参考文献】
〔1〕 相沢輝昭、江原暉将：計算機によるカナ漢字変換，ＮＨＫ技術研究，25巻，5号，1973
〔2〕 Earley, J.: An Efficient Context-Free Parsing Algorithm,Comm.ACM,Vol13, No.2,1970,pp.94-102.
〔3〕 上脇正，田中穂積：辞書のＴＲＩＥ構造化と熟語処理，Proceedings of the logic programming conference 85',12.2,1985,pp 329-340
〔4〕 Kay,M,: Algorithm Schemata and Data Structures in Syntactic Processing, Technical Report CSL-80-12,XEROX PARC,Oct.1980
〔5〕 松本裕治，杉村領一：論理型言語に基づく構文解析システムＳＡＸ，ソフトウエア科学会論文誌，1986.
〔6〕 杉村領一、松本裕治：並列形態素解析，ソフトウエア科学会，論理と自然言語研究会，1987
〔7〕 Ueda Kazunori: Guarded Horn Clauses, ICOT Technical Report, No.103,1985.