

## 先祖表を利用した一般化LR

## パーシングアルゴリズムのファミリー

田中穂積, K. G. Suresh, 山田耕一  
東京工業大学・工学部

Knuthの考案したLR(k)法とよぶ統語解析アルゴリズムは、文脈自由文法(CFG)のサブセットのLR文法から得られる文に対して、決定的に統語解析を進め、無駄なく効率的に統語解析を行うことができる。富田はLR(k)法の持つ利点をそのまま保持し、しかも一般のCFGが扱えるようにLR(k)法を拡張している。これは一般化LR法(generalized LR parsing algorithm; GLR法)の1つであり、富田法とよばれている。経験的に富田法はアーリー法と比べて高速な統語解析が可能であるが、富田法に対して、アーリー法以上のオーダーの解析時間を要す特殊なCFGの存在が知られている[Johnson 91]。これは、富田法がレデュース操作時にグラフ構造化スタック上の同一パスを何度も繰り返したためである。

最近Kippsはグラフ構造化スタック上の同一パスを繰り返したことを避けるために、先祖節点を記憶する先祖表を用い、解析時間が文の長さ $n$ に対して $n^3$ となるGLRパーシング・アルゴリズムを開発している[Kipps 91]。しかし、Kippsのアルゴリズムは、与えられた文の文法的妥当性を知るための認識アルゴリズムに過ぎず、統語解析木(構文構造)を得ることができないので、実用上問題があるとされている[Schabes 91]。本論文では、Kippsの認識アルゴリズムをベースにして、先祖表から統語解析木を作成するための情報を抽出することができることを示す。この場合、抽出された情報から一つの統語解析木を得るのに要す時間は、アーリー法と同様、 $n^2$ のオーダーである。先祖表の内容に工夫をこらすことにより、富田法で用いる圧縮統語森と同様、解析結果から得られる情報から、一つの統語解析木を抽出する時間のオーダーが $n$ のアルゴリズムを得ることができる。この新しいGLR法の使用する記憶空間量は、富田法と同様 $n^2$ のオーダーである。実験により、これらの一群のGLRパーザは、統語解析結果に含まれる曖昧さが増大するに連れて、高速な統語解析が可能であることを確認している。

## A Family of Generalized LR Parsing Algorithms

## Using Ancestor Tables

Hozumi TANAKA, Katare G. Suresh and Kouiti Yamada  
Department of Computer Science  
Tokyo Institute of Technology  
2-12-1 Oookayama Meguro-ku  
Tokyo 152, Japan

We have developed a family of new generalized LR parsing algorithm which make use of a set of ancestor tables Kipps introduced to his recognition algorithm [Kipps 91]. As Kipps's recognition algorithm does not give us a way to extract any parsing result, his algorithm is not considered as a practical parsing algorithm [Shabe 91]. In this paper, we will show that it is possible to extract each parsing tree from a set of ancestor tables. While the time complexity of Tomita's parsing algorithm can exceed  $O(n^3)$  for some CFG's, the time and space complexity of our parsing algorithm using a set of ancestor tables is in the order of  $n^3$  and  $n^2$  for any CFG, respectively, since our algorithm is based on the Kipps's recognition algorithm. In order to extract a parsing tree from a set of ancestor tables, it takes time on the order of  $n^2$ , where  $n$  is the length of an input sentence to be analyzed. However, by making small modifications in the ancestor table, it is possible to extract all parsing trees in order of  $n$  time. A preliminary experiment seems to suggest our parsing algorithm is very promising.

## 1. はじめに

Knuthの考案したLR(k)法とよぶ統語解析アルゴリズムは、文脈自由文法(CFG)のサブセットのLR文法から得られる文に対して、決定的に統語解析を進め、無駄なく効率的に統語解析を行うことができる。富田はLR(k)法の持つ利点をそのまま保持し、しかも一般のCFGが扱えるようにLR(k)法を拡張している。これは一般化LR法(generalized LR parsing algorithm; GLR法)の1つであり、富田法とよばれている。経験的に富田法はアーク法と比べて高速な統語解析が可能であるが、富田法に対して、アーク法以上のオーダの解析時間を要す特殊なCFGの存在が知られている[Johnson 91]。これは、富田法がレデュース操作時にグラフ構造化スタック上の同一パスを何度も繰り返したるためである。

最近Kippsはグラフ構造化スタック上の同一パスを繰り返したることを避けるために、先祖節点を記憶した先祖表を用い、解析時間が文の長さ $n$ に対して $n^3$ となるGLRパーズング・アルゴリズムを開発している[Kipps 91]。しかし、Kippsのアルゴリズムは、解析文の文法的妥当性を知る認識アルゴリズムに過ぎず、統語解析木(構文構造)を得ることができないので、実用上問題があるとされていた[Schabes 91]。本論文では、Kippsの認識アルゴリズムをベースにして、先祖表から統語解析木を作成するための情報が抽出可能なことを示す。この場合、解析結果から一つの統語解析木を得るのに要す時間は、アーク法と同様 $n^2$ のオーダである。先祖表の内容に工夫をこらすことにより、富田法で用いる圧縮統語森と同様、解析結果から、一つの統語解析木を抽出する時間のオーダが $n$ のアルゴリズムを得ることができる。この新しいGLR法の使用する記憶空間量は、富田法と同様 $n^2$ のオーダである。実験により、これらの一群のGLRパーザは、統語解析結果に含まれる曖昧さが増大するに連れて、高速な統語解析が可能であることを確認している。

2章では、富田法による認識アルゴリズムを改良したKippsの認識アルゴリズムについて説明する。3章では、Kippsの認識アルゴリズムをベースにしたDRITパーザのアルゴリズムについて説明する。DRITパーザは解析過程で逆ドット項を作る。4章では、3章で述べたパーザをさらにリファインした高速なGLRパーザについて述べる。5章では、先祖表を用いる一群のGLRパーザの実験による比較を行う。6章ではまとめと今後の課題について述べる。

## 2. 富田とKippsの認識アルゴリズム

本章では、認識器としての富田法の非公式な説明を与えてから、Kippsの認識アルゴリズムを説明する。標準的なGLR法についての知識を読者は既に持っているものと仮定する。これらのアルゴリズムの説明で使われる位置番号とは次のものである。

今、入力文 $w$ が $w = w_1 w_2 \dots w_n \in T^*$ であるとする。 $T^*$ は、終端記号の集合 $T$ に属す要素を0個以上並べたものを表す。終端記号 $w_i$ と $w_{i+1}$ の間に番号 $i$ を振り、これを、語 $w_i$ の位置番号とよぶ。ただし $w_1$ の左は位置番号0を、また $w_n$ の右は位置番号 $n$ である。

### 2.1 認識器としての富田法

本節ではKippsに従って、認識器としての富田法の非公式な説明と、認識時間のオーダの計算法の概略を述べる[Kipps 91]。GLR法では、解析過程で生じる構造的な(統語的な)曖昧性毎にスタックを用意する。各スタックにシフトする要素(節点)は、少なくともパーズの状態と親節点への有向弧を持つと考えてよい。富田法では複数個のスタックを一つのグラフ構造化スタック(GSS)として圧縮して表現する。GSSは節点と有向弧の集合から構成されている。GSSは、節点とその親節点へ向けた有向弧からなるDAGである。GSSは複数個の葉を持つことができる。葉はスタックトップの節点であり、それぞれ現在のパーズの状態を持っている。

[Kipps 91]に従い、認識器としての富田法の節点のデータ構造を $\langle i, s, L \rangle$ の三つ組であるとしよう。ここで $i$ はシフトされた語の位置番号、 $s$ は状態、 $L$ は親節点の集合である。Kippsの認識アルゴリズムは、この三つ組の中の $L$ の内容を変えて得られるのであるが、これは2.2で説明する。

富田法による認識器の動作の概要は次の通りである。文を解析する直前にGSSをクリアし、GSSに $\langle 0, 0, \{\} \rangle$ (位置番号0, 初期状態0)を置き、先読み語(走査中の語)を最初の語 $w_1$ にセットする。入力文の走査が左から右に1語ずつ進むにつれて、GSSは変化する。段階(stage)  $U_i$ は $i$ 番目の語 $w_i$ の走査が終り語 $w_i$ をシフトし、次の語 $w_{i+1}$ を(先読み語として)走査中の段階に対応している。 $U_i$ 中の節点は $\langle i, \dots \rangle$ という形をしている。文を解析する直前では、 $U_0$ に節点 $\langle 0, 0, \{\} \rangle$ のみが存在し、それがGSSの根になると共にスタックトップの節点(葉)にもなっている。

認識器は段階 $U_i$ 中の全ての葉に対して、LR表と先読み語 $w_{i+1}$ から決まる次の「レデュース」、「シフト」

、「受理」、「誤り」のいずれかの操作をGSSに施す。

(1)「レデュース」なら、指定されたレデュース操作を行い、 $U_i$ に新しい葉を付加する。付加した新しい葉は認識器の新たな処理対象となることに注意。

(2)「シフト」なら先読み語 $w_{i+1}$ のシフト操作を行い、 $U_{i+1}$ に一つの新しい葉を付加する。ただし $U_{i+1}$ に付加した新しい葉の処理は、 $U_i$ 中の葉の処理が全て終了し(先読み語 $w_{i+1}$ のシフトも終了し)次の先読み語 $w_{i+2}$ を走査する段階になるまで待たされる。

(3)「受理」なら認識成功とする。

(4)「誤り」なら誤りとしてその葉を消去する。

$U_i$ 中の葉の処理を全て終了すると、 $w_{i+2}$ を新たな先読み語として $U_{i+1}$ 中の葉の処理に進む。

(1)と(2)により、段階 $U_i$ と $U_{i+1}$ に新しい葉を付加する場合には、新しい葉から親節点に向けて有向弧を張る。各段階で、同じ状態を持つ葉が既にあれば、可能な場合両者をマージして一つの葉にする。このマージにより、葉は複数個の親節点を持つことがある。マージ操作は、それ以後の重複計算を避ける目的で導入されているが、それにより、各段階に存在する節点の総数は、高々状態の総数に等しく、定数で抑えられる。

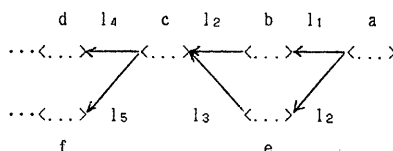
レデュース操作では、生成規則の右辺の非終端記号の数を $q$ とすると、レデュース操作の対象となる葉から $q$ だけ離れた先祖節点の集合を取り出す必要がある。こうして鳥だした各先祖節点の持つ状態とGOTO表から決まる新しい状態をもつ新しい葉を $U_i$ に作る。最後に、先の先祖節点を新しい葉の親節点とする。このようにしてレデュース操作により、離れた段階に属す節点も親節点になりうるので、たとえば $U_i$ に属す節点 $\langle i, s, L \rangle$ の親節点の集合 $L$ には、状態数の総数を $c$ としたとき、高々 $i * c$ 個、すなわち $i$ のオーダの親節点が含まれることになる。各段階は高々状態の総数 $c$ 個の節点しか存在しないので、一般に、節点 $\langle i, s, L \rangle$ の先祖節点の数も高々 $i$ 個である。

さて葉から $q$ だけ離れた先祖節点は、親節点に向けた有向弧を根に向かって $q$ 回辿って求めなければならない。親節点の数は高々 $i$ 個だから、 $q$ だけ離れた先祖節点に至る経路の総数は、最悪の場合 $i^q$ のオーダになる。したがって、生成規則の右辺にくる非終端記号の数の最大数を $m$ としたとき、認識器としての富田法の認識時間は、最悪の場合 $\sum_{i=1}^n i^m = n^{m+1}$ となり $m > 2$ なら $n^3$ を超える。ただしCFGがChomsky標準形なら $m = 2$ であるから、富田法でも認識時間のオーダは $n^3$ になる。

## 2. 2 Kippsの認識アルゴリズム

富田法の認識アルゴリズムが、一般のCFGに対して $n^3$ を超える認識時間を要す原因は、葉から $q$ だけ離れた高々 $i$ 個の先祖節点を求めるために、最悪の場合 $i^q$ のオーダの計算時間が必要になることによる。これは、先祖節点を求める過程で、同一の弧を繰り返し辿ることがあるからである。

たとえば次に示すグラフ構造化スタックでは、葉 $a$ から3だけはなれた先祖節点 $d$ と $f$ を取り出すために、リンク $l_4$ と $l_5$ を2度辿る。



同一の弧を繰り返し辿ることを防ぐことができれば、先祖節点を求める時間を短縮することができる。そのためKippsは、節点の三つ組を $\langle i, s, A \rangle$ に変える。ここで $i, s, A$ は、それぞれシフトされた語の位置番号とパーズの状態、先祖表(ancestors table)である。先祖表 $A$ はタプル $\langle k, L_k \rangle$ の集合から構成されている。ただし $k = 1, 2, \dots, m$  ( $m$ は生成規則の右辺にくる非終端記号の数の最大数)であり、 $L_k$ は、節点 $\langle i, s, A \rangle$ から $k$ 回弧を辿って得られる先祖節点の集合である。言い換えると、節点から深さ $k$ のところに位置する先祖節点の集合が $L_k$ である。ちなみに、2. 1の親節点の集合 $L$ は、先祖表 $A$ のタプル $\langle 1, L_1 \rangle$ として表現される。以上のことから、先祖表は高々 $m$ 個のタプルから構成され、タプル中の集合 $L_k$ の要素数は高々 $i$ 個であることが分かる。

このような先祖表を各節点があれば、シフト操作とレデュース操作時に新たな葉(節点)を作るとき、その葉の先祖表は、過去に作成した先祖表を利用して漸進的に埋めることができる。具体的には、葉の親節点の持つ先祖表 $A'$ 中のタプル $\langle k, L'_k \rangle$ を利用して、葉の先祖表 $A$ 中のタプル $\langle k+1, L_{k+1} \rangle$ を計算することができる。集合 $L'_k$ の要素を、そのまま集合 $L_{k+1}$ の要素とすることができるからである(3. 1. 1で与える関数NEWの定義参照)。こうして、レデュース操作時に先祖節点の集合を求めることは、先祖表の検索に置き換えられ、検索に要す時間は一定時間になる。Kippsによれば、先祖表を埋めるのに要す時間は $i^2$ のオーダーである。以上のことから、長さ $n$ の文に対して、 $i=1, 2, \dots, n$ までの総和をとって、Kippsの認識アルゴリズムの計算時間のオーダーが $n^3$ となることが証明できる。

### 3. DRITパーザ

3. 1以降で詳しく説明するが、シフト操作とレデュース操作時に、葉の持つ先祖表から、アーリー法で作成するアーリー項と相似な逆ドット項 [Tanaka 91a] を作成することができる。Kippsの認識アルゴリズムのシフト操作、レデュース操作にわずかな修正を施し逆ドット項 (Dot Reverse Item; DRIT) を作り出すパーザをDRITパーザとよぶことにする。

逆ドット項の集合  $R_i$  に属す項  $[A \rightarrow \alpha \cdot \beta, j]$  の意味は次の通りである。項内の生成規則の右辺にある  $\beta$  の直後の位置番号が  $j$  であり、ドット記号の位置番号が  $i$  である。そして、入力文の  $i+1$  番目の単語  $w_{i+1}$  から  $w_j$  までの部分が  $\beta$  として解析済みであることを表す。アーリー項の場合には、 $\beta$  ではなく  $\alpha$  が解析済みであるとされていたことに注意しよう。  $R_i$  に属す逆ドット項  $[A \rightarrow \cdot \gamma, j]$  は、入力文の単語  $w_{i+1}$  から  $w_j$  までの部分が  $\gamma$  として解析済みであり、それが  $A$  として認識されていることを表す。

アーリー項ではなく逆ドット項 (DRIT) を作る理由は、文献 [Tanaka 91a] に述べた。葉のもつ先祖表からアーリー項を作ろうとすると、不必要なアーリー項を作る可能性があることが問題であった。興味ある読者は、4. 2の (b) の GSS に対して、アーリー項を作られて、このことを確かめられるとよい。DRITの場合にはその心配はない。これは、GLR法が最右導出をベースにしていることによる。重複した項を検出する場合にも、各段階で作られるDRITの集合を調べるだけでよいので、重複項を検査する範囲を局所化することができる。

以下では葉の持つ先祖表からDRITを作成する方法を順に説明する。

#### 3. 1 レデュース操作時のDRIT生成

CFG中の生成規則には番号が振られているとする。  $p$  番目の生成規則が、  $D_p \rightarrow C_{p1} C_{p2} \dots C_{pq}$  であり、これがレデュース操作で用いる規則であるとしよう。そしてこのレデュース操作の対象となるスタックトップの節点 (葉) が、  $\langle i, s, A \rangle$  であるとしよう。この葉には非終端記号  $C_{pq}$  が、その親節点には  $C_{pq-1}$  が、以下同様にして、葉から  $k$  だけ離れた節点には  $C_{pq-k}$  が対応する。  $p$  番目の生成規則を実際に適用することは、この時点で、この生成規則に対応した部分統語解析木が作成可能なことを意味する。富田法ではこの機会をとらえて部分統語解析を作るが、DRITパーザでは統語解析木の部品であるDRITを作る。

葉  $v$  のもつ先祖表  $A$  を検索するだけで関数  $ANCESTOR$   $S$  は、葉から  $k$  だけ離れた先祖節点の集合を計算する。

```
ANCESTORS(v = <i, s, A>)
  if k = 0, return({v})
  else if  $\exists \langle k, L_k \rangle \in A$ , return( $L_k$ )
```

レデュース操作は、GOTO表を参照して新しい葉を作る。この時、作った葉に先祖表を付与する必要がある。付与する先祖表は、概略2. 2のおわりに述べた方法により、各エントリーを埋めておく。その手続きNEWを以下に示す。

```
NEW(v = <i, a, A>, u = <j, t, B>) %節点uは葉vの親
  for k from 2 to m %mはCFG規則のRHSに現れ
    if  $\exists \langle k-1, L'_{k-1} \rangle \in B$  %非終端記号の最大数
      if  $\exists \langle k, L_k \rangle \in A$ 
        let  $L_k := L_k \cup L'_{k-1}$ 
      else
        let  $A := A \cup \{\langle k, L'_{k-1} \rangle\}$ 
    else return(v)
```

レデュース操作で用いる生成規則が次のものであるとしよう。

$$D_p \rightarrow C_{p1} C_{p2} \dots C_{pq-k} C_{pq-k+1} \dots C_{pq}$$

この場合には、葉に存在する先祖表だけから、DRITを以下のようにして作り出す。

```
for k from 1 to q
  for  $\forall j' \text{ s.t. } \langle j', s', A' \rangle \in ANCESTORS(v, k)$ 
    let  $R_{j'} := R_{j'} \cup \{[D_p \rightarrow C_{p1} \dots C_{pq-k} \cdot C_{pq-k+1} \dots C_{pq}, i]\}$ 
```

ここで、  $k = q$  の時には、  $[D_p \rightarrow C_{p1} C_{p2} \dots C_{pq}, i]$  なるDRITができることに注意したい。

準備ができたので、レデュース操作の手続を以下に与える。

```

REDUCE(v, p)
for k from 1 to q
  for  $\forall j' : s.t. \langle j', s', A' \rangle \in \text{ANCESTORS}(v, k)$ 
    let  $R_{j'} := R_{j'} \cup \{ [D_p \rightarrow C_{p1} \dots C_{pq-k} \cdot C_{pq-k+1} \dots C_{pq}, i] \}$ 
  for  $\forall v1' = \langle j', s', A1' \rangle s.t. v1' \in \text{ANCESTORS}(v, q)$ 
    let  $s'' := \text{GOTO}(s', D_p)$ 
    if  $\exists v'' = \langle i-1, s'', A'' \rangle s.t. v'' \in U_{i-1} \ \& \ \langle 1, L'' \rangle \in A''$ 
      if  $v1' \in L''$ 
        do nothing(ambiguous)
      else
        if  $\exists v2' = \langle j', s', A2' \rangle s.t. v2' \in L''$ 
          let  $vc'' := \langle i-1, s'', Ac'' \rangle$ 
          s.t.  $Ac'' = \{ \langle 1, \{v1'\} \rangle \}$ 
          let  $vc'' := \text{NEW}(vc'', v1')$ 
          for  $\forall' re p' \in \text{ACTIONS}(s'', w_i)$ ,
            REDUCE( $vc'', p$ )
        else
          let  $L'' := L'' \cup \{v1'\}$ 
          let  $v'' := \text{NEW}(v'', v1')$ 
          if  $v'' \in P$ 
            let  $vc'' := \langle i-1, s'', Ac'' \rangle$ 
            s.t.  $Ac'' = \{ \langle 1, \{v1'\} \rangle \}$ 
            let  $vc'' := \text{NEW}(vc'', v1')$ 
            for  $\forall' re p' \in \text{ACTIONS}(s'', w_i)$ 
              REDUCE( $vc'', p$ )
          else
            let  $v'' := \langle i-1, s'', A \rangle s.t. A = \{ \langle 1, \{v1'\} \rangle \}$ 
            let  $v'' := \text{NEW}(v'', v1')$ 
            let  $U_{i-1} := U_{i-1} \cup \{v''\}$ 

```

上記したレデュース操作の手続き中、太字の部分が、Kippsの認識アルゴリズムに新たに付加した部分である。節点 $vc''$ は、Kippsがクローンとよぶ節点であり、 $\varepsilon$ 規則などを扱うために導入されている。これについてはKippsの論文[Kipps 91]を参照されたい。

関数ANCESTORSと関数NEWの定義は、Kippsの与えた定義と大幅に異なる。DRITを作るためには、先祖表の全てのエンタリーが埋められていると都合がよいことを考慮したからである。

### 3.2 シフト操作時のDRIT生成

語 $w_i$ をシフトする場合を考えよう。語 $w_i$ の前終端記号(preterminal)を $C$ とすると、シフト操作時に逆ドット項 $[C \rightarrow \cdot w_i, i]$ を $R_{i-1}$ に作ることができる。

$$R_{i-1} := R_{i-1} \cup \{ [C \rightarrow \cdot w_i, i] \}$$

DRITの集合 $R_{i-1}$ へ、上記したDRITを所属させる理由は、

DRIT中の語 $w_i$ の左にあるドットの位置番号が $i-1$ であることによる。

DRITパーザで用いるシフト操作の手続きを以下に与える。太字の部分がDRITを作成する部分であり、Kippsの認識アルゴリズムに新たに付加した部分である。

```

SHIFT(v, s) %状態sの葉v1を作る、v1の親がv.
 $R_{i-1} := R_{i-1} \cup \{ [C \rightarrow \cdot w_i, i] \}$ 
if  $\exists v1 = \langle i, s, A \rangle s.t. v1 \in U_i \ \& \ \langle 1, L \rangle \in A$ 
  let  $L := L \cup \{v\}$ 
else
  let  $v1 := \langle i, s, A \rangle s.t. A = \{ \langle 1, \{v\} \rangle \}$ 
  let  $v1 := \text{NEW}(v1, v)$ 
  let  $U_i := U_i \cup \{v1\}$ 

```

### 3.3 トップレベルの手続き

トップレベルの手続きPARSEを以下に与える。ここで $w_{n+1}$ は文末の記号を表す。

```

PARSE( $w_1 w_2 \dots w_{n+1}$ )
let  $w_{n+1} := -1$  %文末記号
let  $U_i := \{ \}$  ( $0 \leq i \leq n$ )
let  $U_0 := \{ \langle 0, 0, \{ \} \rangle \}$ 
for i from 1 to n+1
  let  $P := \{ \}$ 
  for  $v = \langle i-1, s, a \rangle s.t. v \in U_{i-1}$ 
    let  $P := P \cup \{v\}$ 
    if  $\exists' sh s' \in \text{ACTIONS}(s, w_i)$ ,
      SHIFT( $v, s'$ )
    for  $\forall' re p' \in \text{ACTIONS}(s, w_i)$ , REDUCE( $v, p$ )
    if  $'acc' \in \text{ACTIONS}(s, w_i)$ , accept
    if  $U_i = \{ \}$ , reject

```

### 3.4 DRITパーザの計算複雑性

段階 $U_i$ で、DRIT作成に要す時間のオーダーを考えてみよう。関数ANCESTORSにより先祖節点の集合を取り出す時間は一定であった。得られる先祖節点の数は高々 $i$ であるからDRIT作成に要す時間は $i$ のオーダーである。一方、レデュース操作を繰り返し、新しい葉を作る度に、高々 $i$ 個の親節点の先祖表から、葉の先祖表を作る時間のオーダーは $i$ で、これが高々 $i$ 回繰り返されるので、段階 $U_i$ で先祖表を作るのに総計 $i^2$ のオーダーの時間がかかる。従って、長さ $n$ の文に対するDRITパーザの計算時間のオーダーは、Kippsのそれと等しく $n^3$ になる。

次に、使用記憶空間は、GSS用として確保した記憶空

間と、作成されたDRITの総数に依存する。前者のオーダはGSSの構造から $n^2$ であることは明かだろう。後者については、ANCESTORSから計算した先祖節点数の数に比例するので、段階 $U_i$ で $i$ のオーダである。これが $n$ 回繰り返されるので、 $n^2$ のオーダになる。以上のことから、DRITパーザの使用記憶空間量は $n^2$ のオーダであることが分かる。

#### 4. AGLRパーザ

本章ではDRITパーザのリファインを考える。

##### 4.1 AGLRパーザ第1版 (AGLR1)

DRITパーザは、レデュース操作時に、葉の持つ先祖表だけを用いて、統語解析木の部品であるDRITを作り出すものであった。そうであるなら、逆ドット項の代わりに、葉の持つ先祖表をそのまま記憶しておくことが考えられる。詳しい理由は述べないが、こうすれば、重複したDRITの排除を行う必要もなくなる。具体的には、3.1で説明したREDUCEの手続きの中で、DRITを作成する部分を次のように変えてやればよい。

$$R := R \cup [p, i, \text{TREE}(A)] \text{ s. t. } v = \langle i, s, A \rangle$$

ここに、 $p$ はCFG規則の番号であり、 $i$ は葉節点の位置番号である。関数TREEは、先祖表 $A$ に含まれる全ての先祖節点を、その先祖節点の持つ位置番号で置き換えた表を値とする。

一方、シフト操作の場合には、3.2の定義中のDRITを作る部分を次のものに変える。

$$R := R \cup [C \rightarrow w_i, i, \{ \langle 1, (i-1) \rangle \}]$$

以上で説明したGLRパーザは、もはやDRITを作成しないので、DRITパーザとよぶのはふさわしくない。そこで以下では、この種の先祖表を用いた一群のパーザをAncestor GLRパーザ、略してAGLRパーザとよぶことにする。DRITパーザはAGLRパーザの仲間である。この方法も、DRITパーザの解析時間のオーダ( $n^3$ )を変えないことは明かだろう。

##### 4.2 統語解析木の抽出

AGLRパーザの出力は $R$ であるが、この $R$ の内容から、DRITの集合と同様、可能な統語解析木をすべて抽出することができる。このアルゴリズムの詳細については、別稿に譲るとして、この場合、一つの木を抽出するのに要す時間は $n^2$ のオーダになる。これは、DRITパーザ

の場合も、アーリー法の場合も同様のオーダである。

一方、富田法は、 $n^4$ 以上の解析時間をもつCFGの存在が知られているが、富田法の出力は圧縮統語森であり、ここから一つの統語解析木を取り出すのに要す時間は高々 $n$ のオーダである。この問題を考えるために、次の例を考察してみよう。

入力文  $w$  が  $w = w_1 w_2 \dots w_n$  であるとしよう。まずはじめに、節点中の位置番号の意味を理解するために、位置番号以外の情報を無視した次のスタック (a) を考えよう。

(a)  $\dots - \langle 3, \dots \rangle \times \langle 5, \dots \rangle \times \langle 6, \dots \rangle$  (top)  
reduce by  $X \rightarrow YZ$

$\langle 6, \dots \rangle$ はスタックトップの節点であり葉である。節点中の位置番号は、その位置番号までの入力文の語が処理済み(シフト済み)であることを意味している。したがって、節点 $\langle 6, \dots \rangle$ は入力文の $w_6$ までが処理済みであり、同様にして節点 $\langle 5, \dots \rangle$ は入力文の $w_5$ までが処理済みであることを意味している。このことから、節点 $\langle 6, \dots \rangle$ には入力文の $w_6$ の部分が対応し、節点 $\langle 3, \dots \rangle$ と節点 $\langle 6, \dots \rangle$ で挟まれた節点 $\langle 5, \dots \rangle$ には $w_4 w_5$ が対応している。スタック (a) へ、レデュース操作"reduce by  $A \rightarrow BC$ "を施すことを考える。この場合には、節点 $\langle 6, \dots \rangle$ が $Z$ に、節点 $\langle 5, \dots \rangle$ が $Y$ に対応している。そして位置番号6から3の間にある入力文の部分( $w_4 w_5 w_6$ )が" $Y Z$ "として認識され、それが $X$ にまとめられたことを意味している。

次にスタックトップがマージされた(b)に示すスタックを用いる。

(b)  $\dots - \langle 3, \dots \rangle \times \langle 5, \dots \rangle \times \langle 6, s, A \rangle$  (top)  
reduce by  $X \rightarrow YZ$   
 $\dots - \langle 2, \dots \rangle \times \langle 4, \dots \rangle$

ただし  $A = \{ \langle 1, \{v_5, v_4\} \rangle, \langle 2, \{v_3, v_2\} \rangle, \dots \}$  であり、各 $v_i$ は節点へのポインタであるとする。

$$v_5 = \langle 5, \dots \rangle$$

$$v_4 = \langle 4, \dots \rangle$$

$$v_3 = \langle 3, \dots \rangle$$

$$v_2 = \langle 2, \dots \rangle$$

$$\dots$$

(b)に、レデュース操作"reduce by  $X \rightarrow YZ$ "を施す。葉から1離れた節点が $v_5$ と $v_4$ であり、それらの位置番号が5と4であること、また葉から2だけ離れた節点が $v_3$ と $v_2$ であり、それらの位置番号が3と2であるこ

とに注意しよう。この時、葉<6, s, A>のもつ先祖表Aから、次を得る。

$R := R \cup \{<1, \{5, 4\}>, <2, \{3, 2\}>\}$

以上から

(1) 葉から2だけ離れた先祖節点v3とv2の位置番号が、それぞれ3と2であるから、w4w5w6とw3w4w5w6がYZとして処理済みで、Xにまとめられる。

(2) 葉<6, s, A>から1離れた親節点v4とv5の位置番号が4と5であるから、w6とw5w6がZとして処理済み。

しかし、

(3) YZとしてまとめられたw4w5w6に対して、w4がYに、w5w6がZにまとめられるのか、w4w5がYに、w6がZにまとめられるのかが不明、

(4) YZとしてまとめられたw3w4w5w6に対して、w3w4がYに、w5w6がZにまとめられるのか、w3w4w5がYに、w6がZにまとめられるのか不明。

(3)と(4)のために、AGLRパーザから得た解析結果Rから、一つの統語解析木を取り出すのに、 $n^2$ のオーダーの時間がかかる。DRITパーザの場合にもアーリー法の場合にも同様な問題が起こる。

#### 4.3 AGLRパーザ第2版(AGLR2)

4.2の最後に述べた問題を解決する鍵は、入力文のどこからどこまでがYとZにまとめられるかを知ることである。そのために、新しい葉vを作るとき、先祖表に自分自身を<0, {v(l)}>の様に書き込む。ただしここで、lはvの親節点の持つ位置番号の集合である。たとえば前節の(b)の場合には、葉<6, s, A>の先祖表Aは、次のようになる。

$A = \{<0, \{v6(5, 4)\}>, \\ <1, \{v5(3), v4(2)\}>, \\ <2, \{v3(\dots), v2(\dots)\}>, \dots\}$

この先祖表から、次を得る。

$R := R \cup \{<0, \{6(5, 4)\}>, \\ <1, \{5(3), 4(2)\}>, <2, \{3(\dots), 2(\dots)\}>\}$

そしてこのRから、位置番号5から6の間の語w6と、位置番号4から6の間の語w5w6がZであり、位置番号3と5の間の語w3w4w5と、位置番号2と4の間の語w3w4がYであることを知ることができる。したがって、

(1) w4w5w6とw3w4w5w6がYZとして処理済みで、さらにXとしてまとめられる。

(2) w6とw5w6がZとして処理済み。

(3) YZとしてまとめられたw4w5w6に対して、w4がYに、w5w6がZにまとめられる。

(4) YZとしてまとめられたw3w4w5w6に対して、w3w4がYに、w5w6がZにまとめられる。

以上のことから、我々は無駄なく統語解析木を抽出することができることが分かる。したがって、解析結果Rから一つの統語解析木を抽出するのに必要な時間のオーダーはnとなる。

#### 5. 実験結果

これまで述べてきた一般化LRパーザの評価をするために、実験を行い比較を試みた。解析を行った文は、PP付加文の、I open the door with a key with a ..であり、およそ20MIPSの性能のワークステーションで規則数123の英語の文法規則を用いて実験を行った。AGLR1パーザは、4.1に説明したものである。SAXパーザは、チャート法をベースにした高速なパーザであるとされていたものである。

使用文法：英文法(規則数123)

使用機器：ソニーNews (20MIPS)

入力文：I open the door with a key with a ...

| 文の長さ | 曖昧性の数 | 解析時間 (msec)   |                |          |
|------|-------|---------------|----------------|----------|
|      |       | DRIT<br>h'-s' | AGLR1<br>h'-s' | SAXh'-s' |
| 16   | 84    | 48            | 26             | 1430     |
| 19   | 264   | 65            | 39             | 4620     |
| 22   | 858   | 95            | 51             | 14760    |
| 25   | 2860  | 122           | 71             | 49550    |
| 28   | 9724  | 167           | 89             | 171031   |
| 31   | 33592 | 220           | 120            | 595430   |

AGLR1パーザの場合、3万強の数の曖昧性を持つ文の統語解析に要する時間は、僅か120ミリ秒である。これは、これまで高速であるとされていた言語解析システムSAXと比べておよそ5000倍ほど速い。

## 6. おわりに

一般化LR法の一つである富田法に対して、アーリー法以上のオーダの解析時間を要す特殊なCFGの存在が知られていた。Kippsは富田法にわずかな修正を加えることで、一般のCFGに対して、アーリー法と同等の解析時間のオーダ（文の長さ $n$ に対して $n^3$ のオーダ）を要す認識アルゴリズムを与えている[Kipps 91]。しかしKippsのアルゴリズムは、与えられた文の文法的妥当性を知るための認識アルゴリズムに過ぎず、統語解析木（構文構造）を得ることができず実用にならないとされている[Schabes 91]。

本論文では、先祖表を利用した一群の新しい一般化LR統語解析アルゴリズム（AGLRパーザ）を提案している。これは、Kippsの認識アルゴリズムにわずかな修正を施すことで、実現可能であり、実験によれば極めて高速な統語解析が可能である。Kippsの認識アルゴリズムをベースにしているの、統語解析時間のオーダは、文の長さを $n$ とした時 $n^3$ であり、使用記憶空間のオーダは $n^2$ である。また、圧縮統語森と同様、解析結果から得られる情報から、一つの統語解析木を抽出する時間のオーダが $n$ のアルゴリズムを示した。

今後の検討課題として、以下のものを挙げることができる。

- ・実験により、さらに詳細なAGLRパーザの評価を行うこと、また富田法と比較することも必要だろう。これらについては、結果が出しだい、後日報告したいと考えている。
- ・並列解析アルゴリズムの研究。
- ・統語解析木を抽出するための、並列アルゴリズムの研究
- ・AGLRパーザをベースにした自然言語解析用のツールの開発。

## 参考文献

- [Aho 72] Aho, A.V. and Ullman, J.D.: The Theory of Parsing, Translation and Compiling, Prentice-Hall, New Jersey(1972).
- [Earley 70] Earley, J.: An Efficient Augmented-Context-Free Parsing Algorithm, Comm. of ACM, 13, 1-2, 95-102(1970).
- [Johnson 91] Johnson, M.: The Computational Complexity of GLR Parsing, in Tomita, M ed.: Generalized LR Parsing, Kluwer Academic Publishers, 35-41(1991).
- [Kipps 91] Kipps, J. N.: GLR Parsing in Time  $O(n^3)$ , in Tomita, M ed.: Generalized LR Parsing, Kluwer Academic Publishers, 43-59(1991).
- [Numazaki 90] Numazaki, H. and Tanaka, H.: A New Parallel Algorithm for Generalized LR Parsing, COLING'90, 305-310(1990).
- [Schabes 91] Schabes, Y.: Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-free Grammars, Proc. of 29th ACL, 106-107(1991).
- [Suresh 91] Suresh, K.G. and Tanaka, H.: Implementation and Evaluation of Yet Another Generalized LR Parsing Algorithm, Proc. of the Indian Computer Congress, Tata McGraw-Hill, 506-515(1991).
- [Tanaka 91a] Tanaka, H and Suresh, K.G.: YAGLR: Yet Another Generalized LR Parser, Proc. of ROCLING IV, Republic of China, 21-31(1991).
- [Tanaka 91b] Tanaka, H and Numazaki, H.: Parallel GLR Parsing Based on Logic Programming, in Tomita, M ed.: Generalized LR Parsing, Kluwer Academic Publishers, 77-91(1991).
- [Tomita 86] Tomita, M.: Efficient Parsing for Natural Language, Kluwer, Boston, Mass(1986).
- [Tomita 87] Tomita, M.: An Efficient Augmented-Context-Free Parsing Algorithm, Computational Linguistics, 13, 31-46(1987).