自然言語処理 90－5
（1992．7．17）

先祖表を利用した一般化ＬＲパージ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の解析時間を要する特殊な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どるためである．

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どる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パージング・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．

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 Ookayama 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 alogorithm 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の解析時間を要する特殊な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どるためである．

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ど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パージング・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，アーリー法と同様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$w = w_1 w_2 \ldots w_n \varepsilon T^*$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孤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は節点と有向孤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孤からなる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＜i,s,L＞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＜0,0,{}＞（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＜i,.....＞という形をしている．文を解析する直前では，U_0に節点＜0,0,{}＞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孤を張る．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．

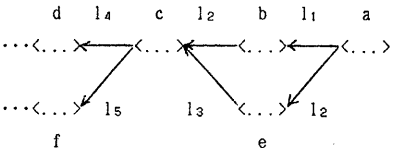
レデュース操作では，生成規則の右辺の非終端記号の数をqとすると，レデュース操作の対象となる葉からqだけ離れた先祖節点の集合を取り出す必要がある．こうして鳥だした各先祖節点の持つ状態とＧＯＴＯ表から決まる新しい状態をもつ新しい葉をU_iに作る．最後に，先の先祖節点を新しい葉の親節点とする．このようにしてレデュース操作により，離れた段階に属す節点も親節点になりうるので，たとえばU_iに属す節点$<i,s,L>$の親節点の集合Ｌには，状態数の総数をｃとしたとき，高々$i*c$個，すなわちｉのオーダの親節点が含まれることになる．各段階は高々状態の総数ｃ個の節点しか存在しないので，一般に，節点$<i,s,L>$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孤を根に向かって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$\Sigma^n i^m = n^{m+1}$となりｍ＞２ならn^3を越える．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を越える認識時間を要する原因は，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，同一の孤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．



同一の孤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$<i,s,A>$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$<k,L_k>$の集合から構成されている．ただしｋ＝１，２，...，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，Ｌｋは，節点$<i,s,A>$からｋ回孤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$<1,L_1>$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持てば，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$<k,L'_k>$を利用して，葉の先祖表Ａ中のタプル$<k+1,L_{k+1}>$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,...,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→α・β，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β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β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，βではなくα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→・γ，j］は，入力文の単語w_{i+1}からw_jまでの部分がγ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，$D_p \rightarrow C_{p1}C_{p2}\cdots C_{pq}$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，<i,s,A>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ANCESTORS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 ∃<k,Lk>∈A, return(Lk)
```

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 ∃<k-1,L'k-1>∈B          %る非終端記号の最大数
      if ∃<k,Lk>∈A
        let Lk := Lk ∪ L'k-1
      else
        let A := A ∪ {<k,L'k-1>}
    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．

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

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∀j' s.t. <j',s',A'>∈ANCESTORS(v,k)
        let Rj' := Rj' ∪ {[Dp→Cp1 ··· Cpq-k ·
                           Cpq-k+1 ... Cpq, i]}
```

ここで，k＝qの時には，$[D_p \rightarrow \cdot C_{p1}C_{p2}\cdots C_{pq}, i]$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∀j' s.t. <j',s',A'>∈ANCESTORS(v,k)
     let Rj' := Rj' ∪ {[Dp→Cp1 ・・・ Cpq-k・
                          Cpq-k+1 ... Cpq, i]}
 for ∀v1' = <j',s',A1'> s.t. v1'∈ ANCESTORS(v,q)
   let s" := GOTO(s',Dp)
   if∃v" = <i-1,s",A"> s.t. v"∈Ui-1 & <1,L"> A"
     if v1'∈ L"
        do nothing(ambiguous)
     else
       if∃v2' = <j',s',A2'> s.t. v2'∈ L"
         let vc" := <i-1,s",Ac">
                    s.t. Ac" = {<1,{v1'}>}
         let vc" := NEW(vc",v1')
         for∀'re p'∈ACTIONS(s",wi),
              REDUCE(vc",p)
       else
         let L" := L" ∪ {v1'}
         let v" := NEW(v",v1)
         if v"∈P
           let vc" := <i-1,s",Ac">
                      s.t. Ac" = {<1,{v1'}>}
           let vc" := NEW(vc",v1')
           for∀'re p' ACTIONS(s",wi)
                REDUCE(vc",p)
   else
     let v":=<i-1,s",A> s.t. A={<1,{v1'}>}
     let v":=NEW(v",v1')
     let Ui-1 := Ui-1 ∪ {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，ε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ト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→・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．
  Ri-1 := Ri-1 ∪ {[C→・wi, i]}
  if∃v1=<i,s,A> s.t. v1∈Ui &<1,L>∈A
     let L := L ∪ {v}
  else
     let v1 := <i,s,A> s.t. A = {<1,{v}>}
     let v1 := NEW(v1,v)
     let Ui := Ui ∪ {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(w1w2...wn+1)
  let wn+1 := -| %文末記号
  let Ui := {}  (0≦i≦n)
  let U0 := {<0,0,{}>}
  for i from 1 to n+1
    let P := {}
    for  v = <i-1,s,a> s.t. v∈Ui-1
        let P := P ∪ {v}
        if∃'sh s'∈ACTIONS(s,wi),
            SHIFT(v,s')
        for∀'re p'∈ACTIONS(s,wi), REDUCE(v,p)
        if 'acc'∈ACTIONS(s,wi), accept
        if Ui = {}, 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れてやればよい．

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

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．この問題を考えてみるために，次の例を考察してみよう．

入力文 w が $w=w_1w_2\ldots w_n$ 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)　$\cdots\langle 3,\ldots\rangle\langle 5,\ldots\rangle\langle 6,\ldots\rangle$　(top)
reduce by X→YZ

<6,...>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<6,...>は入力文のw_6までが処理済みであり，同様にして節点<5,...>は入力文のw_5までが処理済みであること意味している．このことから，節点<6,...>には入力文のw_6の部分が対応し，節点<3,...>と節点<6,...>で挟まれた節点<5,...>にはw_4w_5が対応している．スタック（ａ）へ，レデュース操作"reduce by A→BC"を施すことを考える．この場合には，節点<6,...>がＺに，節点<5,...>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（$w_4w_5w_6$）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)　$\cdots\langle 3,\ldots\rangle\langle 5,\ldots\rangle\langle 6, s, A\rangle$　(top)
reduce by X→YZ
$\cdots\langle 2,\ldots\rangle\langle 4,\ldots\rangle$

ただし　$A=\{\langle 1, \{v5, v4\}\rangle, \langle 2, \{v3, v2\}\rangle, \ldots\}$であり，各$v_i$は節点へのポインタであるとする．

v5=<5,...>
v4=<4,...>
v3=<3,...>
v2=<2,...>
.....

（ｂ）に，レデュース操作"reduce by X→YZ"を施す．葉から１離れた節点がv5とv4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v3とv2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 ∪ {<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が不明．

（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(...),v2(...)}>,...}
```

この先祖表から，次を得る．

```
R := R ∪ {<0,{6(5,4)}>,
          <1,{5(3),4(2)}>,<2,{3(..),2(..)}>}
```

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 ﾊﾟｰｻﾞ	AGLR1 ﾊﾟｰｻﾞ	SAXﾊﾟｰｻﾞ
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ー（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で，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: Implememtation 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).