ICOT Technical Report: TR-627

TR-627

YAGLR法：Yet Another Generalized LR Parser

田中　穂積、G．スレッシュ
（東京工業大学）

March, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

1990.2.6

ＹＡＧＬＲ法: Yet Another Generalized LR Parser

田中穂積，Ｋ．Ｇ．スレッシュ
（東京工業大学・工学部）

概要

プログラム言語などの人工言語のコンパイラでは，Knuthの考案したＬＲ（ｋ）法とよぶ統語解析アルゴリズム［Knuth ６５］が使われることがある．これは先読み情報を用い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．ただしそこで使われる文法は，ＬＲ文法に限られており，自然言語の解析に用いる一般の文脈自由文法（ＣＦＧ）を扱うことができない．

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［Tomita ８６，８７］．これは一般化ＬＲ法（generalized LR parsing algorithm；拡張ＬＲ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［Tomita ８６］，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る．

我々は，アーリー法の利点と富田法の利点をそのまま引き継ぐ新しい一般化ＬＲ法（これをＹＡＧＬＲ法とよぶ）を提案する．それによれば，先に述べた特殊なＣＦＧに対する解析時間のオーダをn^3に抑えることができる．また，富田法ではグラフ構造化スタック（ＧＳＳ）を用いて，使用記憶空間の節約を図っているが，ＧＳＳの構造は必ずしも単純でなく，実装が容易ではない．これに対してＹＡＧＬＲ法では木構造化スタック（ＴＲＳＳ）を用いており，経験的に，富田法のＧＳＳと比較して同程度の使用記憶空間に抑えることが可能であり，構造が単純であるため扱い易い．これは，ＹＡＧＬＲ法では富田法以上にスタックのマージが可能になるためである．本論文では，ＹＡＧＬＲ法による統語解析アルゴリズムの定義と動作例を説明する．また，富田法と比較して，ＹＡＧＬＲ法はどの様な利点があるかを具体例を用いて説明する．

1990.2.7

ＹＡＧＬＲ法: Yet Another Generalized LR Parser

田中穂積，Ｋ．Ｇ．スレッシュ
東京工業大学・工学部

概要

プログラム言語などの人工言語のコンパイラでは，Knuthの考案したＬＲ（ｋ）法とよぶ統語解析アルゴリズム［Knuth ６５］が使われることがある．これは先読み情報を用い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．ただしそこで使われる文法は，ＬＲ文法に限られており，自然言語の解析に用いる一般の文脈自由文法（ＣＦＧ）を扱うことができない．

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［Tomita ８６，８７］．これは一般化ＬＲ法（generalized LR parsing algorithm；拡張ＬＲ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［Tomita ８６］，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る．

我々は，アーリー法の利点と富田法の利点をそのまま引き継ぐ新しい一般化ＬＲ法（これをＹＡＧＬＲ法とよぶ）を提案する．それによれば，先に述べた特殊なＣＦＧに対する解析時間のオーダをn^3に抑えることができる．また，富田法ではグラフ構造化スタック（ＧＳＳ）を用いて，使用記憶空間の節約を図っているが，ＧＳＳの構造は必ずしも単純でなく，実装が容易ではない．これに対してＹＡＧＬＲ法では木構造化スタック（ＴＲＳＳ）を用いており，経験的に，富田法のＧＳＳと比較して同程度の使用記憶空間であり，構造が単純であるため扱い易い．これは，ＹＡＧＬＲ法では富田法以上にスタックのマージが可能になるためである．本論文では，ＹＡＧＬＲ法による統語解析アルゴリズムを与え，富田法との比較を具体例により説明する．

[Abstract]

We have developed a new generalized LR parsing algorithm called YAGLR in which it combines some of the advantages of both Earley's and Tomita's algorithms. We show that for some special CFGs the time consumed by YAGLR parsing algorithm is less than Tomita's parsing algorithm and is in the order of $O(n^3)$. While YAGLR uses a tree structured stack, the sturucture is not so complex compared to the one of Tomita's graph-structured stack. The reason is that YAGLR parsing algorithm enables us to make much more merge operations on the tree-structured stack.

1．はじめに

プログラム言語などの人工言語のコンパイラでは，Knuthの考案したＬＲ（ｋ）法とよぶ統語解析アルゴリズム［Knuth ６５］が使われることがある．これは先読み情報を用い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．ただしそこで使われる文法は，ＬＲ文法に限られており，自然言語の解析に用いる一般の文脈自由文法（ＣＦＧ）を扱うことができない．

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［Tomita ８６，８７］．これは一般化ＬＲ法（generalized LR parsing algorithm；拡張ＬＲ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［Tomita ８６］，富田法に対して，解析すべき文の長さをｎとしたとき，n^m（$m>3$）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 ８９］［Kipps ８９］．一方．，アーリー法は，いかなるＣＦＧに対しても，解析時間のオーダがn^3であるから，この特殊なＣＦＧに対して，富田法はアーリー法以上の解析時間を要することになる．

本稿で提案する新しい一般化ＬＲ法では，アーリー法の利点と富田法の利点をそのまま引き継ぐ．この新しい一般化ＬＲ法をＹＡＧＬＲ法（Yet Another Generalized LR parsing)とよんでいる．ＹＡＧＬＲ法によれば，先に述べた特殊なＣＦＧに対する解析時間のオーダをn^3に抑えることができる．また，富田法ではグラフ構造化スタック（ＧＳＳ）を用いて，使用記憶空間の節約を図っているが，ＧＳＳの構造は必ずしも単純でなく，実装が容易ではない．これに対してＹＡＧＬＲ法では木構造化スタック（ＴＲＳＳ）を用いるが，経験的に，富田法のＧＳＳと比較して使用記憶空間は同程度であり，構造が単純であるため扱い易い．これは，ＹＡＧＬＲ法では富田法以上にスタックのマージが可能になるためである．

ＹＡＧＬＲ法では，解析中に統語解析木は作らず，統語解析木を構成する部品のみを作成する．この部品を逆ドット項と呼ぶが，これはアーリー法で作成するドット項（アーリー項）と対称な構造をしている．２章では，逆ドット項を説明する．３章では，ＹＡＧＬＲ法の鍵となるＴＲＳＳのマージ操作（統合化操作），シフト操作，レデュース操作を説明し，最後にＹＡＧＬＲ法の統語解析アルゴリズムを説明する．レデュース操作の過程で逆ドット項を作成するが，逆ドット項を作成することの利点も３章で説明する．また富田法とＹＡＧＬＲ法による解析例を示し，富田法とＹＡＧＬＲ法の比較を行う．４章では，特殊なＣＦＧに対しても，ＹＡＧＬＲ法の統語解析時間のオーダがn^3であることを証明する．５章では，まとめと今後の研究課題について述べる．

2．　逆ドット項(Dot Reverse Item)

本章では，ＹＡＧＬＲ法による統語解析で重要な役割を果たす逆ドット項を説明する．富田法がレデュース操作時に部分統語解析結果を圧縮統語森として作成するのに対して，ＹＡＧＬＲ法ではアーリー項と対称な逆ドット項の集合を作成する．逆ドット項の定義を以下に与える．

（１）解析対象文ｗを$w=w_1w_2\ldots w_n \in T^*$であるとする．ここに$T^*$は，終端記号の集合Ｔの要素を０個以上並べたものを表す．終端記号（語と考えてよい）w_iとw_{i+1}の間に番号ｉを振り，これを位置番号とよぶ（ｉを，語w_iの位置番号とよぶ）．ただしw_1の左には位置番号０を，またw_nの右には位置番号ｎを振る．

（２）ＣＦＧ規則Ａ→αがあり，αが$\beta\gamma$と書けるとき，$0\leq j\leq n$なるｊに対して，［Ａ→β・γ，ｊ］は，ｗに対する逆ドット項である．特に，［Ａ→・α，ｊ］または［Ａ→α・，ｊ］もｗに対する逆ドット項である．

（３）逆ドット項の集合I_iを次のように定義する．$0\leq i\leq j\leq n$なるｉとｊに対して，

［Ａ→α・β，ｊ］$\in I_i$　iff　$S\Rightarrow_\gamma A\delta$，$\beta\Rightarrow w_{i+1}w_{i+2}\ldots w_j$，and　$\delta\Rightarrow w_{j+1}w_{j+2}\ldots w_n$

アーリー項も逆ドット項も，ドットの位置は項の集合Ｉの添字ｉとして表す．アーリー項と逆ドット項の違いは，項中の位置番号ｊの解釈にある．上記した（３）の定義から明らかなように，逆ドット項では，ドット記号の右の部分がβとして解析済みであるとみなす．項内のＣＦＧ規則の右辺の直後の位置番号がｊであり，そこから先頭方向にｉまで逆戻りした部分$w_{i+1}w_{i+2}\ldots w_j$がβとして解析済みであると解釈する．ちなみにアーリー項では解析対象文のαが解析済みであるとされていた．逆ドット項を作成する利点は３．６で説明する．

3　ＹＡＧＬＲ法

本章ではＴＲＳＳの構造と節点のマージ操作を説明し，次にシフト操作，レデュース操作を説明する．最後にＹＡＧＬＲ法による統語解析アルゴリズムを説明する．

3．1　識別子付き位置番号(Position Number with an Identifier)

ＹＡＧＬＲ法で用いるＴＲＳＳの各節点には，レデュース操作時に逆ドット項，［Ａ→α・β，ｊ］$\in I_i$を作成するために，状態の他に，ｊとｉを得る情報が含まれていなければならない．さらに，マージを効率よく行うために，識別子付き位置番号を導入する．識別子付き位置番号は全解析過程でユニークな番号であり，後述のマージ操作，シフト操作，レデュース操作により，スタックトップの節点が新たに作られる度に作り出され，スタックトップの節点の位置番号にこれを付加する．識別子が付加された位置番号のことを識別子付き位置番号と呼ぶ．位置番号がｉで識別子がｘなら，この識別子付き位置番号を^{x}ｉと書くことにする．識別子はマージの歴史を保存しておくためのもので，それにより重複した統合操作を避けることができる（３．２）．レデュース操作時の逆ドット項の生成には，識別子付き位置番号のなかの，位置番号のみが使われることに注意したい（３．３）．

ＹＡＧＳＳで用いるＴＲＳＳの節点の構造を以下に示す：

［識別子付き位置番号の集合，状態］

たとえば，$\{^{3}2,\ ^{4}3,\ ^{5}4\}$，$\{^{1}2,\ ^{3}2,\ ^{8}6\}$は識別子付き位置番号の集合であり，$[\{^{3}2,\ ^{4}3,\ ^{5}4\},\ 2]$，$[\{^{1}2,\ ^{3}2,\ ^{8}6\},\ 4]$はＴＲＳＳの節点である．

3．2　節点のマージ操作

ＹＡＧＬＲ法は富田法以上にスタックのマージを進めることができる．ＴＲＳＳの構造は富田法で用いるＧＳＳの構造ほど複雑にならない．スタックトップの節点中の識別子付き位置番号の集合には，最後にシフトした語に対する識別子付き位置番号が一つ含まれるだけである．以下では状態が同一の二つの節点のマージ操作について説明するが，これを三つ以上の節点のマージの場合に容易に拡張することができる．（Ｍ１）と（Ｍ２）により，状態が等しい二つの節点をマージして一つの節点（統合節点）にすることができる．

（Ｍ１）スタックトップの節点[[^{x}i],s]と[[^{y}i],s]は，節点［[^{z} i],s］にマージする．但し，zは新たに作り出されたユニークな識別子である．また統合前の二つの節点の持つ全ての子節点を，統合節点の子節点の候補としマージの対象とする．

（Ｍ２）スタックトップ以外の二つの節点[M,s]と[N,s]の場合，節点［M∪N,s]にマージする．ただし，MがNの部分集合なら，節点[N,s]の子節点を統合節点の子節点とする．さもなければ，統合前の二つの節点の持つ全ての子節点を，統合節点の子節点とし，マージの対象とする．前者の場合，統合節点の子節点は節点[N,s]の子節点に等しく，それらの状態は全て異なるので，子節点同士のマージを更に進める必要がない．この事実は，ＴＲＳＳのマージ操作を中途で中断して良いことを意味し，ＴＲＳＳのマージ操作でマージの重複計算を避けることができるので重要である．

なお（Ｍ２）の妥当性については，レデュース操作が関係するので，レデュース操作の説明後に与える．ＴＲＳＳマージ操作アルゴリズムのＰＡＳＣＡＬ風の定義を以下に与える．

［マージ操作のアルゴリズム］

```
procedure main(TRSS: set of trees)
   var current_set: set of nodes;
      begin
         current_set := all the top nodes of trees in TRSS;
         merge(current_set, "top")
      end main;

function merge(current_set: set of nodes, flag: char): char
   var new_current_set: set of nodes;
      begin
         if all the nodes in current_set have distinct states
            then case flag of
                     "top" : "fail";
                     "otherwise" : "success"
                 end
            else begin
                 Form groups of nodes withe the same state in the current_set;
                 For each group of nodes
                     do begin
                           case flag of
                                "top" : apply (M1) repeatedly to get a merged node;
                                "otherwise" : apply (M2) repeatedly to get a merged node
                           end;
                           new_current_set := the children nodes of the merged node;
                           merge(new_current_set,"otherwise")
                        end
                 end
      end  merge;
```

［ＴＲＳＳマージの例］

(a)　----[[^{1}2,^{2}3],8]----[[^{2}3,^{4}4,^{7}5],9]-[[^{8}6],1](ﾄｯﾌﾟ)

(b)　----[[^{1}2,^{2}3,^{6}4],8]-[[^{3}3,^{7}5],9]----[[^{9}6],1](ﾄｯﾌﾟ)

↓（マージ）

(c)　----[[^{1}2,^{2}3,^{6}4],8]-[[^{2}3,^{3}3,^{4}4,^{7}5],9]-[[^{10}6],1](ﾄｯﾌﾟ)

上記したマージ操作のアルゴリズムに従い，(a)と(b)のマージを行う．スタックトップ節点間の統合であるから，(M1)によるマージを行い，統合節点の位置番号６に新しい識別子１０を付加する．次に子節点間のマージを行うが，両者の状態が９で等しいから(M2)によるマージを集の方向に向けて進める．次の孫節点同士も同一の状態８を持つから(M2)によるマージを行う．ただしこの場合，両節点の識別子付き位置番号の集合間に，部分集合関係が成立するので，(M2)により(b)の節点[[^{1}2,^{2}3,^{6}4],8]以下をそのままマージの結果として終了する．こうしてマージのための再計算を避けることができる．富田法ではこうした深いマージは不可能である．

３．３　シフト操作とレデュース操作

［シフト操作］

(a)のスタックトップにシフト操作 "sh u" を行った結果を(b)に示す．それと共に，(c)に示す逆ドット項を作る．シフト操作により，スタックトップの節点の位置番号が１増える．

(a)　----[M,s]－[[^{x}j],t](ﾄｯﾌﾟ)"sh u"

(b)　----[M,s]－[[^{y}(j+1)],u] (ﾄｯﾌﾟ)

(c)　I$_{j}$ ∋　[X → ・w,$_{i}$, j+1]

［レデュース操作］
レデュース操作の指定するＣＦＧ規則 $A \rightarrow X_1 X_2 \cdots X_m$ により，スタックは(a)から(b)に変わると共に，(c)に示す逆ドット項を作成する．

(a) $\cdots\text{-}[N_k, s_k]\text{-}[N_{k+1}, s_{k+1}]\text{-}\cdots\text{-}[N_{k+m}, s_{k+m}]$(ﾄｯﾌﾟ)

↓

(b) $\cdots\text{-}[N_k, s_k]\text{-}[N, t]$(ﾄｯﾌﾟ)

ここで状態 t は，レデュース操作後にＣＦＧ規則の左辺の非終端記号Ａと状態s_kとからＧＯＴＯ表を参照して決めた新しい状態である．また，$N_k = \{^a a, {}^b b, \ldots\}$，$N_{k+1} = \{^c c, {}^d d, \ldots\}$，$\cdots\cdots\cdots$，$N_{k+m-1} = \{^e e, {}^f f, \ldots, {}^g g\}$，$N_{k+m} = \{^j j\}$，$N = \{^z j\}$ であるとする．スタックトップの節点の位置番号の集合N_{k+m}は，最後にシフトした語の識別子付き位置番号$^j j$を含み，レデュース後のスタックトップの節点の位置番号の集合Nは，新しいユニークな識別子付き位置番号$^z j$を含むことに注意したい．これは，レデュース後のスタックのトップに新しい節点が作られるためである．レデュース操作の前後で位置番号 j は変わらないことにも注意したい．

(c) 逆ドット項の作成：

I_a ∋ $[A \rightarrow \cdot X_1 X_2 \cdots X_m, j]$
I_b ∋ $[A \rightarrow \cdot X_1 X_2 \cdots X_m, j]$
……
I_c ∋ $[A \rightarrow X_1 \cdot X_2 \cdots X_m, j]$
I_d ∋ $[A \rightarrow X_1 \cdot X_2 \cdots X_m, j]$
……
I_e ∋ $[A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$
I_f ∋ $[A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$
……
I_g ∋ $[A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$

レデュース操作では，操作の指定するＣＦＧ規則の右辺の非終端記号の数だけ，スタックトップから節点をポップする．ポップする各節点には，ＣＦＧ規則の右辺の非終端記号が対応するので，上記した（１）の場合，X_1，X_2，…，X_mのそれぞれには，スタックの節点，$[N_{k+1}, s_{k+1}]$，$[N_{k+2}, s_{k+2}]$，…，$[N_{k+m}, s_{k+m}]$ が順に対応する．ＴＲＳＳに含まれる節点の位置番号は，どこまでがシフト済み（処理済み）かを示しているので，これを用いて逆ドット項のドット位置（逆ドット項の集合Ｉの添字）を知ることができる．逆ドット項内に書かれる位置番号は全て，スタックトップの節点の持つ位置番号jであり，位置番号に付加された識別子は，逆ドット項の作成には寄与していないことに注意．

３．４　節点のマージ操作（Ｍ２）の妥当性

二つの節点[M, s]と[N, s]に対して（Ｍ２）が成立することは，次の（１），（２），（３）から証明できる．

（１）ＹＡＧＬＲ法では，全てのレデュース操作が終了してからシフト操作を行うため，ＴＲＳＳの全てのスタックトップの節点の持つ位置番号は，レデュース操作中は最後にシフトした語の位置番号に等しく同一である．従って，レデュース操作で作成する逆ドット項中の位置番号はマージ前もマージ後と変わらず，最後にシフトした語の位置番号である．

（２）逆ドット項の集合Ｉの添字は，逆ドット項中のドットの位置番号を表す．この添字は，ＴＲＳＳ中の節点の持つ位置番号の集合から１つずつ取り出されるもので，マージ前の位置番号の集合から個々に取り出しても，マージ後の１つの和集合から取り出しても変わらない．

（３）ＭがＮの部分集合でない場合に，統合前の二つの節点の持つ全ての子節点を，統合節点の子節点として良いことは明らかである．一方，ＭがＮの部分集合である場合に，節点[N, s]の子節点をそのまま統合節点の子節点として良い理由は次の通りである．ＭがＮの部分集合であるということは，節点[M, s]を得るために行ったのと全く同じマージの履歴が，節点[N, s]に残されていることを意味している．従って，節点[M, s]と節点[N, s]以降のマージの結果は，既に節点[N, s]の子節点以降に含まれているので，両節点の統合節点を[N, s]とし，その子節点を，[N, s]の子節点に一致させて良い．

３．５　ＹＡＧＬＲ法による統語解析アルゴリズム

ＹＡＧＬＲ法は以下のアルゴリズムにより入力文を解析する．

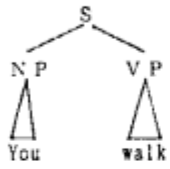
（１）ＴＲＳＳの初期状態を，(ｽﾀｯｸ) | (^{0}0)，0](ﾄｯﾌﾟ)
（２）スタックトップの節点の状態と先読み語の品詞により，ＬＲ表から実行すべき操作を求めて実行する．操作にはシフト操作，レデュース操作，受理，エラーがある．受理なら解析成功として，またエラーなら解析失敗として終了．それ以外は（３）へ．
（３）ＴＲＳＳのスタックに対する操作が全てシフト操作なら，全てのシフト操作実行前と実行後にＴＲＳＳにマージ操作を施し（２）へ，さもなければ（４）へ．
（４）レデュース操作を必要とする一つのスタックＳに対して全てのレデュース操作を施し，その結果えられる新しいスタックと，Ｓ以外の既存のスタックとの間でマージを行い（３）へ．

３．６　なぜ逆ドット項か

レデュース操作を説明したので，ＹＡＧＬＲ法で逆ドット項を作る意義が理解できる．実はＴＲＳＳを利用してレデュース操作時にアーリー項を作ることができるのであるが，ここで逆ドット項を作る理由を述べてみたい．今，以下の(r1)から(r3)に示すＣＦＧを用いて，e You 1 walk 2 in 3 the 4 park 5 の解析を試みて見よう．

(r1) S → NP VP　　(r2) VP → V PP　　(r3) VP → V

解析が進み，"You walk" までをシフトした段階では，規則(r1)を用いたレデュース操作を行い，図３．１に示すアーリー項を作ることができる．解析がさらに右に進み，"in the park" がＰＰ（前置詞句）として受理され，規則(r2)により "walk in the park" がＶＰとして受理されたとしよう．すると再び規則(r1)を用いたレデュース操作を行い図３．２に示すアーリー項を作ることができる．

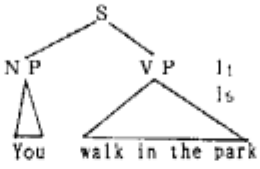


アーリー項：

I_1 ｓ［S→ＮＰ・ＶＰ，0］

I_2 ｓ［S→ＮＰ ＶＰ・，0］

図３．１　部分統語解析木とアーリー項（１）



アーリー項：

I_1 ｓ［S→ＮＰ・ＶＰ，0］

I_5 ｓ［S→ＮＰ ＶＰ・，0］

図３．２　部分統語解析木とアーリー項（２）

ここで，項の集合i_1に所属させようとするアーリー項は，I_1の節点として既に登録済みのアーリー項と同じで重複することに注意したい．このことは，項の集合への節点の重複登録を避けるために，過去に作成したアーリー項の集合を調べる必要があることを意味している．従って過去に作成したアーリー項の全集合が，任意の時点で検索可能でなければならない．

一方逆ドット項を作成する場合には，逆ドット項中の位置番号は，レデュース操作時のスタックトップの節点のもつ位置番号である．この位置番号は，新たなシフト操作がなされぬ限り変わらないだけでなく，過去のシフト操作時の位置番号とも異なる．そのため逆ドット項の重複検査は，現在実行中の全てのレデュース操作により作られる項の集合を調べるだけでよい．この様にして，重複する逆ドット項を検査する範囲を局所化することが可能になり，容易にしかも高速な重複処理を行うことができる．新たなシフト操作を行う度に，重複検査のために，それまでに作られた逆ドット項を参照する必要はないのである．上記例の場合，逆ドット項なら以下が生成され，いずれも互いに重複しない．

図３．１に対応した逆ドット項：　I_1 ｓ［S→ＮＰ・ＶＰ，２］

I_2 ｓ［S→・ＮＰ ＶＰ，２］

図３．２に対応した逆ドット項：　I_1 ｓ［S→ＮＰ・ＶＰ，５］

I_6 ｓ［S→・ＮＰ ＶＰ，５］

これまでの説明から明らかなように，ＹＡＧＬＲ法では，ＰＰ付加規則に対して，アーリー項なら重複した項生成が行われるが，逆ドット項なら重複した生成が行われない．一般に，重複する項の生成は，文法規則に依存するので，逆ドット項の生成が有利かアーリー項の生成が有利かは一概には決められない．しかし，逆ドット項は重複した項の検査を局所化することができることを説明した．これは，ＬＲ法が最右導出をベースとしていることと関連している．

３．７　富田法とＹＡＧＬＲ法による統語解析例

本節では図３．３に示すＣＦＧを用いて，富田法とＹＡＧＬＲ法のそれぞれに対する統語解析例を示し，両者の比較を行う．解析対象文は「文化 が きた から 伝わった から...」であるとし，解析中途まで双方のトレースを行う．図３．３のＣＦＧから図３．４に示すＬＲ表が得られる．なお，富田法のスタックの構造は，［部分木番号，状態］である．

（１）　S→ＰＰ　S　　（５）　v→きた

（２）　S→v　　（６）　v→伝わった

（３）　ＰＰ→n　p　　（７）　n→きた

（４）　ＰＰ→S　p　　（８）　n→文化

（９）　p→から

（１０）　p→が

図３．３　簡単な日本語ＣＦＧ

状態	アクション部				GOTO部	
	n	p	v	$	ＰＰ	S
0	sh4		sh3		2	1
1		sh5		受理		
2	sh4		sh3		2	6
3		re2		re2		
4		sh7				
5	re4		re4			
6		sh5 re1		re1		
7	re3		re3			

図３．４　図３．３のＣＦＧから得られたＬＲ表

［富田法による解析］

T1)　[_,0]-

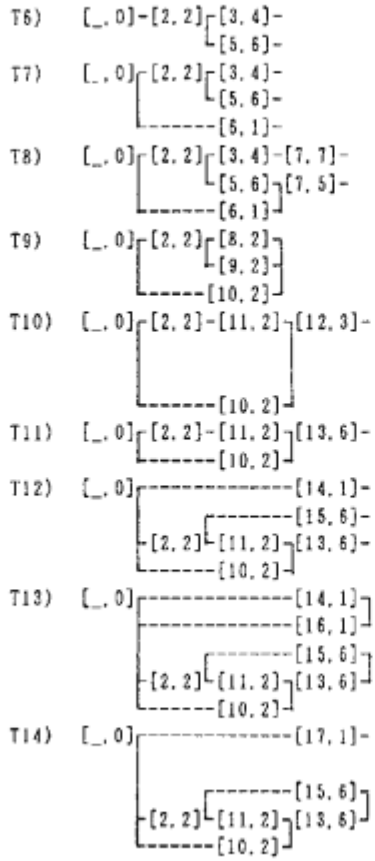
T2)　[_,0]-[0,4]-

T3)　[_,0]-[0,4]-[1,7]-

T4)　[_,0]-[2,2]-

T5)　[_,0]-[2,2]┌[3,4]-
　　└[4,3]-

文化(n) sh4 ;
　が(p) sh7 ;　0 [n 文化]
きた(n;v) re3 ;　1 [p が]
　きた(n) sh4 ;　2 [PP (0 1)]
　きた(v) sh3 ;
　から(p) sh7 ;　3 [n きた]
　　re2 ;　4 [v きた]



から(p) sh7		
sh5,rel	5	[S (4)]
から(p) sh7		
sh5		
sh5	6	[S (2 5)]
伝わった(v) re3	7	[p から]
re4	sh5 (at T8)によるマージ	
伝わった(v) sh3	8	[PP (3 7)]
	9	[PP (5 7)]
	10	[PP (6 7)]
から(p) re2	マージ（圧縮統括森11）	
	sh3 (at T9)によるマージ	
	11	[PP (3 7) (5 7)]
	12	[v 伝わった]
から(p) sh5,rel	13	[S (12)]
から(p) sh5	14	[S (10 13)]
sh5,rel	15	[S (11 13)]
sh5		
から(p) sh5	16	[S (2 15)]
sh5		
から(p) sh5	マージ（圧縮統括森17）	
	17	[S (2 15) (10 13)]
sh5		

（以下省略）

［ＹＡＧＬＲ法による解析］

Y1) [(^{0}0),0]-

Y2) [(^{0}0),0]-[(^{1}1),4]-

Y3) [(^{0}0),0]-[(^{1}1),4]-[(^{2}2),7]-

Y4) [(^{0}0),0]-[(^{3}2),2]-

Y5) [(^{0}0),0]-[(^{2}2),2]-[(^{4}3),4]-
[(^{0}0),0]-[(^{3}2),2]-[(^{5}3),3]-

Y6) [(^{0}0),0]-[(^{3}2),2]-[(^{4}3),4]-
[(^{3}0),0]-[(^{3}2),2]-[(^{6}3),6]-

Y7) [(^{0}0),0]-[(^{3}2),2]-[(^{4}3),4]-
[(^{0}0),0]-[(^{3}2),2]-[(^{6}3),6]-
[(^{0}0),0]----------[(^{7}3),1]-

Y8) [(^{0}0),0]-[(^{3}2),2]-[(^{4}3),4]-[(^{9}4),7]-

[(^{0}0),0]-[(^{3}2),2]-[(^{6}3),6]┐[(^{9}4),5]-

[(^{0}0),0]-[(^{7}3),1]----------┘

Y9) [(^{0}0),0]-[(^{3}2),2]┐[(^{11}4),2]-
[(^{0}0),0]----------┘

Y10) [(^{0}0),0]-[(^{3}2),2]┐[(^{11}4),2]-[(^{12}5),3]-
[(^{2}0),0]----------┘

Y11) [(^{6}0),0]-[(^{3}2),2]┐[(^{11}4),2]-[(^{13}5),6]-
[(^{0}0),0]----------┘

Y12) [(^{0}0),0]-[(^{3}2),2]┐[(^{11}4),2]-[(^{13}5),6]-
[(^{0}0),0]----------┘
[(^{0}0),0]-[(^{3}2),2]----------[(^{14}5),6]-

[(^{0}0),0]--------------------[(^{15}5),1]-

文化(n) sh4	10	[n→・文化,1]
が(p) sh7	11	[p→・が,2]
きた(n;v) re3	10	[PP→・n p,2]
	11	[PP→n・p,2]
きた(n) sh4	12	[n→・きた,3]
きた(v) sh3	12	[v→・きた,3]
から(p) sh7		
re2	12	[S→・v,3]
から(p) sh7		
sh5,rel	10	[S→・PP S,3]
	12	[S→PP・S,3]
から(p) sh7	13	[p→・から,4]
sh5		
sh5	マージ	
伝わった(v) re3	12	[PP→・n p,4]
	13	[PP→n・p,4]
re4	12	[PP→・S P,4]
	13	[PP→S・P,4]
	10	[PP→・S P,4]
	13	[PP→S・P,4]*
伝わった(v) sh3	15	[v→・伝わった,5]
から(p) re2	14	[S→・v,5]
から(p) sh5,rel	12	[S→・PP S,5]
	14	[S→PP・S,5]
	10	[S→・PP S,5]
から(p) sh5		
sh5,rel	10	[S→・PP S,5]*
	12	[S→PP・S,5]
sh5		

```
Y13) [[10]0],0]-[[[3]2],2]-[[[3]2,[11]4],2]-[[[17]5],6]-      から(p)  sh5 ┆マージ
     [[10]0],0]----------┘                                              ┆
     [[3]0],0]-----------------------[[[18]5],1]-                  sh5
(以下省略)                                                              注）＊付きの項は，重複項を表す．
```

ここで富田法とＹＡＧＬＲ法の比較をして見たい．Ｔ１２）からＴ１４）までとＹ１１）からＹ１３）までを比較してみよう．ＹＡＧＬＲ法は，ＴＲＳＳの構造が単純な分だけ，実装が容易である．また，ＹＡＧＬＲ法は富田法以上にスタックのマージを進めることができるので，ＧＳＳを用いずとも計算効率と計算時に使用する記憶空間の縮小をはかることが可能になり，解析結果に含まれる曖昧性が増大するにつれて，ＹＡＧＬＲ法と富田法の違いが際だってくる．４章ではその一端を説明する．

4　解析対象文の長さに対する計算の複雑性

Johnsonらによれば，次の一連の文法Ｌnに対して，解析対象文の長さをn+2としたとき，解析に要する時間はΩ(n^n)である［Johnson ８９］［Kipps ８９］．

（１）Ｓ→a
（２）Ｓ→ＳＳ
（３）Ｓ→Ｓ$^{n+2}$

ただしＳ$^{n-2}$ は記号Ｓがn+2個連結したものの省略形である．解析対象文をa^{n-2}Ｓとする（ただしn>m）．上記した規則に対して右のＬＲ表が得られる．今，ｉ（n+3<i<n+2）番目のａをシフトしrelを施した後のＴＲＳＳの状態を図４．１に示す．ここでＮ$_{p,q}$は節点を表し，各節点の持つ識別子付き位置番号の集合間には，矢印で示す包含関係がある．節点ＸとＹの間にＸ→Ｙがあれば，Ｘの識別子付き位置番号の集合がＹの部分集合であることを示す．

容易に示すことができるように，節点Ｎ$_{p,q}$の持つ状態Ｔ$_{p,q}$は，

$$T_{p,q} = \begin{cases} q & if \quad 1 < q \leq m+3, \\ m+3 & if \quad m+3 < q, \\ 0 & if \quad q-1. \end{cases}$$

状態	アクション部		GOTO部
	a	S	S
0	sh1		2
1	rel	rel	
2	sh1	acc	3
3	sh1/re2	re2	4
4	sh1/re2	re2	5
....	....	....	....
m+1	sh1/re2	re2	m+2
m+2	sh1/re2	re2	m+3
m+3	sh1/re2/re3	re2/re3	m+3

図４．１に対して，次のシフト操作を施すまで，レデュース操作を次々に施す必要があるが，そのための計算コストを考えてみよう．まず，規則（２）を適用してレデュース操作を進める場合の方が，規則（３）を適用する場合に比べてコストがかかることは明らかである．したがって，計算コストの

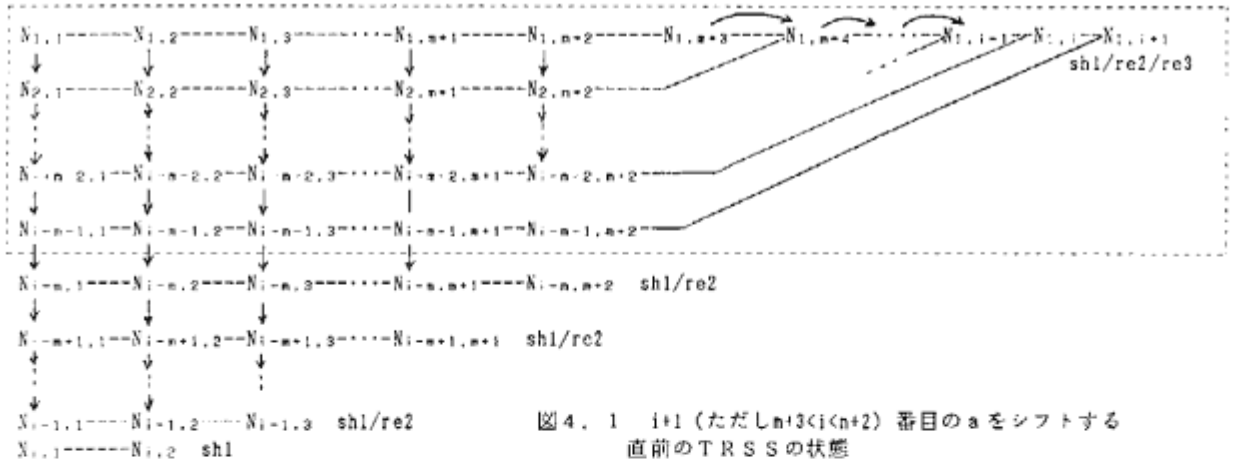
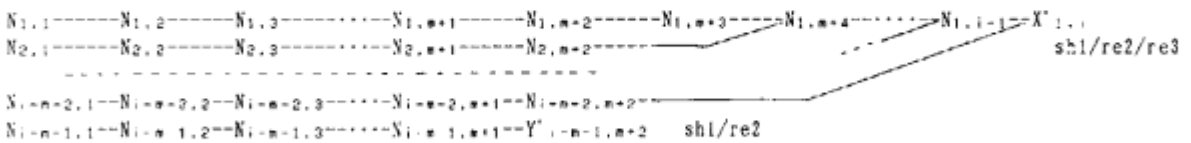


図４．１　i+1（ただしm+3<i<n+2）番目のａをシフトする直前のＴＲＳＳの状態

オーダを求めるためには，規則（２）を適用するレデュースを考えれば十分である．また，最上部の点線で囲まれたＴＲＳＳの部分のみが，入力文の長さに依存しているので，これに対するレデュース操作の計算コストを考えれば十分である．今，図４．１の最上部のスタックに対して，規則（２）によるレデュース操作を一度施すと，下図の二つのスタック構造が得られる．



上図のスタックは，図４．１の統合化可能なスタックとマージされる．たとえば，sh1を待つ最長スタック同士のスタックトップの状態はn+3で等しいのでマージされる．このとき，図４．１に記述した節点間の識別子付き位置番号の集合間に，包含関係が成り立つため，マージ操作は先頭から二つの目までの節点同士（<Ｎ$_{1,i-1}$とＮ$_{1,i}$>，<Ｎ$_{1,i}$とＮ$_{2,i-1}$>，<Ｎ$_{i-m-1,m+2}$とＮ$_{i-n-2,n+2}$>）で行えばよい．先頭から３番目の節点同士（<Ｎ$_{1,i-1}$とＮ$_{1,i-2}$>または<Ｎ$_{i-m,m+1}$とＮ$_{i-n-2}$

,n_{-1}>）のマージなどは，識別子付き位置番号の集合間に部分集合関係が成立しているので，それ以上マージを葉の方向に向けて進める必要はない．

以上の考察から，re2操作後のマージには先頭から３番目のマージまでを考慮すればよいことが分かる．しかも，それぞれのマージには，識別子付き位置番号の集合の要素が高々i+1含まれるだけであるから，マージに要す計算コストはiのオーダである．一方，re2は繰り返し行われるが，その回数は高々j回である．したがって，レデュース操作とマージ操作の総和は，$\Omega(i^2)$である．以上より，a^{n-2}の入力文を解析するために要する総計算コストのオーダは，$\Sigma_{i=2}^{n+2}\Omega(i^2)=\Omega(n^3)$となり，証明された．

次に図４．１の各節点中の識別子付き位置番号の数は：節点$N_{i,j}$に対して，$2\leq j\leq m+3$の場合１，$m+4\leq j\leq i-1$の場合２，$i\leq j\leq i+1$の場合１，節点$N_{k,j}$($2\leq k\leq i-m-1$，$2\leq j\leq m+2$)に対してｋである．したがって，点線で囲まれた部分の節点中の識別子付き位置番号の総和は，1*(n+4)+2*(i-m-4)+(n+2)(2+3+...+(i-m-1))である．この総和のオーダはi^2である．これは，点線で囲まれた部分以外の節点に含まれる識別子付き位置番号の総和のオーダーと等しい．以上から，入力文a^{n+2}を解析するために必要なＴＲＳＳの空間のオーダはn^2であり，また，作成される逆ドット項の数の総和のオーダーは，n^3であることが証明された．

5　おわりに

本論文ではＹＡＧＬＲ法と呼ぶ新しい統語解析アルゴリズムを提案した．ＹＡＧＬＲ法は，既存の方法と比べていくつかの利点を持っている．

(A) Johnsonらの与えた特殊な文法に対しても，解析時間のオーダは，解析対象文の長さｎに対してアーリー法と同等のn^3のオーダに抑えることができる．
(B) レデュース操作時に，アーリー項と対称な逆ドット項を作成しながら統語解析を行う．逆ドット項を生成する利点について考察した．
(C) 先読み情報を用いているので，最終的に生成される逆ドット項の数はアーリー法より少ない．
(D) 富田法以上にスタックのマージを行うことができるため，グラフ構造化スタックより構造が単純で操作し易い木構造化スタックを用いることができる．それによりスタック操作が単純になる．
(E) 解析終了後に，作成した逆ドット項を組み合わせて統語解析木を作る．

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．

（１）ＹＡＧＬＲ法の統語解析時間のオーダが，一般のＣＦＧに対してn^3のオーダであることの証明．
（２）ＹＡＧＬＲ法による統語解析に要する空間量の評価．
（３）ＹＡＧＬＲ法の並列解析アルゴリズムの研究．
（４）逆ドット項からの統語構造の並列抽出アルゴリズムの研究．

（１）は重要な研究課題であるが，（２）に述べたように，解析に要す空間の大きさの評価も行う必要がある．Kippsは富田法に比較的単純な修正を施すことで，一般のＣＦＧに対する計算時間のオーダを$\Omega(n^3)$に抑えることができることを示した［Kipps　８９］．しかし，このアルゴリズムは富田法以上の空間を要す．これに対してＹＡＧＬＲ法では，スタックのマージを徹底的に行うアルゴリズムであるから，計算空間を抑えることができる．一般のＣＦＧに対するＹＡＧＬＲ法の使用記憶空間の大きさの詳細な検討を行う必要がある．

（３）に述べたように，富田法と同様，ＹＡＧＬＲ法の並列解析アルゴリズムを検討することも意味があるだろう［Tanaka　８９］．本稿で述べたアルゴリズムは横型探索の逐次型アルゴリズムであり，できるだけ無駄な解析を省いたアルゴリズムである．この長所を生かしつつ，シフトレデュース・コンフリクトが発生したときの待ち合わせができるだけ少なくなる並列解析アルゴリズムを研究する必要がある．それには沼崎らの研究が参考になろう［Numazaki　９０］．

ＹＡＧＬＲ法では，統語解析後に得られた逆ドット項の集合から統語解析木を作り出さねばならない．これはアーリー法と双対なアルゴリズムを用いれば良い．一つの統語解析木を計算するための時間はn^2のオーダであることが知られている［Aho　７２］．この統語解析木の計算にも多くの並列性が観察されるので，統語解析木抽出用の並列アルゴリズムについて今後研究する必要がある．

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　８９］　Johnson, M.: The Computational Complexity of Tomita's Algorithm, International Parsing Workshop '89, Carnegie-Mellon University, 203-208(1989).

［Kipps　８９］　Kipps, J. R.: Analysis of Tomita's Algorithm for General Context-Free Parsing, International Parsing Workshop '89, Carnegie-Mellon University, 193-202(1989).

［Numazaki　９０］　Numazaki, H. and Tanaka, H.: A new Parallel Algorithm for Generalized LR Parsing, COLING'90 305-310(1990).

［Suresh　９０］　Suresh, K.G.: Yet Another Generalized LR Parser-Report4, Tanaka Lab., Tokyo Inst. of Technology (1990.4.16)

［田中　８９］　田中穂積：自然言語解析の基礎，産業図書（１９８９）．

［Tanaka　８９］　Tanaka, H. and Numazaki, H.: Parallel Generalized LR Parser Based on Logic Programming, 1st Australian-Japan Joint Symposium on Natural Language processing, 201-211(1989).

［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).