自然言語処理 83－11
（1991. 5. 17）

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

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．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，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．

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し，ＹＡＧＬＲ法が高速な統語解析法であることを実証する．

ＹＡＧＬＲ：Yet Another Generalized LR Parser

Hozumi TANAKA,　K. G. SURESH

Department of Computer Science
Tokyo Institute of Technology
2－12－1　Ookayama Meguro－ku Tokyo 152,　Japan

We have developed a new generalized LR parsing algorithm called YAGLR. The parsing algorithm uses a set of tree-structured stacks which is compact as well as graph-structured stack of Tomita's algorithm. The reason is that YAGLR does effective merge operations on the tree-structured stack. We also present the experimental results from which we can reveal that the parsing time of YAGLR is faster than that of Tomita's. Even for the special CFGs for which Tomita's algorithm does not fare well, YAGLR's parsing time is in the order of $O(n^3)$. We conclude that YAGLR has both advantages of Earley's and Tomita's algorithm.

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^3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 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，理論的に，富田法はアーリー法以上の解析時間を要すことになる．

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（付録Ｂ参照）．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（付録Ａ参照）．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trss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，アーリー項と逆ドット項を作成するための木構造化スタックを説明し，逆ドット項を作成することにより，木構造化スタックのより深いマージが可能になることを説明する．逆ドット項を作成するその他の利点についても述べる．３章では，逆ドット項の形式的な定義を与える．４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．５章では，富田法とＹＡＧＬＲ法による解析過程をトレースする．６章では，実験結果を示しＹＡＧＬＲ法の高速性を実証する．７章ではまとめと今後の研究課題について述べる．

2．　アーリー項と逆ドット項

本章では，スタックの節点に位置番号をもたせることにより，レデュース操作時にアーリー項と，逆ドット項を作ることができることを説明する．逆ドット項はＹＡＧＬＲ法で用いる項で，アーリー項と対称な形をしている．アーリー項を作る場合は，富田法以上のスタックのマージが可能になるが，逆ドット項を作る場合は，アーリー項を作る以上のスタックのマージが可能なことを説明する．

さて，入力文 w が $w=w_1w_2...w_n$ であるとしよう．ここで単語w_iとw_{i+1}の間に番号iをふり，これを位置番号と呼ぶことにする．スタックの節点に，部分解析木の代わりにこの位置番号をもたせることにより，レデュース操作時にアーリー項と逆ドット項を作ることができる．（ａ）に示すスタックを考えよう．

（ａ）・・・-[{3},s_3]-[{5},s_2]-[{6},s_1]　(top)　re,x

ここでs_1, s_2, s_3は状態を，{6}, {5}, {3}は位置番号の集合を表す．これらの位置番号は，これらを含む節点の直接右に振られていると考える．従って，節点[{6},s_1]には入力文のw_6が対応し，節点[{5},s_2]にはw_4w_5が対応する．以上のことから，（ａ）のレデュース操作"re,x"で用いる規則"x"が，Ａ→Ｂ　Ｃであるとすると，このレデュース操作によりトップから二つの節点をポップし，（ｂ）に示すアーリー項を作ることができることが分かる．

（ｂ）アーリー項：I_5 ∋ [A→B・C,3], I_6 ∋ [A→B C・,3]

最初のアーリー項の内部にある数字３と・記号は，位置番号３から５の間にある入力文の部分（w_4w_5）がＢとして認識されたことを表す．同様に，二番目のアーリー項は位置番号３から６の間にある入力文の部分（$w_4w_5w_6$）が"Ｂ　Ｃ"として認識され，それがＡにまとめられることを意味している．一方，（ａ）から（ｃ）に示す逆ドット項を作ることもできる．

（ｃ）逆ドット項：I_5 ∋ [A→B・C,6], I_3 ∋ [A→・B C,6]

（ｃ）の項中の数字の６は，スタックトップの節点の位置番号である．最初の逆ドット項は，アーリー項とは逆に，位置番号６から５の間にある入力文の部分（w_6）がＣとして認識されていることを，また二番目の逆ドット項は，位置番号６から３の間にある入力文の部分（$w_4w_5w_6$）が"Ｂ　Ｃ"として認識され，それがＡにまとめられることを意味している．この様に，逆ドット項では，アーリー項とは逆にＣＦＧの右辺の右端からドットの位置までが解析済みであると見なす．逆ドット項はアーリー項と対称な項なのである．

ここで，（ｄ）に示す木構造化スタックを考えよう．これは富田法と同様，スタックトップの節点の状態がs_1で同一であるので，以後の処理を共通して行うためにトップを一本化している．

（ｄ）・・・-[{3},s_3]-[{5},s_2]-[{6},s_1]┐　(top)　re,x
　　===-[{2},s_4]-[{4},s_2]-[{6},s_1]┘

先の規則"x"により，"re,x"を行い，（ｅ）と（ｆ）に示すアーリー項と逆ドット項を作ることができる．

（ｅ）アーリー項：I_5 ∋ [A→B・C,3],
　　I_6 ∋ [A→B C・,3],
　　I_4 ∋ [A→B・C,2],
　　I_6 ∋ [A→B C・,2]

（ｆ）逆ドット項：I_5 ∋ [A→B・C,6],
　　I_3 ∋ [A→・B C,6],
　　I_4 ∋ [A→B・C,6],
　　I_2 ∋ [A→・B C,6]

（ｄ）の代わりに，状態s1を持つ二つのトップの節点を一つにマージした（ｇ）を考えてみよう．

（ｇ）・・・-[{3},s_3]-[{5},s_2]┐[{6},s_1]　(top)　re,x
　　===-[{2},s_4]-[{4},s_2]┘

（ｇ）に示すスタック節点のマージは富田法では許されないことに注意しよう．一段深いマージを行った（ｇ）からも，（ｅ）と（ｆ）に示すアーリー項と逆ドット項が作れることは明かである．スタックトップの次の節点も同一の状態s_2を持つので，更に一段深くマージを進めた（ｈ）を考えてみよう．ただし，マージ後の節点の位置番号の集合は，マージ前の節点の持つ位置番号の集合の和とする．

（h）・・・-[{3},s_3]-[{4,5},s_2]-[{6},s_1]　(top)　re,x
　　===-[{2},s_4]

すると，（h）に"re,x"を施した結果得られるアーリー項には（e）に示すもの以外に，次の二つの余分な項が含まれる：I_4 ∋ [A→B・C,3]とI_5 ∋ [A→B C・,2]である．しかし幸いなことに，（h）から得られる逆ドット項は（f）と全く同一である．これはＬＲ法が最右導出をベースにしていることによるのだが，ここではそれを指摘しておくに留める．いずれにしても逆ドット項を作成することにより，木構造化スタックの深いマージが可能になり，スタックの構造が簡単になり記憶の節約になるだけでなく，スタック操作がグラフ構造化スタックと比べて簡便になるという利点がある．ＹＡＧＬＲ法で，アーリー項の代わりに逆ドット項を作成する最大の理由はここにある．言うまでもなく，このようなスタックの深いマージは，富田法では不可能である．

最後に（e）に示す逆ドット項の内部の数字は全て６であることに注意したい．これはスタックトップの節点の位置番号であり，次のシフト操作が行われるまで変わらないので（４．１．４参照），重複した逆ドット項の検査は，現在のシフト操作から次のシフト操作が行われるまでの間のレデュース操作中に作られた逆ドット項を調べるだけでよい．言い換えると，調べる範囲を局所化することが可能になる．これが逆ドット項作成の第２の利点である．

実際木構造化スタックを用いた富田法と比べて，解析結果の曖昧性が増えるにつれて急激に高速な統語解析が可能であることが実証されている（６章）．

３．　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$w=w_1w_2\ldots w_n\ \varepsilon\ T^*$ 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$A\rightarrow\alpha\gamma$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，［$A\rightarrow\beta\cdot\gamma$，ｊ］は，ｗに対する逆ドット項である．特に，［$A\rightarrow\alpha\cdot$，ｊ］または［$A\rightarrow\alpha\cdot$，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，

［$A\rightarrow\alpha\cdot\beta$，ｊ］$\varepsilon I_i$　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．

４　ＹＡＧＬＲ法

本章では木構造化スタック（以下ではtrssと呼ぶ）の構造と，trss同士のマージ操作，trssに対するシフト操作とレデュース操作を説明し，ＹＡＧＬＲ法の手続きを示す．

４．１　マージ，シフト，レデュース

４．１．１　識別子付き位置番号

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，［$A\rightarrow\alpha\cdot\beta$，ｊ］$\varepsilon I_i$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ことを２章で説明した．２章では説明の簡単のために，trssの各節点には位置番号の集合が付加されていた．しかし，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し，これを付加することにしよう．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，このスタックトップの節点の位置番号に付加される．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（４．１．２，４．１．３）．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内，位置番号のみが使われることに注意したい（４．１．５）．ＹＡＧＬＲ法で用いるtrssの節点の構造は以下の通りである：

［識別子付き位置番号の集合，状態］

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

４．１．２　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ことを２章で説明した．trssの構造は富田法で用いるＧＳＳの構造ほど複雑にはならない．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（Ｍ１）と（Ｍ２）について説明する．これを基本に，三つ以上の節点のマージも行うことができる．

（Ｍ１）スタックトップの節点$[\{^x i\},s]$と$[\{^y i\},s]$をマージして，一つの統合節点 $[\{^z i\},s]$ にすることができる．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，統合節点の子節点とし，その後のマージの対象とする．

（Ｍ２）スタックトップ以外の節点[M,s]と[N,s]をマージして，一つの統合節点 [M∪N,s]にすることができる．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，その後の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同士のマー

ジ操作を中途で中断して良いことを意味しているので重要である．
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，レデュース操作を説明した後で与える．trss同士のマージ操作アルゴリズムのＰＡＳＣＡＬ風の定義を以下に与える．

４．１．３　マージ操作のアルゴリズム

```
procedure merge(current_set: set of trsses, flag: char)
   var new_current_set: set of trsses;
   {after this merge, current_set will be changed into the result of merge}
   begin
     if all the top nodes in current_set have distinct states
        then return
        else begin
             while current_set is not empty do
               begin
                 from current_set, pickup and retract all the trsses with the same state
                 in the top node and name them TRSS';
                 case flag of "top": apply (M1) repeatedly to merge all root nodes of TRSS' into one;
                              "otherwise": apply (M2) repeatedly to merge all root nodes of TRSS' into one
               end;
             end;
           new_current_set := a set of all the immediate decendent subtrees of the newly merged
                              root node of TRSS';
           merge(new_current_set, "otherwise")
           end
   end merge;
```

trss同士のマージ例:

(a)　$\cdots$-[{^{1}2, ^{3}3}, 8]----[{^{2}2, ^{6}4, ^{7}5}, 9]-[{^{8}6}, 1](ﾄｯﾌﾟ)
(b)　===-[{^{1}2, ^{3}3, ^{5}4}, 8]-[{^{4}3, ^{7}5}, 9]----[{^{9}6}, 1](ﾄｯﾌﾟ)
　　↓（マージ）
(c)　===-[{^{1}2, ^{3}3, ^{5}4}, 8]-[{^{2}2, ^{4}3, ^{6}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)のマージを行うと(c)を得る．スタックトップの節点同士の統合から始め，(M1)による統合節点を作り，新しい識別子付き位置番号^{10}6を導入する．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)によるマージをさらに進める．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が，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，マージ操作をここで止め，(M2)により(b)の節点[{^{1}2, ^{3}3, ^{5}4}, 8]以下をそのままマージの結果とする．富田法ではこうした深いマージは不可能であることに注意したい．

４．１．４　シフト操作

ＬＲ表から，先読み語（解析対象文のj+1番目の語w_{j+1}）の品詞がＣでシフト操作 "sh u" を(a)のtrssに施すべきことが分かったとしよう．このシフト操作を(a)に施すと(b)に示すtrssがえられる．それと共に(c)に示す逆ドット項を作る．シフト操作後のスタックトップの節点の位置番号はj+1になる．

(a)　$\cdots$-[M, s]－[{^{x}j}, t](ﾄｯﾌﾟ)　"sh u"
(b)　$\cdots$-[M, s]－[{^{x}j}, t]-[{^{y}(j+1)}, u] (ﾄｯﾌﾟ)
(c)　I_j ∋　[C→・w_{j+1}, j+1]

４．１．５　レデュース操作

trssに対するレデュース操作（ＣＦＧ規則 $A \rightarrow X_1 X_2 \cdots X_m$ を用いる）により，trssの一つのパス(a)は(b)に変わると共に，(c)に示す逆ドット項を作成する．

(a)　$\cdots$-[N_k, s_k]-[N_{k+1}, s_{k+1}]-$\cdots$-[N_{k+m}, s_{k+m}](ﾄｯﾌﾟ)
　　↓
(b)　$\cdots$-[N_k, s_k]-[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={^{n}a, ^{o}b, ...}，N_{k+1}={^{p}c, ^{q}d, ...}，$\cdots\cdots\cdots$，N_{k+m-1}={^{v}e, ^{w}f, ..., ^{x}g}，N_{k+m}={^{y}j}，N={^{z}j}であるとする．スタックトップの節点の識別子付き位置番号の集合N_{k+m}には，最後にシフトした語の位置番号jに対する識別子付き位置番号^{y}jのみが含まれ，レデュース後の新しいスタックトップの節点の識別子付き位置番号の集合Nには，新しいユニークな識別子付き位置番号^{z}jのみが含まれることに注意したい．レデュース操作の前後で位置番号jは変わらない．

(c)　逆ドット項の作成：

I_a ∋ [A→・$X_1X_2 \cdots X_m$, j]　　$\cdots\cdots\cdots\cdots\cdots\cdots$
I_b ∋ [A→・$X_1X_2 \cdots X_m$, j]　　I_e ∋ [A→$X_1X_2 \cdots$ ・X_m, j]
$\cdots\cdots\cdots\cdots\cdots\cdots$　　I_f ∋ [A→$X_1X_2 \cdots$ ・X_m, j]
I_c ∋ [A→X_1・$X_2 \cdots X_m$, j]　　$\cdots\cdots\cdots\cdots\cdots\cdots$
I_d ∋ [A→X_1・$X_2 \cdots X_m$, j]　　I_g ∋ [A→$X_1X_2 \cdots$ ・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，(a)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．(a)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関与していないことに注意されたい．

4．1．6　節点のマージ操作（M2）の妥当性

節点のマージ操作（M1）が成り立つことは自明であろう．二つの節点[M,s]と[N,s]に関して（M2）が成立することは，次の（1）から（3）で証明できる．以下では，trssの集合をTRSSと呼ぶことにする．

（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．シフト操作を行わない限りTRSSの全てのスタックトップの節点の持つ位置番号は，最後にシフトした語の位置番号に等しく同一でなければならない．（それにより，レデュース操作で作成する全ての逆ドット項中の位置番号は，最後にシフトした語の位置番号であり変わらない）．

（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，レデュース操作の対象となる節点の持つ識別子付き位置番号の集合から1つずつ取り出されるもので，マージ前の二つの節点の識別子付き位置番号の集合から別々に取り出しても，マージ後の，識別子付き位置番号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，統合節点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，[N,s]の子節点をそのまま統合節点の子節点としてよい．

4．2　YAGLR法の手続き

YAGLR法による入力文解析の手続きを以下に示す．

（1）trssの集合TRSSの初期状態を，(ｽﾀｯｸA) [[{0},0]] (ﾄｯﾌﾟ)とする．

（2）TempStack:= [] .

（3）TRSSが空（[]）なら（5）へ，
　　さもなければ，TRSSからtrssを一つ取り出し（TRSS:=TRSS-trss）以下を行う．
　　LR表によりtrssに対する'操作'を決める．
　　もし'操作'が受理の場合：成功として（3）へ戻る，
　　誤りの場合：失敗として（3）へ戻る，
　　シフトの場合：trssをTempStackに入れて（3）へ戻る，
　　レデュースの場合：（4）へ，
　　シフト／レデュースの場合：シフト操作とレデュース操作を切り離し，シフト操作付きのtrssをTempStackに入れ，レデュース操作付きのtrssに対して（4）へ．

（4）trssに対して指定のレデュース操作を行い，その結果新たに作られるtrss'をTRSSに入れる，
　　merge(TRSS,"top")を実行後（3）へ戻る．

（5）TempStackが空なら終了，
　　さもなければ，TempStack中の各trssにシフト操作を施し，結果をTRSSに入れ，
　　merge(TRSS,"top")を実行後（2）へ戻る．

以上の手続きでは，シフト・レデュース・コンフリクトが生じたとき，富田法と同様，レデュース操作を優先させていることが分かる．

5．　富田法とYAGLR法による統語解析例

本章では図5．1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5．1のCFGから図5．2に示すLR表が得られる．なお，富田法のスタックの節点の構造は，［部分木番号，状態］である．

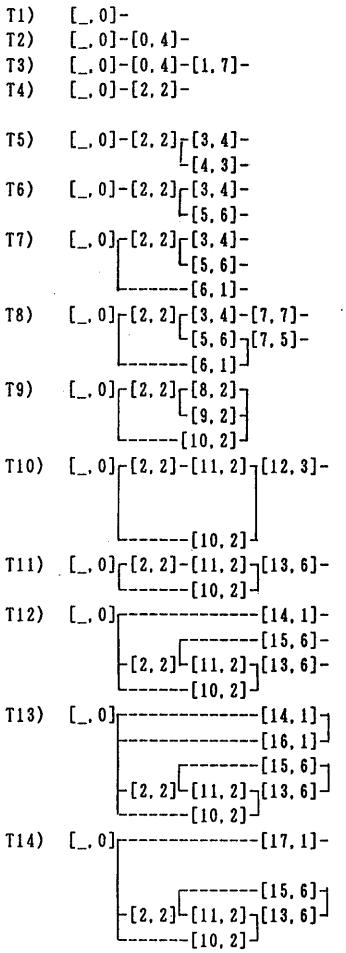
（1）　S→PP　S　　（5）　v→きた
（2）　S→v　　（6）　v→伝わった
（3）　PP→n　p　　（7）　n→きた
（4）　PP→S　p　　（8）　n→文化
　　（9）　p→から
　　（10）　p→が

図5．1　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			

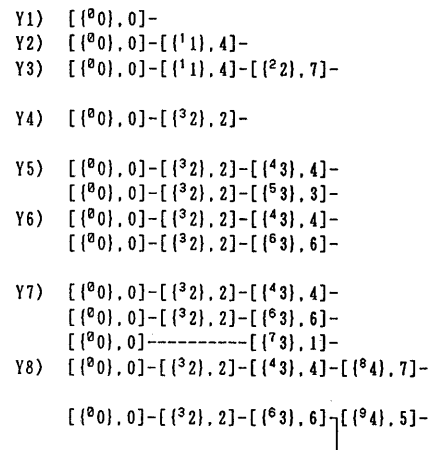
図　5．2　図5．1のCFGから得られたLR表

5．1　富田法による解析



（以下省略）

5．2　ＹＡＧＬＲ法による解析



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,re1	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（圧縮統語森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,re1	13 [S (12)]
から(p)	sh5	14 [S (10 13)]
	sh5,re1	15 [S (11 13)]
	sh5	
から(p)	sh5	16 [S (2 15)]
	sh5	
から(p)	sh5	マージ（圧縮統語森17）
		17 [S (2 15) (10 13)]
	sh5	

文化(n)	sh4	I0 [n→・文化,1]
が(p)	sh7	I1 [p→・が,2]
きた(n;v)	re3	I0 [PP→・n p,2]
		I1 [PP→n・p,2]
きた(n)	sh4	I2 [n→・きた,3]
きた(v)	sh3	I2 [v→・きた,3]
から(p)	sh7	
	re2	I2 [S→・v,3]
から(p)	sh7	
	sh5,re1	I0 [S→・PP S,3]
		I2 [S→PP・S,3]
から(p)	sh7	I3 [p→・から,4]
	sh5	
	sh5	マージ
伝わった(v)	re3	I2 [PP→・n p,4]
		I3 [PP→n・p,4]
	re4	I2 [PP→・S P,4]
		I3 [PP→S・P,4]
		I0 [PP→・S P,4]
		I3 [PP→S・P,4]*
伝わった(v)	sh3	I5 [v→・伝わった,5]

```
Y10) [{0},0]-[{3},2]-┐[{11},2]-[{12},3]-        から(p) re2  |  I4  [S→・v,5]
     [{0},0]---------┘
Y11) [{0},0]-[{3},2]-┐[{11},2]-[{13},6]-        から(p) sh5,rel  |  I2  [S→・PP S,5]
     [{0},0]---------┘                                          |  I4  [S→PP・S,5]
                                                                |  I0  [S→・PP S,5]
Y12) [{0},0]-[{3},2]-┐[{11},2]-[{13},6]-          から(p) sh5
     [{0},0]---------┘
     [{0},0]-[{3},2]----------[{14},6]-              sh5,rel  |  I0  [S→・PP S,5]*
                                                                |  I2  [S→PP・S,5]
     [{0},0]--------------------[{15},1]-                sh5
Y13) [{0},0]-[{3},2]-┐[{3,11},2]-[{17},6]-        から(p)  sh5  |  マージ
     [{0},0]---------┘
     [{0},0]----------------------[{16},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．Ｔ１１）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trss同士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る．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．６章では実験によりこのことを説明する．

６．実験結果

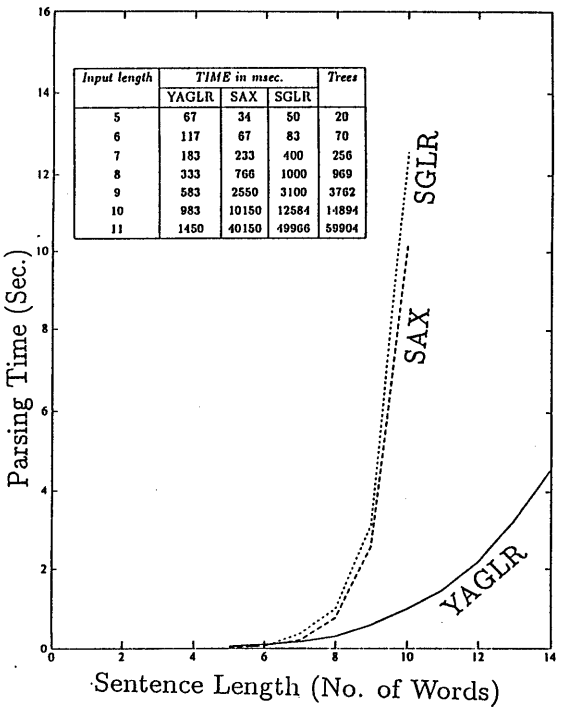
本章では，木構造化スタックを用いた富田法（これはＳＧＬＲとよばれている［沼崎　９０］）とＹＡＧＬＲとの比較を，具体的な文法を用いて実行速度の比較を行う．実験にはSun3/260ﾜｰｸｽﾃｰｼｮﾝ，Quintus-Prologを用いる．

・解析対象文：　aaa...a

・使用文法：　Ｓ→a
　　Ｓ→Ｓ Ｓ
　　Ｓ→Ｓ Ｓ Ｓ Ｓ

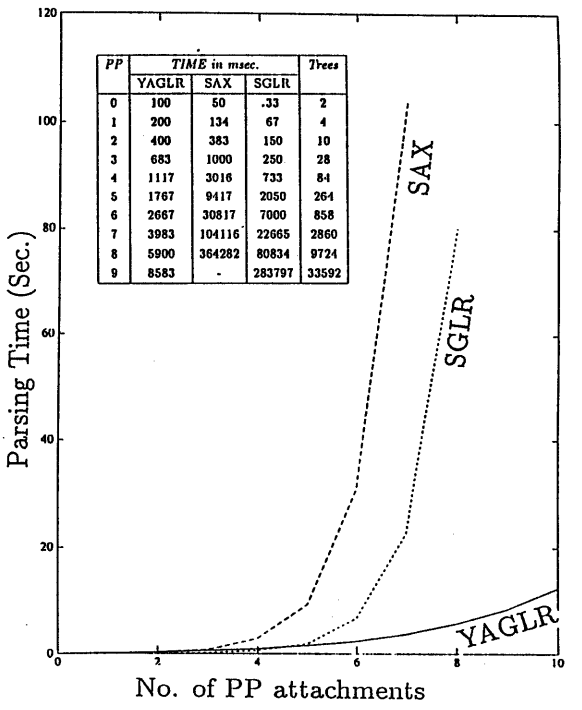
・解析対象文：　I saw a lady with telescope in the park in the park.....

・使用文法：　１２３規則の英語文法



Input length	TIME in msec.			Trees
	YAGLR	SAX	SGLR	
5	67	34	50	20
6	117	67	83	70
7	183	233	400	256
8	333	766	1000	969
9	583	2550	3100	3762
10	983	10150	12584	14894
11	1450	40150	49966	59904

図６．１　実験結果１



PP	TIME in msec.			Trees
	YAGLR	SAX	SGLR	
0	100	50	.33	2
1	200	134	67	4
2	400	383	150	10
3	683	1000	250	28
4	1117	3016	733	84
5	1767	9417	2050	264
6	2667	30817	7000	858
7	3983	104116	22665	2860
8	5900	364282	80834	9724
9	8583	-	283797	33592

図６．２　実験結果２

図６．１には，Johnsonらの与えた文法に対する実験結果を示す．この文法は，富田法ではアーリー法以上の解析時間のオーダを要するものである．これに対して，付録Ａに示すように，ＹＡＧＬＲ法はアーリー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）であることが証明されている．このことが図６．１の結果からも読み取れる．文の長さが長くなるにつれて，ＹＡＧＬＲ法の高速性が際だって来る．たとえばｎが１０ではＹＡＧＬＲ法は約１３倍，ｎが１１では約34倍高速である．図２はＰＰ付加の数を増やすことにより，解析結果の曖昧性を増大させ，解析時間がどの様に変化するかを示したものである．この場合も図６．１とほぼ同様な結果が得られている．図６．１と図６．２には，参考のためにＳＡＸ［Matsumoto　８９］の解析時間も示してあるが，ＳＧＬＲとＳＡＸとも，解析木を作らない場合の解析時間である．一方，ＹＡＧＬＲ法の場合には，逆ドット項を作成する時間が含まれていることに注意．

７．　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ことを説明した．本論文の結論を以下に述べる．

（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．付録Ａにその証明を与えてある．

（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．

・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．

・逆ドット項を用いることにより，富田法以上にtrss同士のマージを行うことができるため，操作し易い木構造化スタックtrssを用いることができる．これは富田法で用いるグラフ構造化スタックより構造が単純であるとともに，グラフ構造化スタックと同様に，使用記憶空間を抑えることができる．

（Ｃ）実験により，ＹＡＧＬＲ法が高速な統語解析アルゴリズムであることを実証した．ＹＡＧＬＲ法の高速性は，解析結果に含まれる曖昧性の数が増えるにつれて際だって来る．

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，trss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可能性がある．今後，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．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対称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. N.: Analysis of Tomita's Algorithm for General Context-Free Parsing, International Parsing Workshop '89, Carnegie-Mellon University, 193-202(1989).

［Matsumoto　８９］ Matsumoto, Y.: Natural Language Parsing Systems based on Logic Programming, Ph.D Theses, Kyoto University（１９８９）.

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

［田中　８９］　田中穂積：自然言語解析の基礎，産業図書（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).

付録A　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Lmに対して，解析対象文の長さをn+2とした時，富田法の解析時間のオーダはn^mである［Johnson 89］［Kipps 89］．　m>3なら，解析時間のオーダがn^3を超えてしまう．

文法Lm：　（1）S→a　（2）S→SS　（3）S→S^{m+2}

ここでS^{m+2}は記号Sがm+2個連結したものの省略形である．

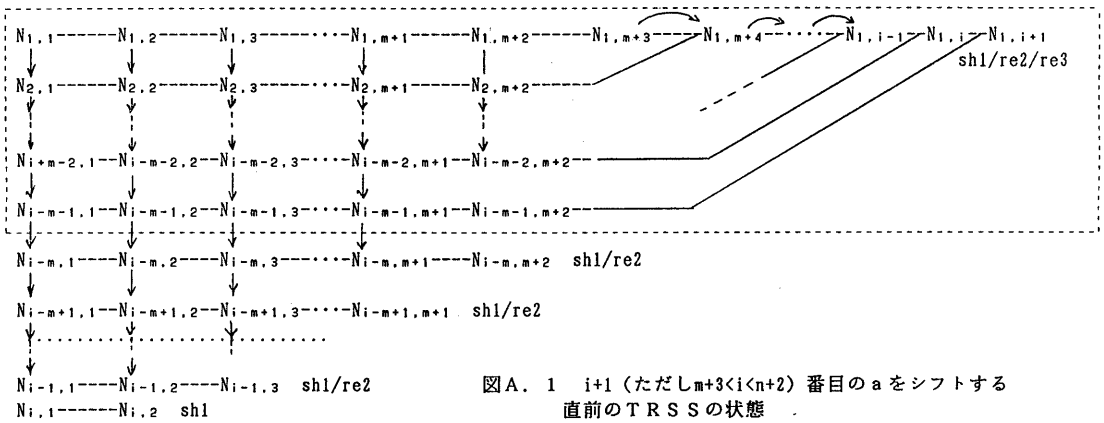
解析対象文を$a^{n+2}$$とする（ただしn>m）．上記した規則に対して右のLR表が得られる．今，i（m+3<i<n+2）番目のaをシフトしrelを施した直後のTRSSの状態を図A．1に示す．ここで$N_{p,q}$は節点を表し，各節点の持つ識別子付き位置番号の集合間には，矢印で示す包含関係がある．節点XとYの間にX→Yがあれば，Xの識別子付き位置番号の集合がYの部分集合であることを示す．

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，

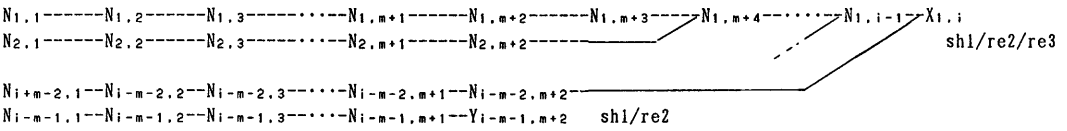
$$T_{p,q} = \begin{cases} q & if \quad 1 < q \leq m+3, \\ m+3 & if \quad m+3 < q, \\ 0 & if \quad q = 1. \end{cases}$$

状態	アクション部		GOTO部
	a	$	S
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A．1のtrssに対して，シフト操作の前に，全てのレデュース操作を施す必要があるが，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，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A．1の最上部のtrssに対して，規則（2）によるレデュース操作を一度施すと，下図の二つの新しいtrssが得られる．



図A．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



上図のtrssは，図A．1の統合化可能な他のtrssと最終的にマージされる．たとえば，シフト操作の直前では，sh1を待つtrssのスタックトップの状態がm+3で等しいもの同士はマージされる．図A．1の最長スタックと，上記した新たな二つのtrssの内，上のtrssは，スタックトップの節点の状態がm+3であるからマージされる．ただし，図A．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$X_{1,i}$>，<$N_{1,i}$と$N_{1,i-1}$>，<$N_{i-m-1,m+2}$と$N_{i-m-2,m+2}$>）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$N_{i-m-2,m+1}$>）のマージは，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，マージ操作はここで終わり，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番目の節点までのマージの手間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i回である．したがって，レデュース操作とマージ操作にかかる時間の総和のオーダはi^2である．以上より，a^{n+2}の入力文を解析するの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A．1の各節点中の識別子付き位置番号の数は，節点$N_{1,j}$に対して，2≦j≦m+3の場合に1，m+4≦j≦i-1の場合に2，

$i \leqq j \leqq i+1$の場合に1，また節点$N_{k,j}$$(2 \leqq k \leqq i-m-1,\ 2 \leqq j \leqq 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，$1*(m+4)+2*(i-m-4)+(m+2)(2+3+\ldots+(i-m-1))$であり，そのオーダは$i^2$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総数のオーダーはn^3であることが証明された．

付録Ｂ　ＹＡＧＬＲ法の位置づけ