論　文

LR(k) 言語アナライザのもう1つの実際的構成法——SLR(k) の拡張——*

牧之内　顕文**

Abstract

A practical method for constructing a syntax analyzer for a subclass of LR(k) languages is described in this paper.

The method is an extension of what is proposed by DeRemer as a construction method for SLR(k) languages. Instead of his CFMS, the concept of grammar graph is proposed and as a result our method may cover SLR(k) and a subclass of general LR(k) langages.

An analyzer that can be obtained by means of our methoed is a Deterministic Pushdown Automaton like that defined by DeRemer.

1. まえがき

プログラミング言語のトランスレータにおける構文解析部（アナライザ）の形式化については種々の提案がなされてきた．その中で Knuth[1] の LR(k) 文法のアナライザはその強力さで注目されたが，構文解析表が大きくなり過ぎるために実現はかなり遅れた．最近になってメーカがコンパイラ作成の効率化を目標として，LR(k) アナライザの自動作成を実現した[6),7)]．これらは DeRemer が彼の論文[3] で提案した手法に基づいている．彼の手法は3つの部分よりなっていると考えられる．

（1） 与えられた LR(k) 文法から CFSM (characteristic Finite State Machine) の作成．CFSM は1つの遷移図で，文法の終端記号，非終端記号や # 遷移記号で状態間の遷移が行われる．

（2） 不適合状態での前方参照記号列の計算．

（3） CFSM を DPDA(Deterministic Push Down Automaton) へ変換．

彼は遷移図と前方参照記号列の計算を分離することによって，LR(k) 文法を SLR(k)，LALR(k) 及び一般 LR(k) 文法の3つのクラスに，前方参照記号列の計算の複雑化に従って分類した．

この論文では DeRemer の手法を発展させた手法を記述する．これは4段階から構成される．

（1） 文法グラフと呼ぶ遷移図の作成．

（2） 文法グラフを DPDA グラフに変換し，不適合ノードについてその前方参照記号列を計算する．

（3） 冗長なノードの排除．

（4） DPDA グラフを DPDA に変換．

この手法は DeRemer の SLR(k) 文法のアナライザと同程度に簡単に，SLR(k) ばかりでなく，一般 LR(k) のあるクラスのアナライザを一貫性を持って——即ち，LaLonde[5] が示したような LALR(k) に必要とされる前方参照記号列計算の補正や，一般 LR(k) のために DeRemer によって提案されたノード分割方を使わないで——構成するものである．

2. 用語の簡単な説明

1つの LR(k) 文法は $G=(V_T, V_N, S, P)$ で表現される．ここで V_T は終端記号の集合，V_N は非終端記号の集合，S は V_N の要素で出発記号，P は構文規則の集合である．別に指示がないときは，V_N の要素は英文字の大文字で，V_T の要素は小文字の英文字で表現する．$V=V_N \cup V_T$ とすれば，V^* は V の要素の任意の長さの列で，長さ零の要素を特別に ϵ で表わ

* Another Practical Method of Analyzer Construction for LR(k) Languages——An Extention of SLR(k)——by Akifumi MAKINOUCHI (Computer Science Lab.,Fujitsu Laboratories Ltd.).

** (株)富士通研究所電子研究部

す．以下簡単のために V の要素をギリシャ文字の小文字で，時にはサフィックスをつけて表わす．そうすると構文規則は $A\to\alpha_1,\alpha_2,\cdots,\alpha_n$ あるいは $A\to\epsilon$ と表現される．この場合 A は構文規則の左辺，$\alpha_1,\alpha_2,\cdots,\alpha_n$ または ϵ は右辺である．一般性を失うことなく次の条件を付与する．

（1） 構文規則は番号づけられている．

（2） 1番目の構文規則は $S\to\vdash S'\dashv$ の形をしている．但し S' は元の出発記号であり，S と終端記号$\vdash$と$\dashv$とは他のどんな構文規則にも現われない新しい記号とする．

例として，次の文法を上げる．

$G_1=(\{a,b,c,d,e,\vdash,\dashv\},\{S,E,A,B\},S,P_1)$

但し P_1 は次の構文規則の集合である．

$S\to\vdash E\dashv$	#1	$A\to eA$	#6
$E\to aAd$	#2	$A\to e$	#7
$E\to aBc$	#3	$B\to eB$	#8
$E\to bAc$	#4	$B\to e$	#9
$E\to bBd$	#5		

#1, #2, …, #9 は構文規則につけられた番号である．以下の説明ではこの例を引き合いに出す．この例は DeRemer が示したように SLR(k) でも，LALR(k) でもない LR(1) である．

本論文での処理の対象はグラフの一種で次の性質を持つ．

（1） 有限個のノードとそれらの間に引かれた方向を持った線分（以後，遷移と呼ぶ）からなる遷移図である．

（2） 各遷移には遷移記号が付与されている（a を遷移記号とすれば，それが付与されている遷移を a 遷移と呼ぶ）．

（3） 各ノードは状態を持つ．その状態は要素状態の集合である．要素状態は **3.** で定義される．

（4） 1つのノードから出る遷移で同一のノードに向かう遷移が複数本存在するときは，それらの遷移に付与されている遷移記号は同一である（実際には処理の途中でこういう事態が生じたときは，その内の1本を除いて他は消去される）．

説明の都合上各ノードにはユニークな番号をつける．また以降では上記の性質を持った遷移図を単にグラフと呼ぶ．**Fig. 2.1** に例を示す．

Fig. 2.1 の a,b は遷移記号であり，1, 2, 3, 4 はノード番号である．S_1, S_2, S_3, S_4 は各ノードの状態である．

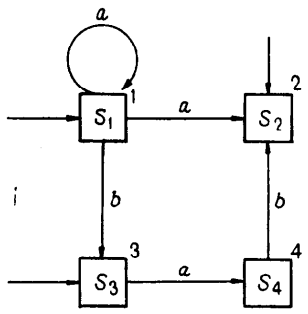

Fig. 2.1 An example of transition graph

ノード n_1 から $n_2, n_3, \cdots$ を経て n_i に至る遷移のつながりを n_1 から n_i へのパスと呼ぶ．パスは区別できる時はその上の遷移記号列で表現する．ノード m が n から到達可能であるというとき，n と m は同一であるかまたは n から m へのパスが存在する．従って Fig. 2.1 ではノード2は1からパス a, bab, aa, etc.… で到達可能である．上記のパスの長さは各各 1, 3, 2 である．

あるグラフとその中の1つのノードが与えられた時，そのノードの induced グラフとは，そのノードから到達可能なすべてのノードとそれらの間の遷移とから成立するサブグラフのことである．Fig. 2.1 の例では，ノード3の induced グラフはノード 3, 4, 2 とそれらの間の a 遷移，b 遷移からなるサブグラフである．

以下の記述でよく使うグラフの変形の手続き“ノードの重ね合わせと同一遷移の消去”について簡単に述べる．まず，状態 S_n と状態 S_m を有するノード n と m があって，n を m に重ね合わせるとは，n に出入りする遷移を，m に出入りするように移し，n を消去することとする．この結果ノード m は新しい状態 $S_n\cup S_m$ を持つことにする．この定義の元で，上の変形のアルゴリズムを記述する．与えられたグラフを G とする．

1 Gのノードの内で同一遷移記号を持つ遷移が複数個出ているようなノードの集合を N とする．

2 N が空であるなら終了．そうでないなら N の内任意の1つを n とする．

3 n から出る遷移で同一の遷移記号を持つ遷移（複数本）を探す．もしないなら N から n を除去してステップ2に．あるなら，それらが向うノードを $m_1, m_2, \cdots, m_k$ $(k>1)$ としてステップ4

に.

4 $m_i=n$ $(1\leqq i\leqq k)$ ならば，n を，そうでないなら任意のノード m_i を選んで，それに m_j $(1\leqq j\leqq k,\ i\neq j)$ を重ね合わす．n から同一遷移記号で同一のノードに向かう遷移の内，1本だけを残し他はすべて消去する．ステップ3に戻る．——終り——

3. 文法グラフ

この章では文法グラフと呼ぶ遷移図とその作成アルゴリズムの理解に必要な性質について述べる．この文法グラフは基本的には DeRemer の CFSM と類似しているが，次の2点において異なっている．

(1) 文法グラフのノード数は CFSM より等しいか多い．これは DeRemer の手法によれば必要とされるノード分割を必要としないという事を意味する．しかしノードが多いという不都合は **5.** で述べる冗長ノードの除去を行えば改善される．

(2) 文法グラフには CFSM にはない還元 (reduction) 遷移がある．この遷移は前方参照記号列を，与えられた文法からではなく，それより作られた文法グラフから計算するのに役立つ．

グラフ作成のアルゴリズムを簡単にし，明確にするために必要ないくつかの概念を導入する．

構文規則グラフ

第 i 番目の構文規則を

(1) $A->\alpha_1\alpha_2\cdots\alpha_n$ $\#i$ または

(2) $A->\epsilon$ $\#i$ とすると

Fig. 3.1 に示されるグラフを第 i 構文規則グラフと名付ける．

入ってくる遷移のないノードを構文規則グラフの出発ノードという．点線は還元遷移であって ϵ 記号での遷移とみなす．$(i,1),(i,2),\cdots,(i,n)$ と $(0,i)$ は i 構文規則グラフの要素状態で，それぞれのノードの状態の要素である．

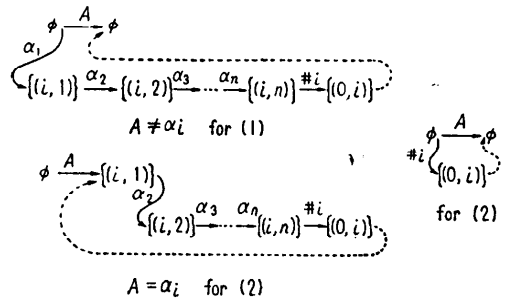

Fig. 3.1 The i-th production rule graph for (1) $A\to\alpha_1,\alpha_2,\cdots,\alpha_n$ $\#i$ or (2) $A\to\epsilon$ $\#i$

ノード ϕ は空集合を状態とするノードである．状態 $\{(0,i)\}$ を有するノードは特別なノードで i 構文規則グラフの終点ノードと呼ぶ．各グラフの終点ノードの状態は他と混同することがないので以後単に二重線のノードで表現する．

一般に第1構文規則グラフは他と次の2点で異なっている．

(1) 出発ノードは常に状態 $\{(1,0)\}$ が与えられている．

(2) 還元遷移がない．

非終端記号ノードの展開，configuration グラフ，上部ノード集合

与えられた文法 G の文法グラフは G の構文規則グラフを用いて，その第1構文規則グラフを最初のグラフとして，段階を踏んで作られる．今 GR_i をある段階でのグラフとしよう．そのグラフのノード n は，もし n から少なくとも1本の非終端記号遷移が出ているとき，非終端記号ノードと呼ばれる．GR_i 中の非終端記号ノード n は，下のアルゴリズムに従って1つのグラフが作られ GR_i に連結されるとき，展開されたという．

1 C を非終端記号の集合を表わす変数とする．C にノード n から出ているすべての非終端記号遷移の遷移記号をセットする．

2 C に属してかつ印のつけられていない非終端記号を1つとり出しそれを Q とする．

Q を左辺にもつすべての構文規則の構文規則グラフの出発ノードをノード n に重ね合わせる．

3 ステップ2の結果，C に含まれない非終端記号による遷移が n につけ加わるならばその非終端記号を C に加える．Q に印をつける．

4 C のすべての要素に印がつけられるまでステップ2と3をくり返す．

5 ステップ4までに作られたグラフから還元遷移を除いたサブグラフを考える．そのサブグラフ上のノード n の induced グラフを SGR_n とする．SGR_n に“ノードの重ね合わせと同一遷移の消去”手続きを適用する．この手続きの途中で，ステップ2～4で作られた新しいノードと，既に GR_i に含まれていたノードとを重ね合わせる事態になった時は新しいノードを既存のノードに重

ねることにする．また還元遷移は保存する*．

——終り——

次にこのようにして作られたグラフから還元遷移を除いた部分グラフ上でのノード n の induced グラフを考え，これを n の configuration グラフ CGR_n と表わす．

ノードを重ね合わせるとき，重ね合わすべきノードの状態とそれに出入りする遷移はそのまま重ね合わされるノードに保存されることと，文法グラフを構成する素材の構文規則グラフの形**とその終点ノードに入る遷移の遷移記号は皆異なっていること***とを考えれば，次の事は明らかである．

(1) CGR_n のノード m が状態 $\{(i_1, j_1), (i_2, j_2), \cdots, (i_q, j_q)\}$ を持てば，m を出発ノードとして，第 i_p 構文規則グラフの終点ノード $(p=1, 2, \cdots, q)$ で終る q 本のパスが存在して，各パスをたどって得られる遷移記号列は，対応する構文規則グラフの対応する要素状態を状態とするノードから終点ノードまでのパスの遷移記号列と同じである．これをノード m の状態から決まるパスという．

(2) ノード n には，出力遷移を持った空集合のノードが重ね合わせられるから，n の状態がそのままでも新しい出力遷移が付与される可能性****がある．

(3) n 以外のノードでは，もし，新しい出力遷移が付与されたならば，そのノードがもともと持っていた状態に新しい要素状態が付加されている．

n を展開する前のグラフ GR_i から還元遷移をとり除いたサブグラフ上のノード n の induced グラフが n の状態から決まるパスのみから成立する木グラフであると仮定すれば，展開後の CGR_n もやはり木グラフであり，かつ CGR_n 上の，n を除くノードからなるサブグラフも，やはりそのノード状態から決まるパスのみによって構成される木グラフであることも容易に分かる．

後述するアルゴリズムでは展開すべきノードをうまく選んでこの仮定が常に成立するようにしてある．

CGR_n は木グラフであるから，その中の各ノードは出発ノードからの距離に従って一意的に順序づけられる．即ち，今 CGR_n の2つのノードを p と m としたとき，もし n から p へのパスの長さが，n から m へのそれより短かい場合には p は m より n に近いという．この順序づけは次に展開すべきノードの選択に重要な役割をはたす．

最後に上部ノード集合を定義する．今1つの configuration グラフ CGR_n の1つのノード m をとると m の上部ノード集合 UP_m は次の式で与えられる．

$$UP_m = UP_n \cup \{n\}$$

但し UP_n は n の上部ノード集合である．

ノード n の展開の前に既に自身の上部ノード集合を持っており，かつ展開後 n の configuration グラフの1つのノードになるようなノードが存在する場合がある．その場合そのノードの上部ノード集合は再計算される．

これだけの準備の後で，文法 G が与えられたときその文法グラフを構成するアルゴリズムを記す．

ここでは2つの変数 OR と ND を用いる．OR は configuration グラフの出発ノードの集合を，ND は次に展開すべき非終端記号ノードの集合を表わす．最初は共に空である．

1 G の最初の構文規則グラフを作る．OR にその出発ノードをセットする．このノードの上部ノード集合は空集合である．このグラフはそのノードを出発ノードとする configuration グラフと考えられる．

2 もし OR が空なら終り．そうでないなら OR の1要素をとり出しそれを n とする．n を OR より除く．

3 CGR_n の n 以外の非終端記号ノードを ND にセットする．

4 もし ND が空ならステップ2に行く．

5 ND の中で n に一番近いノードを1つ取り上げる．それを m とする．m の上部ノード集合 UP_m を計算する．もし UP_m が m と同じ状態をもつノードを含んでいるならばそれを m' としてステップ6に行く．そうでないならステップ7に．

6 m を m' に重ね合わせる．全体のグラフから還元遷移を除いたサブグラフ上で m' の induced グラフに“ノードの重ね合わせと同一遷移の消去”手続きを適用する．m を m' とを重ね合わせない前の（還元遷移を除いた）グラフの各々のノ

---

* 但し，ノードの重ね合わせの結果，1つのノードから2本以上の還元遷移が同一のノードに向かう時はその内の1本だけ残し，他は消去する．

** 構文規則グラフは還元遷移を除けば，終点ノードを葉ノードとする木グラフである．

*** 異なる構文規則グラフの終点ノードが重ね合わされることはないことを意味する．

**** (1)で述べた状態で決まるパス以外のパスを持つノードはそのようなノードであり，またアルゴリズムから明らかに分るようにそのようなノードに限る．

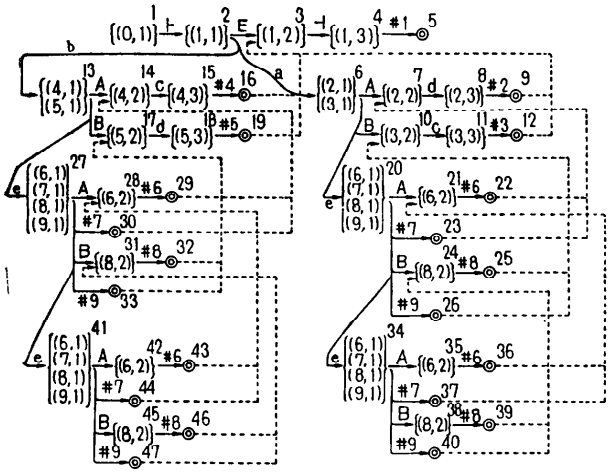

Fig. 3.2 Grammar graph for G_1 is going to be constructed. Nodes 2, 6, 13, 20 and 27 were successively developed. The nodes to be developed next are nodes 34, 41. Since nodes 20 and 27 have the same states as that of nodes 34 and 41 and nodes 20 and 27 are in UP_{34} and UP_{41} respectively, nodes 34 and 41 are superposed to 20 and 27 instead of being developed. The result is shown in Fig. 3.3.

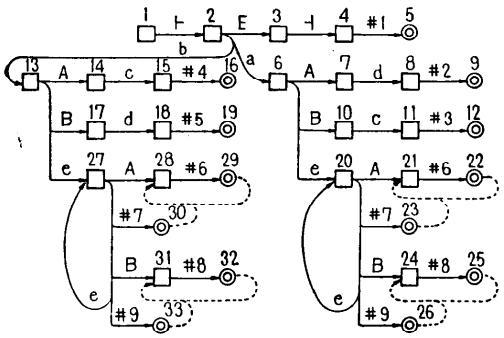

Fig. 3.3 Grammar graph for G_1. Note that the reduction arrows shown here are only those changed from Fig. 3.2.

ードの induced グラフを IG_m と $IG_{m'}$ とすると次の事に注意する．いま IG_m のノード p と $IG_{m'}$ のノード p' を上記手続き中に重ね合わせねばならない時には p を p' に重ねることにする．また還元遷移は保存する．コメント——m と m' は同じ状態を持つから，IG_m と $IG_{m'}$ は，各々 m と m' の状態から決まるパスよりなる木グラフを含む．しかも，IG_m 上のどんなノードもまだ展開されていないから，IG_m は実は，それらパスのみより構成される木グラフである——コメント終り．

ND からノード m と，そして存在するなら，上記手続き中で消去された（重ね合わされた）ノードを取り除く．ステップ4に戻る．

7 m を展開する．ND より m と，もし存在するなら，新しく作られた m の configuration グラフ CGR_m のノードに組み入れられたノードを取り除く．m を OR に加えてステップ4に戻る——終り——

文法 G_1 の文法グラフを **Fig. 3.3** に，その直前のグラフを **Fig. 3.2** に例示してある．

4. 前方参照記号列の計算

DeRemer は，CFSM の不適合状態に対して必要な前方参照記号列を単純に文法のみから計算するという手法を基にして SLR(k) 文法のアナライザの構成法を示した．

筆者は CFSM の代りに文法グラフを創案しかつその上で必要な前方参照記号列を計算するという方式をとる．これは文法グラフに還元遷移を導入したことにより容易になった．

文法 G の文法グラフから非終端記号遷移をすべて消し去って得られるサブグラフを文法 G の DPDA グラフと呼ぶことにする．DPDA グラフのノードで，そこから少なくとも2本以上の遷移が出ており，かつその遷移の内少なくとも1本が # 遷移記号による遷移である場合，そのノードは不適合であるという．前方参照記号列集合は DPDA グラフ上の遷移記号の列の集合であって下のように定義される．今 m と n_i を2つのノードとし，m から出る遷移を $x_1, x_2, \cdots, x_l$，n_i から出る遷移を $y_1, y_2, \cdots, y_h$ とし，x_i が m と n_i をつないでいるものとする．ここで x_j, y_g ($j=1, \cdots, l$, $g=1, \cdots, h$) は文法の終端記号か # 遷移記号か ϵ(空記号)（即ち，還元遷移上の遷移記号）かである．ノード m の長さ k の前方参照記号列集合を $LS_m{}^k$ と表現すれば，

$$LS_m{}^k = \bigcup_{i=1}^{l} \{ls_{x_i}{}^k | ls_{x_i}{}^k \text{ は遷移記号列の集合であり，その要素の記号列は長さ } k \text{ で次のようにつくられる．}$$

1 x_i と $LS_{n_i}{}^k$ の各記号列とを連結する．

2 もしその記号列中に # 遷移記号が含まれるなら，それを除去する．

3 もし必要なら，各列の右端に適当な数の⊣記号を連結するかまたは右端の適当な数の遷移記

号を除去して長さを k にする.}

即ち, LS_m^k は m からのすべてのパスをたどることによって得られる終端記号からなる長さ k の遷移記号列の集合であって, 各パスの最初の遷移に従って分類されている. $LR_m^k = \bigcup_{i=1}^{l} ls_{x_i}^k$ はもし任意の i, j ($i \neq j$) に対して $ls_{x_i}^k \cap ls_{x_j}^k = \phi$ ならば, 排反であるという.

さてある LR(k) 文法 G が与えられたとき, その文法グラフを作り, そこから DPDA グラフを導き, その上の不適合ノードに対して, 前方参照記号列集合を計算する. もしその集合のどれもが排反ならば, 与えられた文法 G のアナライザはこの手法で作ることができることを意味する. そうでないならば, もっとより一般的な構成法をとらねばならない.

文法 G_1 の DPDA グラフは Fig. 3.3 のグラフから非終端記号遷移を取り除いたものである. ノード 27, 20 が不適合ノードである. $LS_{27}^1 = \{\{c\}_{\#7}, \{d\}_{\#9}, \{e\}_e\}$, $LS_{20}^1 = \{\{d\}_{\#7}, \{c\}_{\#9}, \{e\}_e\}$ だから, 共に排反である. 従って G_1 のアナライザは今まで述べてきた手法で構成できることが分った.

5. 冗長なノードの消去

ある与えられた文法の DPDA グラフは通常 DeRemer の CFSM より多くのノードを持つ. これは, グラフ中の冗長なノードを消去することによって改善される. アルゴリズムの説明の簡単化のために, DPDA グラフから還元遷移を除いたサブグラフ上でのノード n の induced グラフ (以後 IGR_n と表わす) の適合性という概念を導入し, かつノードの重ね合わせの処理を拡張して前方参照記号列集合も扱えるようにする. ノード n の induced グラフ IGR_n は, もしそれが少なくとも1つの不適合ノードを含んでいるとき, 不適合であるという. そうでないとき適合である. DPDA グラフは文法グラフから非終端記号遷移を除いただけのものであるから, 文法グラフの作り方とノードの状態が意味することから次の事は明らかである. 即ち, もし2つのノード m と n とが同じ状態を持つときは, IGR_m と IGR_n は m と n とを対応させかつ同じ遷移記号の遷移を対応させるようなノード間と遷移間の1つの対応が2つのグラフ間に存在するという意味で同型である. またこの対応によって対応づけられるノードは同じ状態を持つ. 例えば, 文法 G_1 の DPDA グラフ上では (Fig. 3.3 を参照), IGR_{20} と IGR_{27} は同型でありまた IGR_{21} と IGR_{28} 及び IGR_{24} と IGR_{31} もそうである. この性質より, もし m と n とが同じ状態を持つなら, IGR_m は IGR_n に完全に重ね合わすことができる. またその重ね合わせによってノードの状態が変わることもない.

ノードを重ね合わすとき, 次の条件を荷す. p と q を各々 IGR_m と IGR_n の対応する不適合ノードとする. IGR_m を IGR_n に重ね合わせるとき, p は q に重ね合わせられる. p と q は各々 $\{ls_{\alpha_i}^k\}$, $\{ls_{\alpha_i'}^k\}$ なる前方参照記号列集合を持つとすると, 重ね合わせの結果 q は $\{ls_{\alpha_i}^k \cup ls_{\alpha_i'}^k\}$ なる複合前方参照記号列の集合を持つことにする.

冗長ノードを消去させるアルゴリズムは次のようになる.

1 CPL をノードの対の集合を表現する変数とする. CPL を空にする.

2 DPDA グラフをサーチして, 同じ状態を持つノードの対でまだ CPL に入れられていないものを探す. もしなかったら終了. もしあったらそれらを n と m として次に.

3 n と m の induced グラフを計算する. IGR_n と IGR_m とが適合ならば IGR_n を IGR_m に重ね合わせる. ステップ2に戻る. 不適合ならば次に.

4 IGR_n と IGR_m の対応する不適合ノードの任意の対の複合前方参照記号列集合が排反ならば, IGR_n を IGR_m に重ね合わせる. そうでないなら対 (n, m) を CPL に入れる. ステップ2に戻る——終り——

この手続きを文法 G_1 の DPDA グラフに適用して得られたグラフが **Fig. 5.1** に示されて

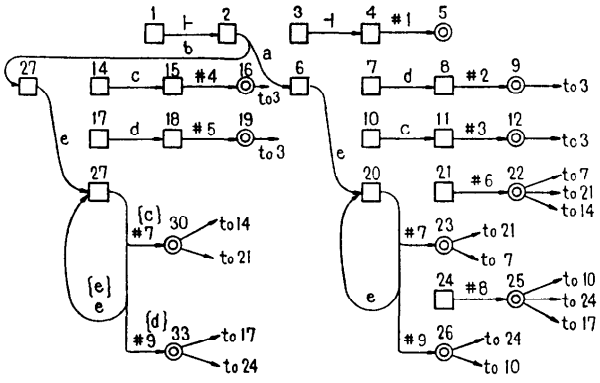

Fig. 5.1 DPDA graph for G_1 after redundant nodes elimination. Nodes 28, 29, 31, 32 were eliminated.

いる．ノード 28, 29, 31, 32 が冗長なものとして除去された．

6. むすび

DPDA グラフから DeRemer の DPDA への変換は，彼の論文[3]に詳述してあるのでここでは述べない．しかしながら，DPDA グラフの還元遷移の指すノードが DPDA で対応する還元が起ったときに次に遷移する状態であることを指摘しておく．

DeRemer は LR(k) 文法のアナライザを作成するのに，それを LR(0) 文法とみて遷移図を作ることから始めた[3),8)]．この遷移図の作り方は本質的には Knuth[1)] のと異ならない．筆者はこの遷移図の作り方に工夫をこらすことにより，SLR(k) を拡張した文法のアナライザの構成法をこの論文で述べた．この「拡張」の程度は重ね合わせの対象となる上部ノード集合の計算法に依存する．いま，上部ノード集合を付与される対象ノードを m とすれば

(B) UP_m＝{すでに展開されたすべての非終端記号ノード}

(C) $UP_m=UP_n\cup\{n\}$ 但しノード n は，ノード m が最初に作られたのは，n が展開されたときであるようなノード．

(A)を本論文中で示した計算法とし，G_2, G_3 を下に記した L(2)R(1)，一般 LR(1) 文法とすると次のことがいえる．G_2 は (A), (B), (C) のどれに従っても，本論文で示した構成法によってそのアナライザを作成できる．その場合，途中で作られる冗長ノードの数は (B), (A), (C) の順で多くなる．G_1 は (A), (C) を適用したときのみ作成可能．G_3 は (A), (B) を使用したとき作成不可能．(C) のみ適用可能．

$G_2=(\{a, b, c, d, e, \vdash, \dashv\}, \{S, A, B, E\}, S, P_2)$

P_2: $S\to\vdash E\dashv$ ♯1　　$E\to aBc$ ♯3

$E\to aAd$ ♯2　　$E\to bAc$ ♯4

$E\to bBd$ ♯5　　$B\to e$ ♯7

$A\to e$ ♯6

$G_3=(\{a, c, d, e, f, \vdash, \dashv\}, \{S, E, A, B\}, S, P_3)$

P_3: $S\to\vdash E\dashv$ ♯1　　$A\to eAfBd$ ♯5

$E\to aAd$ ♯2　　$A\to e$

$E\to aBc$ ♯3　　$B\to eB$

$A\to eAfAc$ ♯4　　$B\to e$

おわりに，有益な助言を与えて下さった当社林達也氏，及び本論文に目を通していただき，文献8)の存在をお知らせ下さった東工大教授 井上謙蔵氏に深謝します．

参考文献

1) Knuth, D.E.: On the Translation of Languages from Left to Right, Information & Control, Vol. 8 (1965).
2) Korenjak, A.J.: A Practical Method for Constructing LR(k) Processors, C. ACM, Vol. 12, No. 11 (1969).
3) DeRemer, F.L.: Practical Translation for LR(k) Languages, Project MAC MIT, (1969)
4) 林達也: On the Constraction of LR(k) Analyzer, Proc. of ACM 1971 National Conference.
5) Lalonde, W.R.: An Efficient LALR Parser Generator, Technical Report CSRG-2, University of Toronto, (Feb., 1971).
6) Dubbs, E.W. et al.: A Microprogram System Translator—An Introduction—, Proc. of IEEE 1973 Computer Conference.
7) 小島富彦，他: LR(k)-parser 生成システムとその Fortran コンパイラへの応用，情報処理，Vol. 15, No. 2 (1974).
8) A.V. Aho, J.D. Ullman.: The Theory of Parsing, Translation and Compiling. I, II 巻の第5，7章．

(昭和49年11月15日受付)
(昭和50年12月26日再受付)