C 7　State Graph 作成のリスト処理について

（言語用 Expression の処理）

藤　野　喜　一　（早稲田大学）

0. INTRODUCTION　Regular expression（RE)を認識する順序回路の構成アルゴリズムとしてMcNaughton & Yamada〔1〕, その他の方法がある.

ここではREをstate graphへ変換する問題をリスト処理手法によつて行なう手順を解説する. REに5種類の基本タイプを導入しこれらの間に∪(or), •(and), *(iteration)の3つの演算の基本形を定義する. この際REをリストで表現し演算の実行にはコンパイラーの算術式の変換手順を応用する. 又REの基本タイプに対応する基本state graph(BSG)を定義して与えられたREのstate graphを自動的に作成する.

1. アルゴリズムの説明

1.1　Regular expressionの定義

この方法では, Regular expression（REとかく）を次の如く定義する.

（1）REの構成要素

- symbols
  - φ（empty elementを表わすatom）
  - φ以外のatom
- operators
  - binary operator ∪(or), •(and)
  - unary operator　*(iteration)
- parenthesis　'(', ')'

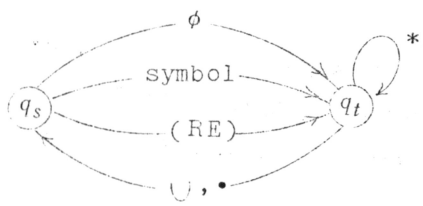
（註）atomとは, それ以上分解できない情報の単位である.

（2）REの定義

（i）Backus notationによる

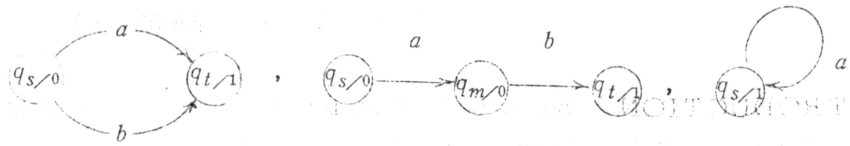
<RE>::=<φ>|<atom>|<(RE)>|<RE>*|
<RE><∪><RE>|<RE><•><RE>

（ii）REをstate graphで示せば次の図になる.



1.2　ＲＥの基本パターン

定義Ｔ１タイプのＲＥ　　ＲＥの集合をRとし，R(Ｔ1)をRの部分集合とする．a，bをR(Ｔ1)に属する任意のＲＥとするとき，a，bから合成される３つのＲＥ ① $a \cup b$ ② $a \cdot b$ ③ a^* のSG(state graph)が夫々

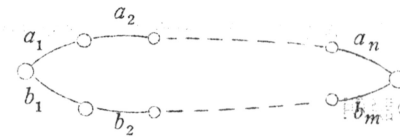


で矛盾なく表現できるならば，R(Ｔ1)はＴ１タイプのＲＥの集合という．

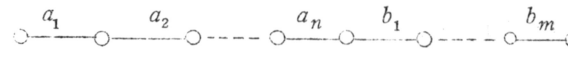
Ｔ１タイプのＲＥの例

（ｉ）有限個のａｔｏｍの積　　$a_1 \cdot a_2 \cdot \cdots \cdot a_n = \prod_{i=1}^{n} a_i \quad (n \geq 1)$

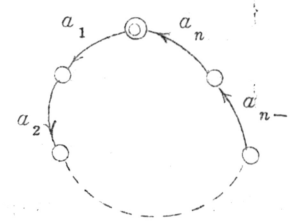
$$\prod_{i=1}^{n} a_i \cup \prod_{j=1}^{m} b_j \sim$$



$$\left(\prod_{i=1}^{n} a_i\right) \cdot \left(\prod_{j=1}^{m} b_j\right) \sim$$



$$\left(\prod_{i=1}^{n} a_i\right)^* \sim$$



上の対応が成立つので，$\prod_{i=1}^{n} a_i$ (a_i=atom)はR(Ｔ1)に属する．

定　義　　Ｔ２タイプのＲＥとはＴ１タイプのＲＥに＊演算を施したものである．則ち$a \in R$(Ｔ1)ならば$a^* \in R$(Ｔ2)である．

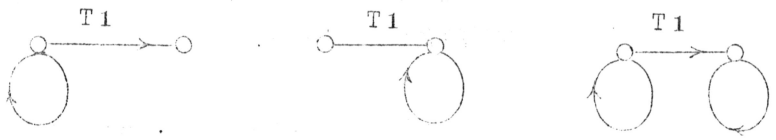
またＴ１タイプのＲＥは次のようにrecursivelyに定義することも出来る．

Ｔ1::=atom | Ｔ1∪Ｔ1 | Ｔ1・Ｔ1 | Ｔ1・Ｔ2・Ｔ1

但しＴ２は$T1^*$とかいてもよい．

定　義　　aがＴ２タイプ，bがＴ１タイプのＲＥであるとき，$a \cdot b$をＴ３タイプのＲＥといい，又$b \cdot a$をＴ４タイプのＲＥという．又a，cをＴ２タイプのＲＥ，bをＴ１タイプのＲＥとするとき$a \cdot b \cdot c$はＴ５タイプのＲＥとよぶ．夫々次の如く３つのSGに対応する．

T3タイプ　　T4タイプ　　T5タイプ

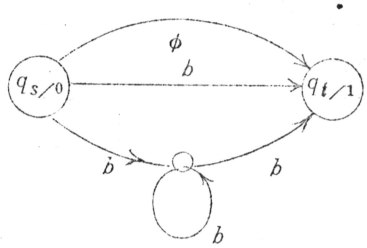


性質1　T2タイプのREはT1タイプのREに変更できる.

aがT2タイプのREならば，定義によつて，$a=b^*$とかける．但しbはT1タイプのREである．したがつて，aのSGは $\circlearrowleft\, b$ となる．一方b^*は見かけの異なる表現のRE $\phi\cup b\cup b\cdot b^*\cdot b$ と同値で，このREはT1タイプの3つのREの和になつているから，T1タイプのREである．そこで*3とかく新しいoperatorを設定して，

$$b^{*3}=\phi\cup b\cup b\cdot b^*\cdot b$$

と定義することによつて，T2タイプのRE b^* をT1タイプのRE b^{*3} に変更する変換τを定めれば，見かけ上異なるT1タイプのREに変えることができる．b^{*3}のSGは次の如くなる．$\tau(b^*)=b^{*3}$



性質2　全てのREはT1タイプに帰着できる.

(i)　T2タイプの場合……上の性質による.

(ii)　T3タイプの場合……T3=T2・T1であるからT2の部分即ちT3タイプのREのもつとも左側(又はtop)の部分をT1タイプに変更すればよい.

(iii)　T4タイプの場合……T4=T1・T2であるからT2の部分即ちT4タイプのREのもつとも右側(又はbottom)の部分をT1タイプに変更すればよい.

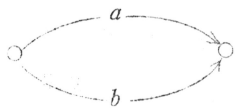
(iv)　T5タイプの場合……T5=T2・T1・T2であるからT5のREの両端のT2タイプのRE's をT1タイプに変更すればよい.

1.3　$\cup$, $\cdot$, $*$のoperationの直接演算可能性について

基本REパターンT1, T2, T3, T4及びT5の間に$\cup$, $\cdot$, $*$の演算を考えてみる.

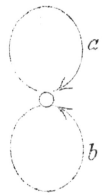
2つのRE's a, bが与えられるとき，$a\cup b$の演算の結果をa, b の表現に何らの変

更も必要としないで，直ちに( )のSG



に対応させることができるとき，a と b をoperator ( )に関して直接演算可能であるという．

同様に・及び＊に関する直接演算可能性を定義することができる．例えば a^*( ) b^* (a, b はatom)は直接演算可能ではない．もし a^* と b^* のSGのinitial stateを重ねterminal statesを重ねるとしたら次のSGができる．



このSGには例えば $a \cdot b$ のようなREが含まれてしまつて具合が悪い．この場合は a^*, b^* をT1タイプの a^{*3}, b^{*3} に変更すれば直接演算可能になる．

以下の2つの表は各タイプの間の直接演算可能性についての表である．○印は可能，×印は不可能を示し，○の場合の記号はその演算結果のタイプを示している．又LOPT, ROPTは夫々のoperatorにとつて左側，右側のargumentのタイプを示す．×印の個所はパターンの修正を必要とする．

( )，・に関する直接演算可能性の表

各entryの左側は( )，右側は・について

LOPT \ ROPT	atom & T1		T2		T3		T4		T5	
atom & T1	○ T1	○ T1	×	○ T4	×	○ T1	×	○ T4	×	○ T4
T2	×	○ T3	×	×	×	×	×	○ T5	×	×
T3	×	○ T3	×	○ T5	×	○ T3	×	○ T5	×	○ T5
T4	×	○ T1	×	×	×	×	×	○ T4	×	×
T5	×	○ T3	×	×	×	×	×	○ T5	×	×

（注意） 下側のTypeは演算結果のタイプ

＊に関する直接演算可能性の表

LOPT	
atom & T1 	○ T2
T2 	×
T3 	×
T4 	×
T5 	×

1.4　SG作成の例による説明

任意のREをLogical circuit（LC）又はSGへ変換する手順を例題で説明する．尚一般的なプログラミングの説明は第2章で行なう．

例題1.　$RE=A\bullet(A\cup B\cup C)^{*}$

Step 1.　RE to REL　このstepはREを直接演算可能な基本RE（BRE, basic regular expression）のリストRELに変換する．

BREのリスト表現は次の形式である．

（BREのタイプ, Operator, arg 1, arg 2, ····, arg n）

ここで，argument (arg i)は又REのリスト表現であつてもよい．このstepの手法はコンパイラーの式のanalysis partに類似している．

この例のREのリスト表現は

$$REL=(T1,\bullet,A,(T2,*,\underbrace{(T1,\cup,A,B,C)}_{a}))$$

$$\underbrace{\underbrace{\quad}_{b}}_{c}$$

である．又a, b, cは夫々BREである．即ち$b=(T2,*,a)$, $c=(T1,\bullet,A,b)$ である．

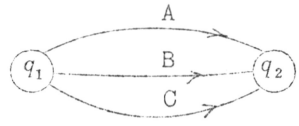
Step 2. REL→SG　このstepはいくつかのsub-stepsに分れる．

Step 21. REL→{BSG (basic state graph)}の変換　ここでは各BREを対応するBSGに変換する．

BRE $a=(T1, \cup, A, B, C)$ に

$BSG(a)=((\text{initial } q_1)(\text{terminal } q_2)(A(q_1, q_2))$
$(B(q_1, q_2))(C(q_1, q_2)))$

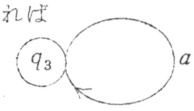
を対応させる．これを図示すると



BRE $b=(T2, *, a)$ に

$BSG(b)=((\text{initial } q_3)(\text{terminal } q_3)(a(q_3, q_3)))$

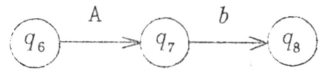
図示すれば



BRE $c=(T1, \bullet, A, b)$ に

$BSG(c)=((\text{initial } q_6)(\text{terminal } q_8)(A(q_6, q_7))$
$(b(q_7, q_8)))$

図示すれば

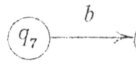


以上の如くRELはBSGの集合にかわる．

Step 22. SGの代入

step 21でできたリストBSG(c)にb，つきにaを代入してargumentsが全てatomであるリストを作る．

(i) BSG(b)をBSG(c)に代入

BSG(c)の  にBSG(b)を代入するのであるが，bはT2タイプでであるから，q_7, q_8 は同一のものと考える．この代入の結果を次の如くかく．

$\rho_b(c)=((\text{initial } q_6)(\text{terminal } q_7)(A(q_6, q_7))(a(q_7, q_7)))$

(ii) SG $\rho_b(c)$ にBSG(a)を代入

この結果は次の如くになる．

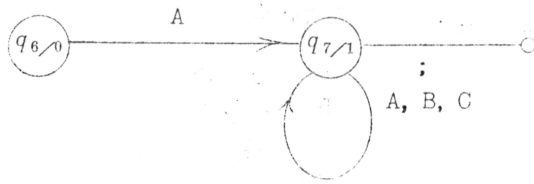
$\rho_{ab}(c)=\rho_a(\rho_b(c))=((\text{initial } q_6)(\text{terminal } q_7)(A(q_6, q_7)(q_7, q_7))$
$(B(q_7, q_7))(C(q_7, q_7)))$

Step 2. ϕ-arcの消去

この例にはϕ-arcがないから不要.

Step 3. minimalization

この例ではstep 2で作成された$\rho_a(\rho_b(c))$は極小であるからこの手順は不要である.
補助記号として';'を使用すれば A・(A∪B∪C)*のstate graphは次の図になる.



例題2 RE=A∪A*・B A, Bはatomとする.

A*・BはT3タイプであるから, ∪演算は直接演算可能ではない. よつて$\tau(A^*)=A^{*3}$を行なう. この修正の操作はPush down stackによつて実行する.

(i) まずA*が演算可能であるからリストaができる.

a=(T2, *, A)

(ii) 次にa・Bが演算可能なのでリストbを作る.

b=(T3, ・, a, B)

(iii) A∪bは上記のようにbを修正する. 即ちbの左端のargument aをT2タイプからT1タイプにかえてA^{*3}にする. このためリストaは次のように修正される.

a=(T1, *3, A)

したがつてリストbも次のように修正される.

b=(T1, ・, a, B)

こうしてA∪bは演算可能になるからリストcは

c=(T1, ∪, A, b)

となる.

BSGへの変換

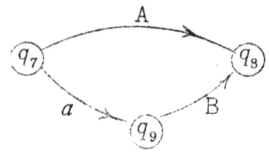
BSG(a)=((initial q_1)(terminal q_3)($\phi(q_1, q_3)$)(A(q_1, q_3)(q_2, q_2)
(q_2, q_3)))

BSG(b)=((initial q_4)(terminal q_6)($a(q_4, q_5)$)(B(q_5, q_6)))

BSG(c)=((initial q_7)(terminal q_8)(A(q_7, q_8))($b(q_7, q_8)$))

代入の実行

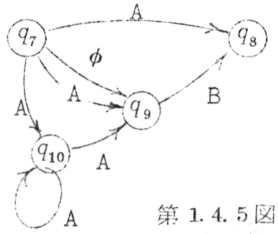
$\rho_b(c)$=((initial q_7)(terminal q_8)(A(q_7, q_8))($a(q_7, q_8)$)
(B(q_7, q_3)))



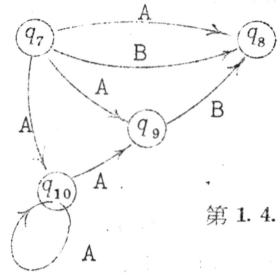
第 1. 4. 4 図

BSG (c) の arc $b(q_7, q_8)$ の上に BSG (b) の copy が重ねられる.

$\rho_a(\rho_b(c))$=((initial q_7)(terminal q_8)($\phi(q_7, q_9)$)
($A(q_7, q_8)(q_7, q_{10})(q_{10}, q_{10})(q_{10}, q_9)$)($B(q_9, q_8)$))



第 1. 4. 5 図



第 1. 4. 6 図

以上によつて一応 RE に対応する SG (第 1,4 ,5 図参照) が出来る.

ϕ-arc の処理

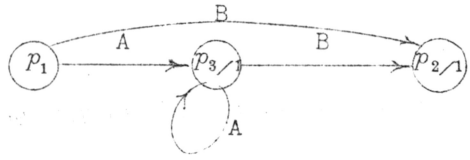
$\rho_a(\rho_b(c))$ は ϕ-arc を含むからこの処理を行なう. $\phi(q_7, q_9)$ と $B(q_9, q_8)$ を接続すると, $B(q_7, q_8)$ になる. よつて, この arc を附加して, もとの ϕ-arc を削除する. 第 1. 4. 6 図参照.

Decided graph への変換

第 1. 4. 6 図にえられた SG は A に関して decided ではないのでこれを decided に変更する. その結果は

((initial p_1)(terminal p_2, p_3)($A(p_1, p_3)(p_3, p_3)$)
($B(p_1, p_2)(p_3, p_2)$))

になる. 図示すれば第 1. 4. 7 図になる.



§2. プログラミングの説明

1.4 で例題によつて説明したプログラムの主要部分をこの節で概説する.

2.1 Step 1のプログラム　Step 1はRE→RELのmain part, Eop, {MODIFY(∪), MODIFY(•), MODIFY(*)}及びMakelistのSubroutineより成立つ.

2.1.1 RE-RELのmain part (表2.1)　Step 1 全体のコントロール部分である.

表 2.1

RE-REL main part

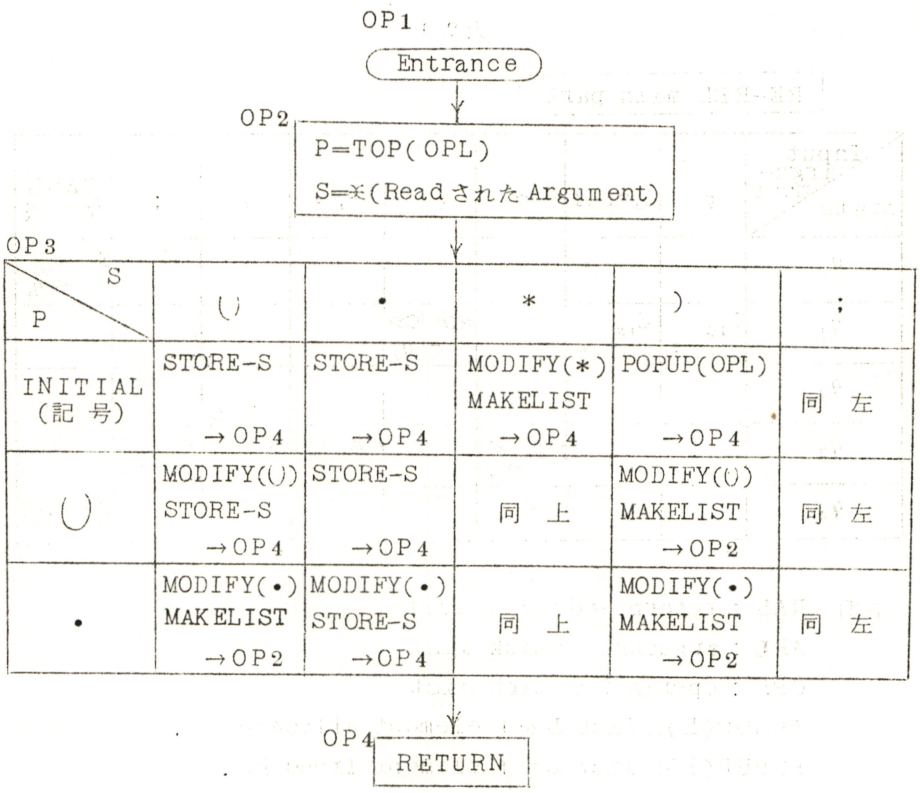
state \ Input argument	ϕ	symbol	*	(	∪, •	)	;	CALL情報
q_0								$E_{0CALL} \rightarrow q_1$
q_1	$E_{1\phi} \rightarrow q_3$	$E_{1S} \rightarrow q_3$		≪RE≫ $\rightarrow q_2$				
q_2						$\rightarrow q_3$		
q_3			<E_{op}> $\rightarrow q_3$		<E_{op}> $\rightarrow q_1$	$E_{3)} \rightarrow q_2$	<E_{op}> $\rightarrow q_4$	
q_4								E_{return}

説明　RAL : return address の list
APL : argument の stack list
OPL : operator の stack list
ALLOC(L)はlist Lを1 element allocateする.
POPUP(L)はlist Lを1 element freedする.
PUSHD(L, X)はlist Lのtopに Xをstackする.
E_{0CALL} : ALLOC(RAL), PUSHD(RAL, return address)
ALLOC(APL, OPL), PUSHD(APL, OPL; 'Initial');
$E_{1\phi}$: ALLOC(APL), PUSHD(APL; 'ϕ');
E_{1S} : ALLOC(APL), PUSHD(APL; symbolの登録番号)
$E_{3)}$: <E_{op}>,)を戻す;
<E_{op}> : subroutine E_{op} の calling
≪RE≫ : RE-REL subroutineのrecursive calling
E_{return} : return address:=TOP(RAL), POPUP(RAL)
GOTO(return address);

2.1.2 E_{op} (表2.2)　OPL上のoperator (P)と新たによみこまれたoperator (S)とのpair (P, S)の状態によつて演算可能なoperatorを検出し対応するActionをコントロールする.

表2.2

E_{op} フローチャート



P ＼ S	( )	•	*	)	;
INITIAL (記号)	STORE-S →OP4	STORE-S →OP4	MODIFY(*) MAKELIST →OP4	POPUP(OPL) →OP4	同左
( )	MODIFY(( )) STORE-S →OP4	STORE-S →OP4	同上	MODIFY(( )) MAKELIST →OP2	同左
•	MODIFY(•) MAKELIST →OP2	MODIFY(•) STORE-S →OP4	同上	MODIFY(•) MAKELIST →OP2	同左

(註) STORE-S: ALLOC(OPL), PUSHD(OPL; S)

2.1.3 MODIFY(( )), MODIFY(•), MODIFY(*) (表2.3, 2.4, 2.5)

E_{op}で演算可能と判定されたoperatorのoperandsがそのoperatorに対して直接演算可能かどうかを調べる, 又必要ならばoperandsのタイプの修正を行なう. さらにそのoperatorをoperandsにapplyしたとするときのexpression (<( ) or•>, LO, RO)(*, LO)のタイプ(これをaccumulative type ACTという)を決定して

ACT(TOP(APL))に入れる.

表2.3

MODIFY(∪)

ENTRANCE

LOPT \ ROPT	ATOM & T1	T2	T3	T4	T5
ATOM & T1		AR3(i)	AR3(L, i)	AR3(R, i)	AR3(L, i) AR3(R, i)
T2	AR3(i-1)	AR3(i) AR3(i-1)	AR3(L, i) AR3(i-1)	AR3(R, i) AR3(i-1)	AR3(R, i) AR3(L, i) AR3(i-1)
T3	AR3(L,i-1)	AR3(i) AR3(L,i-1)	AR3(L, i) AR3(R, i-1)	AR3(R, i) AR3(L,i-1)	AR3(R, i) AR3(L, i) AR3(L, i-1)
T4	AR3(R, i-1)	AR3(i) AR3(R,i-1)	AR3(L, i) AR3(R, i-1)	AR3(R, i) AR3(R,i-1)	AR3(R, i) AR3(L, i) AR3(R, i-1)
T5	AR3(R, i-1) AR3(L, i-1)	AR3(i) AR3(R,i-1) AR3(L,i-1)	AR3(L, i) AR3(R, i-1) AR3(L, i-1)	AR3(R, i) AR3(R,i-1) AR3(L,i-1)	AR3(R, i) AR3(L, i) AR3(R, i-1) AR3(L, i-1)

SACT(T1)

RETURN

(註) AR3(i) : TOP→*3

AR3(i-1) : 2ND→*3

AR3(L, i) : LO(TOP)→*3

AR3(R, i) : RO(TOP)→*3

AR3(L, i-1) : LO(2ND)→*3

AR3(R, i-1) : RO(2ND)→*3

ここでTOPはTOP(APL), 2NDは2ND(APL)のこと.

表 2.4

MODIFY(∘)

ENTRANCE

ROPT / LOPT	ATOM & T1	T2	T3	T4	T5
ATOM & T1	SACT(T1)	SACT(T4)	SACT(T1)	SACT(T4)	SACT(T4)
T2	SACT(T3)	AR3(i) SACT(T3)	AR3($i-1$) SACT(T1)	SACT(T5)	AR3($i-1$) SACT(T4)
T3	SACT(T3)	SACT(T5)	SACT(T3)	SACT(T5)	SACT(T5)
T4	SACT(T1)	AR3(i) SACT(T1)	AR3(L, i) SACT(T1)	SACT(T4)	AR3(L, i) SACT(T4)
T5	SACT(T3)	AR3(i) SACT(T3)	AR3(L, i) SACT(T3)	SACT(T5)	AR3(L, i) SACT(T5)

RETURN

表 2.5

MODIFY(*)

ENTRANCE

LOPT	
ATOM & T1	
T2	AR3(i)
T3	AR3(L, i)
T4	AR3(R, i)
T5	AR3(L, i), AR3(R, i)

SACT(T2)

RETURN

2.1.4 Makelist BREのリスト

(ACT, operator, arg 1, arg 2, ⋯, arg n)

を作るためのサブルーチンである. arg i はatom又はリストである. operatorは, , •, *, *3のいずれかである.

最終的にMakelistによりREL={BRE}ができる.

2.2 Step 2のプログラム

Step 2のプログラムは1.4でみたように

Step 21 REL={BRE}→{BSG}への変換

Step 22 {BSG}の逐次的代入

Step 23 ϕ-arcの処理

Step 24 undecided graphからdecided graphへの変換
minimalizationの手続き

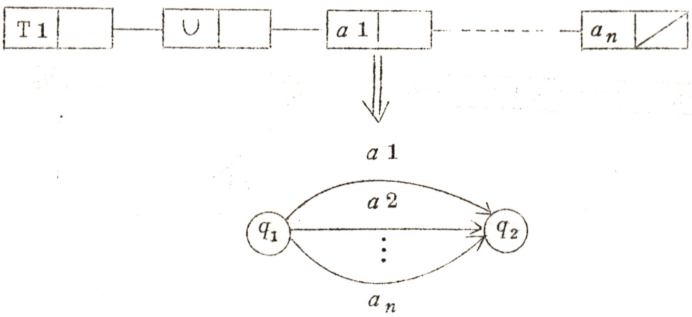
に分けられる.

2.2.1 Step 21のプログラム ここで用いられるBREとBSGとの対応は次の如く定義される.

(a) BRE(∪)→BSG(∪)

リスト表現 (T1, ∪, $a1$, $a2$, ⋯, a_n)

図式表現



(b) BRE(•)→BSG(•)

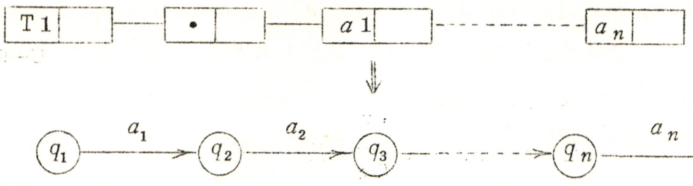
リスト表現

(T1, •, $a1$, $a2$, ⋯, a_n)

→((initial q_1)(terminal q_{n+1})

(a_1(q_1, q_2, 0))(a_2(q_2, q_3, 0))⋯(a_n(q_n, q_{n+1}, 0)))

図式表現

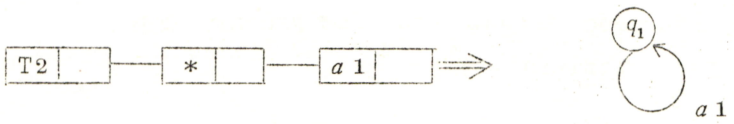


(c) BRE(*) → BSG(*)

リスト表現

(T2, *, a 1) → ((initial q_1)(terminal q_1)(a_1(q_1, q_1, 0)))

図式表現



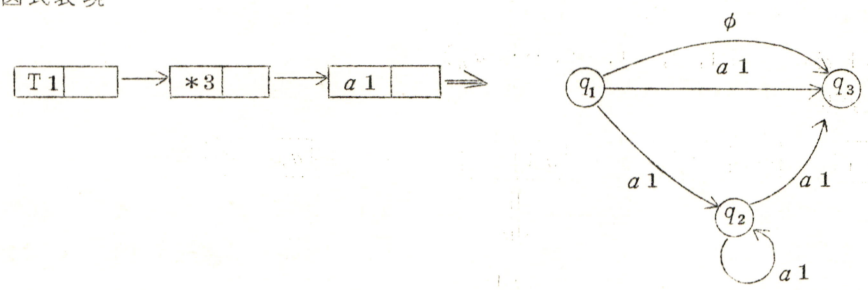
(註) a 1はT1タイプであることが必要.

(d) BRE(*3) → BSG(*3)

リスト表現

(T1, *3, a 1) → ((initial q_1)(terminal q_3)(ϕ(q_1, q_3, 0)))
(a 1(q_1, q_3, 0)(q_1, q_2, 0)(q_2, q_2, 0)(q_2, q_3, 0)))

図式表現



(註) $(a\,1)^{*3} = \phi \cup a1 \cup a1 \cdot a1^{*} \cdot a1$ であるから

(T1, *3, a 1) ≡ (T1, ∪, φ, a 1, $a1 \cdot a1^{*} \cdot a1$)

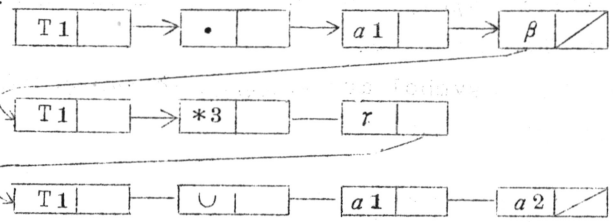
となる. よつてBRE(*3)とBSG(*3)の対応が出来る. ここでφは他のsymbol と同じに扱うがφの処理参照.

2.2.2 Step 22 {BSG}の代入

REがリストでかかれているときこのRELに含まれるBREのlevelを次の様に定める.

BRE=(type, operator, arg 1, arg 2, ⋯, arg n)の全てのargがatom(ϕも含む)であるときこのBREはlevel 1という. BREのarg i($1 \leqq i \leqq n$)の中に, 少くとも1つlevel ($k-1$)のBREがあるとき, level kであるという. atomはlevel 0のBREと考える.

(例) (α)



ここで level (α)=3, level (β)=2, level (γ)=1.

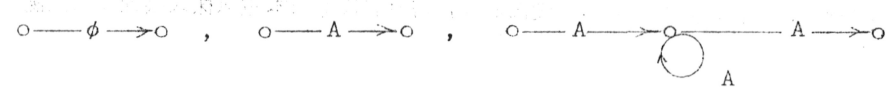
REL={BRE}のBREの中最も高いlevelをkとすればlevel kのBRE は1つだけである. このBRE(k)に順次低位のBREを代入することにより1位まで下げる操作を{BSG}の代入という.

2.2.3 Step 23 ϕ-arcの処理

Step 21, Step 22で作られたSGが

$$SG=((\text{initial } q_s)(\text{terminal } q_t)$$
$$(\phi(q_{i\phi,1}, q_{j\phi,1}) \cdots (q_{i\phi,m\phi}, q_{j\phi,m\phi}))$$
$$(a_1(q_{i1,1}, q_{j1,1}) \cdots (q_{i1,m1}, q_{j1,m1}))$$
$$\vdots$$
$$(a_k(q_{ik,1}, q_{jk,1}) \cdots (q_{ik,mk}, q_{jk,mk})))$$

であるとする. Step 21におけるREの修正の際もしMODIFY(•)の代りにMODIFY(∪)を使用したとすれば次の事がいえる. SGの各state間を接続するarcはT1タイプのargumentである. 即ち

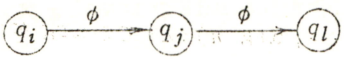


のいずれかである.

ϕ-arc処理の手順

(1) ϕ-arcリストに含まれる1つのarcを$\phi(q_i, q_j)$但し$q_j \neq q_k$とする. 他の全てのϕ-arc $\phi(q_k, q_l)$について, もし$q_k = q_j$ならば$\phi(q_i, q_j)$と$\phi(q_k, q_l)$は接続

しているといい



となるから，$\phi(q_i \cdot q_j) \cdot \phi(q_k, q_l) \rightarrow \phi(q_i, q_l)$ とすることができる．よつて新しい $\phi(q_i, q_l)$ を ϕ−arc リストに附加する．

(2) $\phi(q_i, q_j)$ に対して，symbol arc $a_s(q_u, q_v)$ があり，$q_j = q_u$ ならば $\phi(q_i, q_j)$ と $a_s(q_u, q_v)$ は接続しているといい，

となつているから $\phi(q_i, q_j) \cdot a_s(q_u, q_v) \rightarrow a_s(q_i, q_v)$ とすることが出来る．よつて $a_s(q_i, q_v)$ を symbol−arc リストに附加する．

(3) (1),(2) の演算を1つの $\phi(q_i, q_j)$ に対して，それ以外の全ての ϕ−arc 及び全ての symbol−arc について行なつたとき，$\phi(q_i, q_j)$ に対する演算は終了したといい ϕ−arc リストから取除く．

(4) 新しい ϕ−arc リストに対して(1)〜(3)を繰返して行ない，ϕ−arc リストに含まれる全ての ϕ−arc が $\phi(q_i, q_t)$ の形になつたとき，与えられた SG に対する演算は全て終了したという．

このとき ϕ−arc リストに含まれる arc の各始点は全て q_t に ϕ で接続している．したがつて ϕ−arc リストに含まれる全ての states は SG の terminal state とみなすことが出来る．

2.2.4　minimalization の説明は省略．

2.2.5　応用としてコンパイラーの syntax の自動解析がある．

おわりに　アルゴリズムの programming は PL／1 言語で行なつている．この問題について種々，助言，協力を戴いた渡部和氏，若月宏氏（日本電気株式会社）に感謝する．

参考文献

〔1〕 R.E.Miller : Switching Theory (Volume Ⅱ) Sequential Circuits and Machines.1965.

本 PDF ファイルは 1968 年発行の「第 9 回プログラミング一シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html