グラフ処理用言語 GML-56†

杉藤芳雄†† 真野芳久†† 鳥居宏次††

我々は，視覚的に把握しやすいというグラフの利点を生かしたグラフ処理言語GMLを既に提案し実際にインプリメントしてきた．本論文は，このシステムをTosbac 5600計算機のTSS環境下に移行する機会にGMLに検討を加え，これまでの経験に基づいて再設計されたグラフ処理言語 GML-56 について述べる．

GML-56 プログラムは二つの部分から成る．一つはプログラムの制御をつかさどる部分で，これはいくつかのGML-56ライブラリ・ルーチンを含んだ Fortran プログラムの形をとる．他の一つはグラフの変換規則であり，これは二次元に表現された1対のグラフによって表わされる．

本論文はまず GML-56 の言語仕様について述べる．GML との主な相違点は，使用環境の違いのためブロック線図によるプログラム制御の表現を Fortran ベースのテキストに変更したこと，グラフの表現能力や変換機能をより強化するため辺のラベルを導入し点の分離や重ね合せを可能としたこと，backtracking を容易に行えるようにしたこと，等である．

さらに GML-56 プログラムの例を示す．これは与えられた連結無向グラフのブロックを認識するもので，出力として一つのブロックを1辺に対応させたグラフを求めている．

最後に GML-56 およびグラフ表現方式についての評価を試みる．

1. まえがき

グラフは理論面からも実際面からも興味深いものであるが，いずれの場合にも「グラフの変換」が重要な役割を果している．この反映として，グラフの変換を目的とする言語がいくつか提案されている[1]．しかし，これらの言語の大部分は文字列のみによるグラフの変換の表現法を用いており，グラフを対象としていながら，視覚的に把握しやすいというグラフの利点を等閑視している傾向が見受けられる．我々はこのグラフの利点に着目して，既に，グラフの変換やプログラムの構造の表現法にも二次元的グラフ表現を積極的に利用したグラフ処理用言語 GML を提案し[2]，その支援システム GMS をも製作した[3]．GMS は Hitac-8410 のバッチシステム上でライトペン式グラフィックス端末を用いたシステムであったが，このシステムを Tosbac-5600 の TSS 上（カーソル式グラフィックス端末）に移行する機会に GML の言語仕様にも大幅な変更を施した．この結果，新しい言語とみなしうるほどになり GML-56 と命名した．本稿では，この新しいグラフ処理用言語 GML-56 に関して，その言語仕様を詳細に述べることによりテキスト表現とグラフ表現との融合という特徴を明示し，その記述例として連結グラフのブロックを認識する問題を扱うことにより本言語の有効性を示し，そして本言語を実際にインプリメントした結果をふまえてグラフ処理用言語としての評価を行う．

2. 設計方針およびその結果

GML の使用経験，新しい環境への適応，移行労力の軽減，の3点が本言語設計上の基礎となっているが以下に主要方針を列挙する．

（1） GML ではグラフ的表現にできる限り徹するということから，グラフの変換規則表現のみならずプログラムの構造の表現をもグラフの一種と考えられるブロック線図形式にしていた．しかし，グラフィックス端末の画面の大きさの制約，ライトペン方式でないことによる図形操作の多少の不便さ，移行労力の軽減とからブロック線図形式は不向きと判断されるため本言語ではプログラムの構造の表現にはテキスト形式を採用する．

（2） グラフの変換規則の表現にグラフを用いることは GML においても有益であったし一大特徴でもあるので本言語においてもそれを踏襲する．

（3） プログラムの構造の表現に用いられるテキスト形式は拡張されたライブラリを持つ Fortran とする．これは，既存の TSS 用 Fortran コンパイラを直接利用できること，Fortran が普及度の高い言語であること，容易にインプリメントされるプリプロセサを利用することで構造的なプログラムを記述することも可能であること*とによるものである．

† Graph Manipulation Language GML-56 by Yoshio Sugito, Yoshihisa Mano, and Koji Torii (Computer Science Division, Electrotechnical Laboratory).

†† 電子技術総合研究所ソフトウェア部

* 実際，構造的 Fortran 言語である Westran[5] と共に GML-56 を使用できるようになっている．

表 1 GML-56 と GML との機能的な相違点
Table 1 Functional differences between GML-56 and GML.

項目		GML-56	GML
プログラムの形式		FORTRAN テキスト（コンパイル方式）	ブロック線図形式（インタプリタ方式）
グラフの表現	ラベルの種類	点ラベル，辺ラベル	点ラベル
	自己ループ	あ り	な し
ラベル変数の表現		グラフ中では &n，プログラム中では共通領域中の変数．共通領域中の位置によって結合する(3.3節参照)．	プログラム中もグラフ中も同一の名前で表現できる．
マッチングとエンベッディングの対応付け		規則番号（変数の使用も可）	制御の流れ
マッチング結果の保存		各規則ごとに高々一つ	最新のマッチングのみ
エンベッディングの結果		成功または失敗	常に成功
点の分離，重ね合せの機能		あり (3.5 節参照)	な し
中間結果の保存機能 (backtracking 用)		あり（表 3 (8)(9) 参照）	な し

（4） グラフの表現能力，グラフの変換機能をより強化する．たとえば，辺のラベルを許し，点の分離や重ね合せを可能にする．

（5） backtracking が容易に行えるようにするため，グラフの変換過程の中間結果の保存機能を考える．

この設計結果は 3 章で詳細に示されるが，本言語と GML との機能上の主な相違点を**表 1** に要約する．

3. 言語仕様

GML-56 は，プログラム・テキスト上からは FORTRAN をホスト言語とし，グラフ処理関係の機能を FORTRAN のサブルーチンや関数として組み入れたものである．変換規則表現用のグラフは，その規則番号だけがプログラム・テキスト上に現れる．また，入力グラフは常に 1 種類だけが処理対象とみなされていることから識別名などの必要がなく，プログラム・テキスト上には明示されない．そして，これらグラフのデータ構造がプログラム・テキスト上に登場することは一切ない．本言語のユーザは，グラフィックス端末上で図形としてのグラフ（外部表現）の形でこれらのグラフを定義するので，これを然るべきデータ構造（内部表現）に変換するためのユーティリティ（グラフ・エディタ）の存在を想定している．ここで「グラフ・エディタ」とは，図形としてのグラフを対象とするエディタであり，グラフに関する各種の編集機能（たとえば点の追加・削除・移動など）が用意されている．そして，グラフィックス端末上の外部表現とメモリ上の内部表現との間での両方向の変換をつかさどるものである．しかしながら，グラフ・エディタやデータ構造に関する仕様は明らかに本言語仕様の範囲外であるので本稿では触れない．GML-56 におけるプログラム・テキスト，諸グラフおよびグラフ・エディタの関係を**図 1** に示す．

* GML ではウェブ文法の影響を受けて，エンベッディングは常に成功することが仮定されていたが，本言語では成功しない場合を許している．

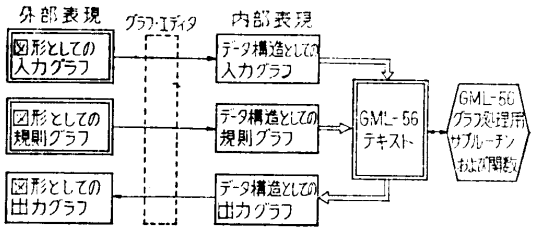

図 1 GML-56 におけるプログラム・テキスト，諸グラフ，グラフ・エディタの関係（二重線の部分は利用者が作成する）
Fig. 1 Relations among text, graphs and graph-editor in GML-56. (The contents in doubled-lined box are defined by users.)

以降では，諸グラフの外部表現，変換規則，グラフ処理用サブルーチン／関数，等について詳述する．

3.1 入力グラフ

入力グラフは GML-56 プログラムの入力データに相当するものであり，通常の点 (node) および有向または無向の辺(edge)から成り，点や辺にはラベル（それぞれ点ラベル，辺ラベルとよぶ）が添えられているものであって，これをとくに点・辺ラベル付きグラフとよぶ．2 点間の同一方向での多重辺は許されず，同一の点に対する辺（自己ループ辺）は許される．

なお，GML-56 プログラムの実行結果として出力されるグラフ（出力グラフとよぶ）の外部表現が入力グラフのそれと同様であることは言うまでもない．

3.2 変換規則

グラフの変換規則の表現にはやはりグラフが用いられるが，このための一対のグラフを規則グラフ（RG と記す）とよぶ．変換規則は，α, β を一対の RG とするとき〔規則番号，α, β〕の三つ組から成る．これは与えられた入力グラフ G 内に α と同形な部分グラフ G_α が存在するか否かを判定（この操作をマッチングとよぶ）して，もし存在すれば（これをマッチング成功とよぶ）G_α を β と同形なグラフで置きかえることを試みる*（この操作をエンベッディングとよぶ）という方法により，グラフの変換を行うことを意味す

る．なお，与えられた入力グラフ内でαを反復的にマッチングさせる場合，すでに調べた部分を除外してマッチングさせることを，とくにリマッチング (rematching) とよぶ．また，α,βをそれぞれマッチング規則グラフ (MRG と記す)，エンベッディング規則グラフ (ERG と記す) とよぶ．

マッチング，エンベッディングに関する詳細は3.4節および3.5節で述べられる．

ここでRGの外部表現について述べる．RG は基本的には入力グラフと同じく点・辺ラベル付きグラフである．RG に固有なものとしては，①MRG と ERG 間の点の対応関係を定める点対応番号，②ラベルの表現法の一部分，③とくに MRG に固有なものとして点や辺に関するマッチング条件を指定する特殊記号，がある．入力グラフおよび規則グラフの外部表現を**表2**に示す．

表2 入力グラフと規則グラフの外部表現
Table 2 External representation on input graphs and rule graphs.

要素＼グラフ		入力グラフ	規則グラフ MRG	規則グラフ 特殊記号	規則グラフ ERG
点 (iは点対応番号)		○	ⓘ	◎	ⓘ
辺	無向	— ○	— ○	—* ○	— ○
	有向	←— ○	←— ○	←*— ○	←— ○
点・辺ラベル		$l(x_1,\cdots,x_n)$			
ラベル名l		定数	定数または＊(don't care 記号)		
サフィックスx_i		定数	定数または＊または&$_i$(ラベル変数)		

3.3 サブルーチン／関数

グラフの変換に寄与する各種のサブルーチン／関数が**表3**に示すごとく用意されている．これらはデータ構造（内部表現）を処理対象としており，いずれもXで始まる名前をもつ FORTRAN のサブルーチン／関数である．同表において，与えられた入力グラフの変換過程における現在形のことを CIG (Current Input Graph) と記す．

ここで(3),(4),(5)の関数間の関係について説明しよう．(3)や(4)が成功すると，同定された CIG 内の部分グラフG_αに関する情報（これをG_α情報とよぶ）が該当する規則番号ごとに保存される．ただし，一つの規則番号に対して最新のものだけが保存される．(5)は，同じ規則番号のG_α情報に対して適用され，それが成功するとその時点で保存されていたすべての規則番号に関するG_α情報が破棄される．もちろん，(5)はその規則番号に対するG_α情報が保存されていない場合には失敗（すなわち関数値が偽）となる．以上のことを例により以下に説明する．

〔例〕 XMATCH(1) …… ①
XRMATC(1) …… ②
XMATCH(2) …… ③
XEMBED(2) …… ④
XEMBED(1) …… ⑤

いま，上記の順に実行して①,②,③がいずれも成功した場合を考える．④では，同じ規則番号“2”に関するG_α情報，すなわち③の結果に対してエンベッディングが試みられる．それが成功の場合には，すべての規則番号に関するG_α情報が破棄されるので⑤の時点では規則番号“1”に関するG_α情報が存在せず，⑤の結果は失敗に終る．逆に④が失敗の場合には，すべてのG_α情報は④に入る直前の状態のまま保存されているので，⑤の時点では規則番号“1”に関する最新のG_α情報，すなわち②の結果に対してエンベッディングが試みられることになる．

(7)の関数について補足する．一般に点・辺ラベル$l(x_1,\cdots,x_n)$は長さ$n+1$の配列を内部表現として想定しているが，この関数はこの配列の要素を構成する際に用いられ，これにより XNLAB や XELAB の引数を設定することになる．外部表現&$_j$のラベル変数は内部表現としては定められた名前の COMMON 領域（XCLVAL とする）内の第j要素となることを想定している．したがって GML-56 の FORTRAN テキスト中に XCLVAL の COMMON 宣言をすることにより，ラベル変数の値の参照や変更が可能となる（図4参照）．

3.4 マッチング

CIG 内に，指定された MRG と同形な部分グラフが存在するか否かの判定をマッチングとよび，存在すればマッチング成功という．具体的に言うと，次の2条件が満足されるときマッチング成功となる．

(i) 後述のラベルの一致の定義に基づき，点・辺にラベルの付いたグラフとして，CIGの部分グラフに MRG と同形のものが存在する．

(ii) MRG 内の点や辺に特殊記号が用いられている場合には，その条件をも満足している．

点の特殊記号としては◎があり，MRG 内の◎に対応する CIG 内の点の次数（とくに有向グラフの場合には出次数および入次数）が ◎ に図示されている

表 3 GML-56 ライブラリとして用意されるサブルーチンおよび関数
Table 3 Subroutines and functions in GML-56.

番号	関数／サブルーチン名（引数）	説明
(1)	XINIT（有向／無向）	入力グラフ・データの初期処理を行う．たとえば，グラフ・エディタを介して入力グラフ用ファイルに格納されていた入力グラフ（内部表現）をコア上に移す．
(2)	XTERM	GML-56 プログラムの実行により変換されたグラフ・データの終了処理を行う．たとえば，コア上のグラフ・データ（内部表現としての出力グラフ）を出力グラフ用ファイルに格納する．
(3)	XMATCH（規則番号）	CIG と，規則番号で指定される MRG とのマッチングを試みる論理型関数．その真偽によりマッチング成功／失敗を表わす．
(4)	XRMATC（規則番号）	CIG と，規則番号で指定される MRG とのリマッチングを試みる論理型関数．その真偽によりマッチング成功／失敗を表わす．ただし，同一の MRG に対してマッチングがまだ行われていない場合には XMATCH と同じである．
(5)	XEMBED（規則番号）	CIG と，規則番号で指定される MRG とのマッチング（またはリマッチング）成功により同定されていた CIG 中の部分グラフに対して，同じ規則番号で指定される ERG のエンベッディングを試みる論理型関数．その真偽によりエンベッディング成功／失敗を表わす．エンベッディング失敗の場合，CIG の状態はそのエンベッディングを試みる以前のままである．
(6)	XNODE	CIG 内の点の総数．
	XEDGE	CIG 内の辺の総数．
	XDEG（点対応番号）	CIG 内の，与えられた点対応番号で指定される点の次数（有向グラフの場合には入次数と出次数の和）．
	XDEGI（点対応番号）	CIG 内の，与えられた点対応番号で指定される点の入次数．
	XDEGO（点対応番号）	CIG 内の，与えられた点対応番号で指定される点の出次数．
	XMAXD	CIG 内の点の次数の最大値．
	XMAXDI	CIG 内の点の入次数の最大値．
	XMAXDO	CIG 内の点の出次数の最大値．
	XMIND	CIG 内の点の次数の最小値．
	XMINDI	CIG 内の点の入次数の最小値．
	XMINDO	CIG 内の点の出次数の最小値．
	XNLAB（点ラベル）	CIG 内の，与えられた点ラベルと一致する点ラベルをもつ点の総数．
	XELAB（辺ラベル）	CIG 内の，与えられた辺ラベルと一致する辺ラベルをもつ辺の総数．
(7)	XSVAL（指定値）	点・辺ラベルの外部表現（表 2 参照）における *, &j 等を指定値により設定し，しかるべき内部表現に変換する関数．
(8)	XSAVE（CIG 番号）	入力グラフの変換過程における中間結果の各グラフに対しユーザが適宜番号付けを行う（この番号を CIG 番号とよぶ）ことにより，GML-56 プログラムの任意のステップで CIG を中間結果用ファイルに格納する．その際，格納すべき CIG の CIG 番号の値以上の CIG 番号をもつ CIG 群は中間結果用ファイルから除去される．
(9)	XRLOAD（CIG 番号）	GML-56 プログラムの任意のステップにおいて与えられた CIG 番号に対応する CIG を中間結果用ファイルからコアに移し，それを以降の CIG とする．その際，その時点で保存されていたすべての G_α 情報は破棄される．
(10)	XBREAK（識別番号）	GML-56 プログラムの任意のステップにおいてプログラムを中断し，その時点の CIG をグラフ・エディタを介して画面表示する．その際副作用の心配がなければ（たとえばエンベッディング成功の直後），グラフ・エディタによる CIG の編集（点や辺の追加・削除など）が可能である．このあと，プログラムの中断時点にもどって再実行できる．引数の識別番号は，他の中断箇所と区別するために用いられ，その値が画面表示される．

ものに丁度等しいことを意味する．

辺の特殊記号としては—#—があり，MRG 内の—#—に対応する CIG のその部分には，—#—に図示されているラベルの付いた辺が存在しないことを意味する．

最後に，前述の「ラベルの一致」について詳述する．MRG におけるラベル変数は，マッチング直前の時点ですでにある値が代入されている場合には確定状態にあるといい，まだ値が代入されていない場合には未定状態にあるという．入力グラフ，MRG 中のラベルをそれぞれ $l_G(c_1,\cdots,c_m)$, $l_\alpha(x_1,\cdots,x_n)$ とするとき，両者が一致する条件とは $m=n$ かつ表 4 の条件が成立することである．マッチングが成功すれば，x_i における未定状態ラベル変数には対応する c_i が，代入されて確定状態となる．

表 4 ラベルに関するマッチング条件
Table 4 Matching conditions on labels.

ラベル要素	状態		一致条件
ラベル名 l_α	定数		$l_\alpha=l_G$
	* (don't care)		l_G は任意
サフィックス x_i	定数		$x_i=c_i$
	ラベル変数	確定状態	$a_i=c_i$ (a_i は x_i の確定値)
		未定状態	c_i は任意
	* (don't care)		c_i は任意

3.5 エンベッディング

同一の規則番号 r をもつMRG, ERGをそれぞれ α_r, β_r とする．そして XMATCH (r) または XRMATC (r) の成功により同定された CIG の部分グラフを G_α

とする．この G_α に対して，α_r を参照しつつ β_r で置きかえる操作をエンベッディングという．エンベッディングの方法は，RG の点に添えられる点対応番号により規定される．ある点対応番号 N が α_r 上に p 個，β_r 上に q 個ある場合，エンベッディング成功によるそれらの点の変換結果は次のようになる*．

$p=0$, $q\geq 1$: 点の追加（G_α において，β_r 上の N に相当する点が追加される）

$p=1$, $q=1$: 点の残留（G_α において，α_r 上の N に相当する点を β_r 上の N を持つ点で置きかえる）

$p=1$, $q>1$: 点の分離（後述）

$p\geq 1$, $q=0$: 点の除去（G_α において，α_r 上の N に相当する点，およびその点に接合している辺が除去される）

$p>1$, $q=1$: 点の重ね合せ（後述）

以上のいずれの場合にも，点の増減に伴って辺の増減も行われる．その際には「ラベルの置きかえ」も関連するので次にそれについて述べる．

原則として，β_r と G_α との対応関係がある点（すなわち，残留，分離または重ね合せとなる点）については，G_α 情報と β_r の情報とからその点に関するラベルが確定されなければならない．β_r と G_α との対応関係がない点（すなわち，追加される点）については，β_r の情報だけからその点に関するラベルが確定されなければならない．いずれの場合にも，ラベルが確定されないときにはエンベッディング失敗となる．当然ながら，エンベッディングにより除去される点については「ラベルの置きかえ」が起らない．以上のことをまとめると**表5**のようになる．辺ラベルに関しても同表に準ずる．

表 5 点ラベルとエンベッディング
Table 5 Embedding on node labels.

エンベッディング操作	β_r 上のその点のラベル			エンベッディング操作後の CIG 上のその点のラベル
点の残留または分離または重ね合せ	ラベル名	定数		同左
		*		G_α 上の対応するラベル名
	サフィックス	定数		同左
		*		G_α 上の対応するサフィックス
		ラベル変数	確定状態	同左の確定値
			未定状態	エンベッディング失敗
点の追加	ラベル名	定数		同左
		*		エンベッディング失敗
	サフィックス	定数		同左
		*		エンベッディング失敗
		ラベル変数	確定状態	同左の確定値
			未定状態	エンベッディング失敗

* $p>1$, $q>1$ のときは常にエンベッディング失敗である．

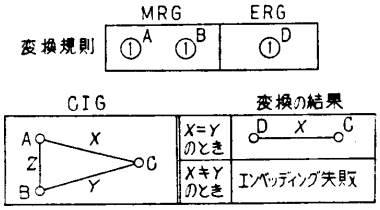

図 2 点の重ね合せ
Fig. 2 Superposition of nodes.

「点の分離」に関しては，「点の残留」と同様の処理でよい．ただし，分離される点のその周囲の点との接続関係は，もとの点のそれが複製化される．

「点の重ね合せ」に関しては，2点A,Bを重ね合せる場合，AB 間の辺は（もし存在すれば）エンベッディングにより除去される．また，2点 A,B が第3の点Cに対して辺をそれぞれ有する場合，2点 A,B を重ね合せることにより，それらの辺 AC および BC は一つの辺にされる．その際，それらの辺の辺ラベルが一致しない場合には変換後の辺ラベルが確定されないのでエンベッディング失敗となる（**図2**参照）．

4. 記述例

GML-56 によるグラフの変換の記述例として，与えられた連結無向グラフのブロックを認識する問題を考える．ここで簡単に用語の説明をする．連結グラフとは，任意の2点間に路が存在するグラフである．関節点 (cut point) とは，与えられた連結グラフをその点で切断すると非連結グラフになるような点をいう．ブロックとは，与えられた連結グラフをそのすべての関節点で切断したときに分離される各部分連結グラフのことをいう．たとえば，**図 3**(ⅰ)の連結グラフの関節点は a および b であり，同グラフのすべてのブロックは図 3(ⅱ)の B(1) ないし B(5) である．考えるべき問題は，与えられた連結グラフの各ブロックをそれぞれ適当な一つの辺に縮退させる（この辺をブロック辺とよぶ）ことで各ブロックを各ブロック辺に代表させ，結果としてブロックの総数に等しいブロック辺から成る連結グラフ（これをブロック辺グラフとよぶ）に変換することである．図 3(ⅲ)のブロック辺グラフは図 3(ⅰ)の連結グラフに対するものであり，5

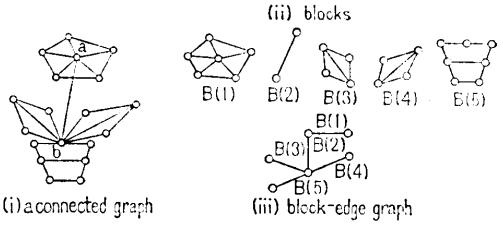

図 3 連結グラフとそのブロック辺グラフ
Fig. 3 A given connected graph and its block-edge graph.

ブロックから成ることが分る．ここで注意すべきは，ブロック辺グラフではブロック間の隣接関係が各ブロック辺の両端の任意の一方の点に連結させることで表わされるため，その表現が一意的に定まらないことである．しかし，木表現になることや辺の個数が与えられた連結グラフ内のブロックの総数に等しいことは常に成立している．

与えられた一つの連結無向グラフからそのブロック辺グラフを求める GML-56 プログラムのテキストおよび変換規則をそれぞれ**図 4**(ⅰ)*，図 4(ⅱ)に示す．テキスト上の L1 ないし L6 はラベル変数であり，変換規則の RG 内の &1 ないし &6 にそれぞれ対応している．本プログラムの入力グラフとしては，点ラベルおよび辺ラベルをもたない連結無向グラフを想定している．その出力グラフは，各ブロック辺にそのブロックが認識された順番を示す値が辺ラベルとして添えられるブロック辺グラフである．

本アルゴリズムの中心部分は規則6～12である．規則6または7により同定される辺（これを $\overline{AB}$ とする）の一端の点Aから出発して辺 $\overline{AB}$ を経由せずに点Bに到達する経路（すなわち辺 $\overline{AB}$ を通るサイクル）が存在するか否かを調べていく．このサイクルが存在しなければ辺 $\overline{AB}$ は一つのブロック（いわゆるブリッジ）であり，規則 10 で確定される．サイクルが存在する場合には，このサイクルは同一ブロック内にあるので辺 $\overline{AB}$ を通るサイクルについて縮退化を行う．縮退化とは，ブロックという性質を保存したまま，見出されたサイクルを点の重ね合せ（規則 12）の反復により一つの辺に縮める作業である．規則 11 はこの作業によりブロック数が誤って増加することを防ぐものである．L4(すなわち &4)は，与えられた入力グラフの点の総数の 10 倍を値とする定数であり，点の初期状態を表わすのに用いられる．なお，規則 2～5 はブリッジの判定や縮退化がグラフの形状のみから可能な場合に実施される部分であり，本質的には規則 6～12 があれば無しで済ませられるものであるが，効率の向上，RG の特殊記号の有効例，本問題の面白さの強調，等のために採用されている．

このプログラムの実行例を，中間結果としての CIG の系列によって**図 5** に示す．

* 理解の容易さを増すため，構造的 FORTRAN 言語 WESTRAN を用いて記述してある．

```
010 common /XCLVAL/L1,L2,L3,L4,L5,L6,LVAL(23)
020 integer XNODE,XSVAL,L1,L2,L3,L4,L5,L6
030 logical XMATCH,XRMATC,XEMBED,X,STOPSW,BRIDGE
040C ----- initialize -----
050 call XINIT(1)
060 L4=XNODE(X)*10
070 while(XRMATC(1)); X=XEMBED(1); endwhile
080 L3=0      ! L3 is #(blocks currently found) !
090C ----- iterate until all blocks are found --
100 doforever
110C    -- preprocess for efficiency --
120  doforever
130   while(XMATCH(2))
140    X=XEMBED(2);            call XBREAK(20).
150   endwhile
160   while(XRMATC(3))
170    L3=L3+1; X=XEMBED(3); call XBREAK(30)
180   endwhile
190   while(XMATCH(4))
200    L3=L3+1; X=XEMBED(4); call XBREAK(40)
210   endwhile
220   if(XMATCH(5));then
230    X=XEMBED(5);            call XBREAK(50)
240   else exit
250   endif
260  enddoforever
270C    -- find a starting node to traverse --
280  if(XMATCH(6));then
290   X=XEMBED(6);            call XBREAK(60)
300  elseif(XMATCH(7));then
310   X=XEMBED(7);            call XBREAK(70)
320  else exit             ! end of process !
330  endif
340C    -- traverse --
350  L1=0
360  repeat
370   L1=L1+1; L2=L1+1; STOPSW=.true.
380   while(XMATCH(8))
390    X=XEMBED(8); STOPSW=.false.
400   endwhile
410  until(STOPSW)

420C    -- find a bridge or a cycle --
430  L2=XSVAL(0)  ! set L2 undefined !
440  BRIDGE=.true.
450  while(XRMATC(9) .and. BRIDGE)
460   if(L2.NE.L4);then BRIDGE=.false.;endif
470   L2=XSVAL(0)
480  endwhile
490  if(BRIDGE);then   ! Bridge !
500   L3=L3+1
510   if(XMATCH(10));then
520    X=XEMBED(10);           call XBREAK(100)
530   endif
540  else              ! Found a cycle !
550   L5=XSVAL(0); L6=XSVAL(0)! to avoid a false
560   while(XRMATC(11))       !   degeneration !
570    if(L5.LT.L4 .and. L6.LT.L4);then
580     X=XEMBED(11);         call XBREAK(110)
590    endif
600    L5=XSVAL(0); L6=XSVAL(0)
610   endwhile
620   if(XMATCH(12));then  ! degeneration !
630    X=XEMBED(12);          call XBREAK(120)
640   endif
650  endif
660  L5=XSVAL(0)
670  while(XRMATC(13))     ! reset labels !
680   if(L5.LT.L4);then X=XEMBED(13);endif
690   L5=XSVAL(0)
700  endwhile
710 enddoforever
720C ----- finalize -----
730 while(XMATCH(14)); X=XEMBED(14); endwhile
740 call XTERM
750 stop
760 end
```

図 4(ⅰ) 例題プログラムのテキスト
Fig. 4(ⅰ) Text of an example program.

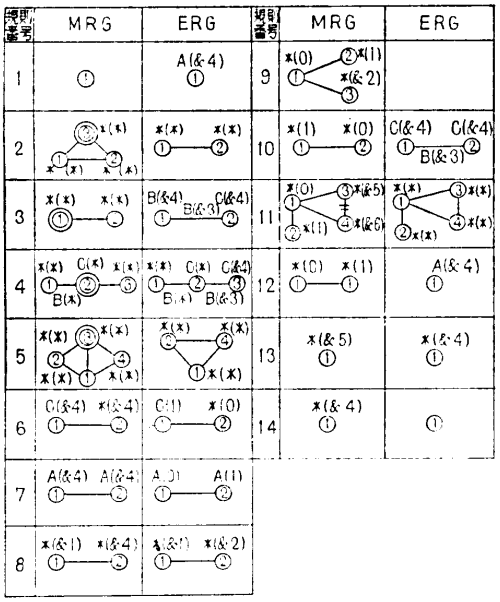

図 4(ii) 例題プログラムの変換規則
Fig. 4(ii) Transformation rules of an example program.

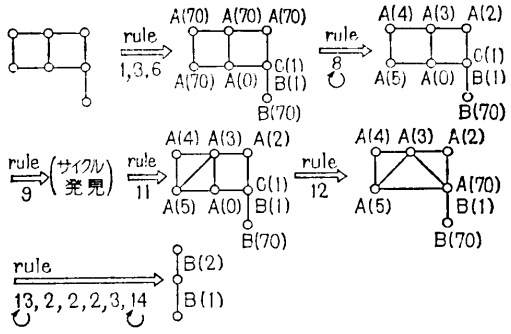

図 5 図4のプログラムの実行による CIG の変化
Fig. 5 Current input graphs during the execution of Fig. 4 program.

5. 評 価

GML-56 は実際にそのプログラミング・システム (GMS-56 とよぶ) が作成されている[4]ので，その使用経験も蓄積されてきている．本言語の最大特徴は，グラフの変換規則の外部表現に図形としてのグラフを，プログラムの制御構造の表現に FORTRAN のテキストを，それぞれ採用していることである．この方式の利点はテキストと変換規則とを独立に修正できることである．たとえば，変換規則の内容のみを（グラフ・エディタにより）修正する場合には，テキストを再コンパイルする必要なしに実行可能である．入力グラフを与えるたびにテキストを再コンパイルする必要がないことも大きな利点であるが，その反面，テキスト上からは入力グラフや出力グラフの形態が判断できないのでコメント等により補完されねばならない．このことは一般のテキスト形式によるグラフ処理言語のすべてにあてはまる問題であろう．本方式の欠点といえば，今も述べたように，テキストと変換規則（の外部表現）との両方を用意しないとプログラムの内容が理解できないことである．すなわち，プログラム・テキストのみからはその内容を完全には理解できないことである．

グラフの表現に関連する部分を一切テキスト上から排除してグラフ・エディタにそれを吸収させてあることの得失は，にわかに判断しがたい．そもそもグラフを表現するのに，そのデータ構造がテキストとして記述されているものよりも，図形として表示される方がユーザにとり理解しやすいのは明らかであり，要はそれほどの手間をかける（すなわちグラフ・エディタを導入する）だけの利点があるかということである．使用経験から言えば，グラフ・エディタ方式はユーザにとり気軽にシステムを利用できて想像以上に効果的であり，試みるだけの価値があろう．

6. おわりに

ユーザにとり直観的に理解しやすいグラフの図形的表現をそのまま変換規則の外部仕様として用い，プログラムの制御構造の表現に用いられているテキスト上にはグラフのデータ構造が一切登場しないというかなり大胆な方式は，外部表現と内部表現とを結ぶグラフ・エディタの存在があればこそ可能となるものである．

言語の外部仕様を徹底的にユーザ本位にしておき，そのためのしわ寄せを一手に引き受ける裏方（GML-56 の場合ではグラフ・エディタ）が外部仕様には一切浮上しないという本言語の試みは，各種の特殊目的言語にも適用されるべきと思われる．

謝辞 本研究の機会と激励を与えていただきました電子技術総合研究所石井治ソフトウェア部長に感謝します．また，本言語の構想段階で有益な助言をしていただいた山梨大学有沢誠氏に感謝します．

参考文献

1) たとえば Crespi-Reghizzi, S. Morpurgs, R.: A Language for Treating Graphs, Commun. ACM, 13-5, pp. 319-323 (1970).
2) 杉藤，真野，鳥居：グラフ処理用2次元言語GML

とその機能，電子通信学会論文誌D，J 59-D，9，pp. 597-604 (1976).
3) 真野，杉藤，鳥居：グラフ処理システム GMS とその応用，情報処理，18-3，pp. 257-264 (1977).
4) 杉藤，真野，鳥居，有沢：TSS 利用の会話型グラフ処理システムGMS-56，電子通信学会オートマトンと言語研資，AL 75-49 (1975).
5) 真野，杉藤，鳥居：Westran - Fortran をベースとした構造的言語とその処理系，情報処理，18-8，pp. 808-813 (1977).

(昭和53年10月26日受付)
(昭和54年 5 月17日採録)