GRAPHICS, UM SISTEMA DE REESCRITA SEQÜENCIAL DE FIGURAS

LILIA DE ASSUNÇÃO HESS

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho - D.Sc.
(Presidente)

Prof. Brian Henry Mayoh - Ph.D.
(Orientador)

Prof. Carlos José Pereira de Lucena - Ph.D.

Profa. Doris Ferraz de Aragon - D.Sc.

Profa. Ana Regina Cavalcanti da Rocha - D.Sc.

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 1989

HESS, Lilia de Assunção

Graphics, um Sistema de Reescrita Seqüencial de Figuras [Rio de Janeiro], 1989.

x, 90 p, 29,7 cm (COPPE/UFRJ, D.Sc. Engenharia de Sistemas e Computação, 1989).

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Gramáticas n-dimensionais, gramáticas de grafos I. COPPE/UFRJ II. Título (série).

Aos meus paemae e Nicolau.

Em homenagem a Dinorah pela sua força de oração e sabedoria de vida.

Em memória de meus avós e de Chiquinha.

É Deus quem guia os passos do homem, pois como poderia o homem compreender seu caminho? Provérbios 20.

AGRADECIMENTOS

Em todas as histórias que começam por: "era uma vez, em um país distante ...", depois que o personagem principal perdeu todas as esperanças, aparece no auge da crise um ente de poderes sobrenaturais, que transformará toda a história, viabilizando um final feliz. Assim também, na minha longa busca de um doutorado, Prof. Mayoh transformou tudo e me restituiu a confiança em mim mesma, permitindo-me vencer obstáculos intransponíveis. Agradeço sua orientação perfeita, segura e humana. Agradeço a sua esposa Grete, pois suas palavras ressaltando uma virtude de seu marido — a constância ao assumir seus compromissos com os alunos e o trabalho — constituiram minha única certeza, no transcorrer destes longos anos. Agradeço ao meu pai, pelo seu otimismo e crédito absoluto em mim, sempre. A minha mãe, pelo seu trabalho invisível de mãe e como revisora inigualável. A minha madrinha, por seu encorajamento constante. Aos professores componentes de minha banca de tese, Ana Regina, Doris, Lucena e em especial ao professor Maculan, por todo o apoio prestado. Ao professor Lucena devo ainda agradecer a orientação acadêmica informal,sobretudo nos anos de 1970 e 1983. Ao Maurício Kritz, pela sua orientação informal em 1983, e pela sua revisão e crítica ao segundo capítulo desta tese. Agradeço a Lúcia Kubrusly, por me conceder acesso ao correio eletrônico do LNCC, agora indispensável para cumprir os prazos. Ao Beauclair,pela implementação dos algorítmos. Ao Lopes, pelas ilustrações da tese. Ao Edgard Cândido,pelas transparências. Ao Paulo, as fotos. A Maria Antonieta Henriques de Pontes,

pelo excelente trabalho de datilografia. A Cláudia, a datilografia de última hora. Ao Luiz e a Néli, por todos os "finalmentes" dos volumes da tese. Às Irmãs Marcelinas, na pessoa da Irmã Dulce, e ao Sr. Nilo, a viabilização da Floresta da Tijuca como local de estudo. Aos professores Jayme Sczwarcfiter, Furtado e a todos os meus professores e amigos do IME, LNCC, INPA, PUC, UCLA e ESDI, pois seria impossível agradecer nominalmente a cada um.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.).

GRAPHICS, UM SISTEMA DE REESCRITA SEQÜENCIAL DE FIGURAS

LILIA DE ASSUNÇÃO HESS

Outubro de 1989

Orientador: Prof. Brian Henry Mayoh
Programa: Engenharia de Sistemas e Computação

Nesta tese será proposta uma nova entidade de representação n-dimensional, *graphics*, com sua gramática de reescrita. Este sistema formal de geração de figuras proporcionará aos modelos computacionais benefícios advindos de sua base interdisciplinar.

Graphics são grafos com atributos nos seus vértices. Seus atributos podem ser termos de uma Σ-álgebra, e suas gramáticas uma extensão da abordagem algébrica das gramáticas de grafos.

Graphics serão apresentados como modelos da realidade e solução para o problema de representação do conhecimento humano.

Aspectos de implementação e vários exemplos serão apresentados e discutidos.

A área das gramáticas de grafos, onde se insere o tópico desta tese, embora relativamente nova (originou-se em 1968 motivada por considerações sobre reconhecimento de padrões), tem sido um campo de pesquisa ativo e atuante nas mais diversas aplicações e seus resultados foram apresentados em três *workshops* : Bad Honnef, Osnabruck e Warrenton,respectivamente publicados em volumes dos *Lecture Notes In Computer Science* [1,2,3].

Algumas das aplicações motivadoras de *graphics* são em CAD/CAM; em descrição de figuras obtidas por aplicações médicas e por satélite; em arte e visão computacional.

Abstract of thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.).

GRAPHICS, A SEQUENTIAL REWRITING SYSTEM FOR FIGURES

LILIA DE ASSUNÇÃO HESS

OCTOBER 1989

Thesis Advisor: Prof. Brian Henry Mayoh

Department: Engenharia de Sistemas e Computação

It will be presented in this thesis a new n-dimensional representation entity — *graphics* and it's rewriting grammar.This formal rewriting system will furnish the computational models with benefits originated by it's interdisciplinary basis.

Graphics are graphs with attributes at their vertices. It's attributes can be Σ-algebra terms, and it's grammars an extension of the algebraic approach of graph grammars.

Graphics will be presented as reality models and as a solution to human knowledge representation.

Aspects of implementation programming and various examples will be presented and discussed.

The area of graph grammars,where the topic of this

thesis is included, is rather new (originated in 1968 and motivated by considerations concerning pattern recognition); nevertheless it has been an active research field,with many different applications and which results where presented in three workshops, at Bad Honnef, Osnabruck and Warrenton, respectively published in *Lecture Notes in Computer Science*, volumes [1,2,3].

Some motivating aplications to *graphics* are in CAD/CAM; medical and satellite picture description; in art and computer vision.

ÍNDICE

CAPITULO I

INTRODUÇÃO

Ao longo da história, o homem se exprimiu através de uma linguagem gráfica, na tentativa de expor seus planos e projetos para o grupo. A interação foi essencial no progresso dos resultados: modificando-se os gráficos, modificavam-se os planos.

Plantas de um modelo mental, os gráficos são usados hoje nos modelos computacionais.

Serão introduzidos nesta tese conceitos de representação n-dimensional. Será proposta uma nova representação n-dimensional que visa proporcionar descrições significativas, exatas e com grande abrangência de aplicações.

Simulando o processo de percepção, os gráficos serão tratados como modelos construtivos para interpretação de novos gráficos, através de gramáticas gráficas de reescrita em processos interativos.

As gramáticas gráficas de reescrita, ou gramáticas n-dimensionais, formam um ramo da área de gramáticas de grafos, termo genérico que se refere a uma variedade de métodos para especificação de conjuntos de grafos ou mapas [1], [2], [3]. Essa área que surgiu no final da década de 60 é eminentemente interdisciplinar, e seu primeiro artigo de PFALTZ e ROSENFELD [4] foi apresentado numa conferên

cia de inteligência artificial. Suas aplicações na área de ciência da computação incluem: reconhecimento de padrões, especificação de software, base de dados, biologia, animação e visão por computador, computação gráfica — representação de sólidos e fractais.

Nesta tese os capítulos estão distribuídos como segue: Capítulo I - introdução. Capítulo II-1 traz considerações sobre modelos de gramáticas de grafos, sem descer a detalhes de cada modelo (para um *survey* na área recomenda-se [5]), extraindo partes escolhidas e mantendo a notação dos autores. É já que a teoria das gramáticas de grafos é uma extensão da teoria das linguagens formais de *strings*, o Capítulo II-1 transmite a idéia de que os modelos de descrição de figuras (*webs, n-diagramas, grafos*) caracterizarão o tipo de gramática e determinarão uma hierarquia de Chomsky. Embora o poder do embutimento, no estabelecimento desta hierarquia, seja sobrepujado pela criação de identificadores de nós, que criarão uma dependência do contexto, mesmo com uma produção livre do contexto. Capítulo III trata das gramáticas n-dimensionais com coordenadas e das gramáticas com atributos que são os modelos de gráficos propriamente ditos. Capítulo IV apresenta o modelo *graphics* com a definição de *graphics*, seus morfismos e sistema de reescrita, discutindo suas aplicações e implementações. Capítulo V - conclusões.

CAPÍTULO II

GRAMÁTICAS DE REESCRITA N-DIMENSIONAIS

As gramáticas de reescrita n-dimensionais tratadas foram as gramáticas de grafos seqüenciais e as gramáticas gráficas de reescrita, com coordenadas.

Embora as características das gramáticas seqüenciais residam em seus modelos de descrição da figura, geralmente o modelo conceitual grafo engloba os conceitos de *Webs*, *n-diagramas*, *grafos coloridos* e significa modelo n-dimensional.

As gramáticas de reescrita n-dimensionais seqüenciais seguem a hierarquia de Chomsky, mas quando o embutimento é irrestrito estas classes colapsam, a despeito da complexidade dos lados esquerdo e direito da produção.

II.1 - GRAMÁTICA DE GRAFOS

Os sistemas generativos, isto é, gramáticas de reescrita ou gramáticas n-dimensionais, são sistemas formais, que geram ou analisam conjuntos de entidades n-dimensionais chamadas livremente de figuras, devido a suas primeiras aplicações. Uma entidade n-dimensional, onde n tem valor unitário é um *string*. Este aspecto conduz à expansão da teoria de linguagens formais de *strings*.

Atualmente há uma variedade de modelos de descrição de figuras. Dependendo deles as gramáticas ficam caracte

rizadas como gramáticas de *webs*, *grafos coloridos*, *n-diagramas*, *maps*, etc. Detalhes destas descrições são postos de lado, ao serem todas englobadas no conceito de grafo.

Os mecanismos de reescrita podem ser seqüenciais ou paralelos. Quando a reescrita é paralela, várias mudanças são feitas ao mesmo tempo em um passo de derivação. Isto é, a reescrita é executada de forma não localizada, e tem sua aplicação principal em biologia.

A reescrita seqüencial é destes mecanismos o que será adiante enfatizado, por ser também o mecanismo da abordagem algébrica empregada no formalismo desta tese. Neste capítulo, a reescrita seqüencial é vista na abordagem de teoria dos conjuntos para gramáticas de grafos, também denominada abordagem algorítmica.

Uma gramática de reescrita de grafo, ou gramática de grafo, é definida geralmente por uma ênupla, na qual tem-se:

a) o conjunto do alfabeto (símbolos terminais e não terminais);

b) o axioma (descrição de um grafo inicial) e

c) o conjunto das produções.

Uma produção é definida por outra ênupla, que por sua vez, especifica:

c.1) dois grafos que constituem seus lados esquerdo e direito;

c.2) as funções de embutimento; e

c.3) condições de aplicabilidade desta produção.

A derivação, uma transformação de grafos, é obtida como nas gramáticas uni-dimensionais. Procede-se a uma derivação substituindo no grafo *hospedeiro* a ocorrência de um subgrafo, isomorfo ao grafo esquerdo de uma regra,por grafo isomorfo ao grafo do lado direito desta produção. Mais livremente, os autores se referem à substituição no hospedeiro "do grafo esquerdo da produção pelo grafo direito". Todas derivações são feitas a partir de algum axioma inicialmente especificado.

Chama-se de *contexto* ao resultado da extração no hospedeiro do subgrafo correspondente ao grafo do lado esquerdo da produção, e de *embutimento* (*embedding*) ao modo como são realizadas as ligações entre o contexto e uma imagem equivalente ao grafo do lado direito da produção.

A comparação entre modelos de descrição de figuras, embora generalizados pelo conceito de grafo, pode ser estabelecida a partir da própria representação escolhida. Entre os modelos de descrição de figuras estão os *webs*, grafos não direcionados com nós rotulados mas sem rótulos nas arestas. Este capítulo trata das gramáticas de *webs* propostas tanto por MONTANARI [6] como por ROSENFELD e MILGRAM [7]. Nestes autores, um *web* é formalmente dado pela ênupla (N_W, F_W, A_W), onde N_W é o conjunto de vértices, F_W é uma função de rotulação e A_W é o conjunto de arcos, que são pares ordenados de vértices.

Através de um artigo discursivo e informal, Montanari determina o embutimento de uma produção utilizando-se de uma função normal. A função de embutimento estabelece as conexões do grafo contexto, — cujas rupturas se deram ao ser retirado o subgrafo correspondente ao lado esquerdo da produção —, com o grafo do lado direito da produção.

Uma função do conjunto de vértices do *web* esquerdo, α, de uma produção, no conjunto de vértices do *web* direito, β, se chama uma função normal quando for injetiva; estabelecendo então quais vértices são correspondentes aos retirados no reestabelecimento das conexões com o contexto.

Visto Montanari determinar o embutimento com uma função normal, suas gramáticas são monotônicas, isto é, $|\alpha| \leqslant |\beta|$. Na produção $R = (\alpha, C, \beta, E)$, interpreta-se α e β como *webs*, C como uma função lógica relativa ao contexto da derivação e E como um conjunto de funções lógicas de embutimento. Em seu texto, encontra-se o seguinte exemplo de produção:

$R = \{$ $\overset{t}{\underset{L_1}{\cdot}}$,

em W não pode existir mais do que um arco de L_1,

$\overset{t}{\underset{R_1}{\text{———}}}\ \overset{t}{\underset{R_2}{}}$,

todos os vértices de W que forem conectados com L_1 são conectados com R_1 e nenhum vértice é conectado com R_2 }.

A leitura deste exemplo segue o formato da ênupla R, isto é, o primeiro elemento da quádrupla R é o *web* α com rótulo L_1 e vértice de índice t; o segundo elemento é a condição de aplicabilidade da regra no *host* W; o terceiro elemento é o *web* β com rótulos R_1 e R_2 e vértices de índice t; e finalmente o quarto elemento é a função de embutimento.

Pode-se ver, em ROSENFELD e MILGRAM [7], como a função de embutimento relaciona conjuntos de nós apenas pelos seus rótulos. Rosenfeld e Milgram usam a função de embutimento $\varphi: N_\beta \times N_\alpha \rightarrow 2^V$, — onde N_α é o conjunto de nós do *web* esquerdo da produção, N_β o conjunto dos nós do *web* direito e V um conjunto de rótulos, relacionados aos vértices de um *web* através de uma função de rotulação, — para estabelecer que em qualquer que seja $m \in N_\alpha$, todos os rótulos de vizinhos de m em $w-\alpha$, contexto da derivação, estão em φ (n, m) para algum $n \in N_\beta$.

Embora a especificação do embutimento, no exemplo geral de MONTANARI [6], empregue índices de nós para especificar o embutimento, e se refira a um i-ésimo vértice do lado esquerdo da regra na especificação de elementos conectados da função de embutimento; e embora Montanari tenha também usado uma correspondência geométrica entre os vértices, para especificar o embutimento normal; — conclui-se, quando a descrição de figuras for dada por um *web*, que a especificação do embutimento nas gramáticas depende principalmente do uso dos rótulos, para assim se redefinirem ligações.

Quando esta descrição de figuras for dada por um

n-diagrama, as gramáticas irão depender também do próprio nó para caracterizá-lo na derivação. Um n-diagrama é um grafo direcionado e rotulado tanto nas arestas como nos nós, representado pela ênupla $G = (K, \rho_1, \ldots, \rho_n, \beta)$ onde K é o conjunto de nós, $\rho_i \subseteq K \times K$ são subconjuntos das arestas com rótulo i, $\beta : K \to V$ é a função de rotulação dos vértices.

As gramáticas n-dimensionais, que usam os n-diagramas como descritores, aqui abordadas foram as de SCHNEIDER [8] e NAGL [9]. Nestas gramáticas, o embutimento é resultado de uma composição de arestas. Como estas são direcionadas, é necessário especificar suas conexões de entrada e saída.

As relações Π_i e Π_{-i} são relações entre os conjuntos de nós dos n-diagramas dos lados esquerdo e direito de uma regra, e encontradas na função de embutimento de Schneider. Esta função determina a composição de um conjunto de arestas, com os conjuntos de arestas de entrada ou de saída, integrantes da "conexão entre o n-diagrama esquerdo e o contexto".

Pode-se observar que estas relações tratam com o conjunto de nós K e assim o embutimento se torna dependente também da especificação do próprio nó.

Já na gramática de n-diagramas de Nagl existe uma composição entre as entidades nós e rótulos. Conforme exemplo da interpretação do conjunto de arestas ℓ_i de entrada em [5], que formará com outras arestas a conexão do n-diagrama a ser embutido com o contexto, uma aresta de entrada será ro

tulada $\ell_i = (t_3\ L_\ell\ R_j(3);\ 7,8)$ onde se lê: partindo do nó 3 do lado esquerdo, siga uma aresta de saída de rótulo j, e então siga uma aresta de entrada com rótulo ℓ e tome apenas os nós do resultado cujo rótulo é t_3. Estes nós obtidos serão nós fontes para arestas rotuladas i de entrada, que terminam em 7 e 8, nós do subdiagrama inserido e isomorfo ao lado direito da regra.

Esta caracterização do nó, que também diferencia nós de mesmo rótulo, é encontrada nas gramáticas de abordagem algébrica. Tomando-se este fato por base, pode-se afirmar que a descrição de *grafo colorido*, usada na abordagem algébrica, é equivalente à de n-diagrama.

Concluindo, o uso do próprio nó nas especificações, além do seu rótulo, caracteriza o seu grau e distingue nós de rótulos iguais, permitindo operações para unir vértices, criar ou contrair arestas. Além disto, a entidade que descreve e representa a figura caracteriza a gramática, conforme vêm enfatizar as gramáticas gráficas de reescrita.

Para dar continuidade à discussão da comparação entre os modelos de descrição de figuras e suas gramáticas, é preciso observar a complexidade crescente da função do embutimento nestes modelos. É esta complexidade crescente que influe na classificação da gramática dentro da hierarquia de Chomsky.

Em MONTANARI [6], um arco entre o "*web* esquerdo", α, e o contexto, w-α, nunca é criado ou apagado, mas apenas transferido para o "*web* direito", β, com a função do embuti-

mento sendo uma função normal na maioria dos exemplos.

Em ROSENFERD e MILGRAM [7], um nó pode ter mais de uma imagem, desde que seu rótulo se encontre em mais de um dos conjuntos de rótulos de φ (n, m), dados pela função de embutimento.

Em SCHNEIDER [8], o mecanismo de embutimento permite que arestas de entrada e saída possam ser multiplicadas,ou combinadas em apenas uma.

Em NAGL [9], as expressões de embutimento com seus operadores tornam o embutimento irrestrito, a despeito da complexidade do lado esquerdo e direito da produção, e assim as classes de Chomsky colapsam. Por exemplo, por causa do embutimento o resultado de uma substituição livre de contexto pode depender do contexto do diagrama do lado esquerdo da regra, através da geração de arestas dependentes do rótulo destes nós na expressão de embutimento.

Verifica-se, com algumas variações, os resultados obtidos em linguagens formais de seqüência de símbolos uni-dimensionais, *strings:*

ling. regulares $\subset$ ling. livres de contexto $\subset$ $\subset$ ling. sensíveis ao contexto $\subset$ ling. monotônicas $\subset$ ling. recursivamente enumeráveis.

Entretanto, devido à transformação irrestrita do embutimento, as classes colapsam, NAGL [9].

ling. regulares $\subset$ ling. livres de contexto normal = = ling. livres de contexto = ling. sensíveis ao contexto = = ling. monotônicas.

Assim, como resultado da comparação entre os modelos de descrição de figuras e da complexidade do embutimento obtém-se:

ling. Montanari $\subset$ ling. Rosenfeld e Milgram $\subseteq$ ling. Schneider $\subset$ ling. Nagl.

II.2 - Introdução à Teoria Algébrica de Gramática de Grafos

Sistemas que visam validade representacional automática são desenvolvidos através de modelos que utilizam especificação algébrica.

Destes modelos conceituais avançados, alguns em vez de codificarem figuras como seqüências de símbolos descrevem-nas através de representações e gramáticas n-dimensionais.

Por sua vez, as gramáticas n-dimensionais podem se beneficiar de uma abordagem algébrica.

II.2.1 - Conceitos Algébricos, Um Referencial

A visão global e informal de alguns conceitos básicos, dada a seguir, se propõe a criar um referencial para a leitura e introduzir o estudo formal da teoria de modelagem conceitual através da abordagem algébrica, propiciando uma base para a compreensão das precisas construções das gramáti

cas de grafo que empregam a especificação algébrica.

Embora as técnicas de especificação algébrica sejam mais conhecidas através da fundamentação de tipos abstratos de dados e o conceito de abstrato, enunciado a seguir, tenha sido retirado de GOGUEN e THATCHER [10], não se deve restringir a especificação algébrica ao âmbito desta construção.

Abstrato significa independente de representação. Significa a noção de sintaxe abstrata, independente de implementação. Uma entidade se caracteriza abstratamente através de estrutura e objeto inicial, envolvendo conceitos de:

Signature
Especificação
Álgebra
Categoria

Uma álgebra é uma semântica dos conceitos sintáticos de *signature* e especificação.

Sendo tipos especiais de álgebras, a álgebra inicial e a álgebra dos termos conduzem à idéia de dar à sintaxe um significado de representação inicial, — de uma álgebra gerada por suas operações —, o que caracterizaria a classe de isomorfismos de um objeto e portanto caracterizaria este objeto abstratamente.

Uma *signature* consiste de conjuntos de:

Sorts
Símbolos de operações

Uma *Σ-álgebra*, ou álgebra universal, é formada pelas famílias de:

conjuntos (carriers, conjuntos base, ou universos de *sort)*

funções

Por sua vez, um ***sort*** é um conjunto de nomes de domínios disjuntos, sendo portanto um conceito sintático.

Igualmente sintático é o conceito dos ***símbolos de operações***, que são o conjunto de funções cujos elementos do domínio são seqüências finitas de *sorts* do conjunto de *sorts* dado, e contra-domínio um elemento do conjunto de *sorts*. Conhecidas como ***declarações***, são apenas funções de nomes em nome. Dentre os símbolos de operações, as ***constantes*** são símbolos especialmente definidos, onde o domínio é uma seqüência vazia de *sorts*; são conhecidas também como símbolos de operações de aridade zero. O módulo destas seqüências finitas de sorts, domínio de declarações, é conhecido como *aridade*.

Uma ***especificação*** é composta de uma *signature* e um conjunto de equações.

Uma ***família de conjuntos,*** ou ***carriers,*** é constituída dos universos associados a cada um dos elementos do conjunto de *sorts*. Assim, cada elemento do conjunto de sorts dá um nome para um respectivo conjunto base, sendo portanto um conceito semântico.

Igualmente semântico é o conceito de uma *família*

de funções, que são as operações da álgebra, portanto denotadas pelos símbolos de operações, de acordo com as declarações, isto é, domínio e contradomínio dos símbolos de operações, que designam a aridade da operação interpretada, e também os conjuntos base envolvidos. Dentre estas funções, as *constantes da álgebra*, denotadas pelos símbolos constantes, são elementos pertencentes ao conjunto base correspondente ao contra-domínio. Neste caso, existe uma correspondência direta entre elementos (constantes) e funções (operações da álgebra), já que a aridade das declarações de constantes é zero.

Quanto à concepção de álgebra dos termos, esta depende, além do conceito de *signature*, do conceito de um conjunto de variáveis de *sort*, distintas entre si e diferentes dos símbolos de operações. Na *álgebra dos termos* os conjuntos base são os conjuntos de termos,—isto é, constantes e termos gerados a partir das constantes, obtidos das expressões ou fórmulas e resultantes da aplicação recursiva das operações nelas encontradas, de acordo com as declarações dos seus respectivos símbolos de operações —, e as operações são a família de funções das próprias operações construtoras dos termos.

Uma *categoria* consiste de uma classe de objetos, de um K-morfismo para cada par ordenado destes objetos (uma abstração do conceito de função), e de uma composição dos K-morfismos, para cada tripla de objetos (uma função no sentido de teoria dos conjuntos), com as propriedades de associatividade e identidade. Numa categoria de Σ-álgebras os objetos são classes de Σ-álgebras e os K-morfismos são Σ-homomorfis-

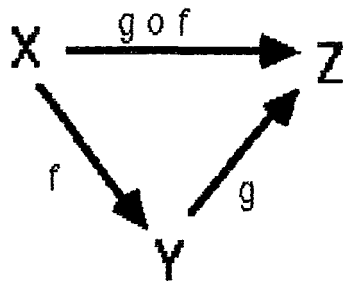
mos (que preservam as operações das álgebras) e a composição dos K-morfismos é também um Σ-homomorfismo.

O estudo formal desses conceitos deve incluir os textos de HUET e OPEN [11], EHRIG e MAHR [12].

II.2.2 - Introdução à Categoria dos Grafos

A abordagem algébrica das gramáticas de grafos permite a utilização de técnicas e regras da teoria de categorias nas construções e provas; dentre estas técnicas, as dos diagramas universais. Provas de teoremas podem ser reduzidas ao desenho de um diagrama de funções e à seleção de caminhos que verifiquem a comutatividade.

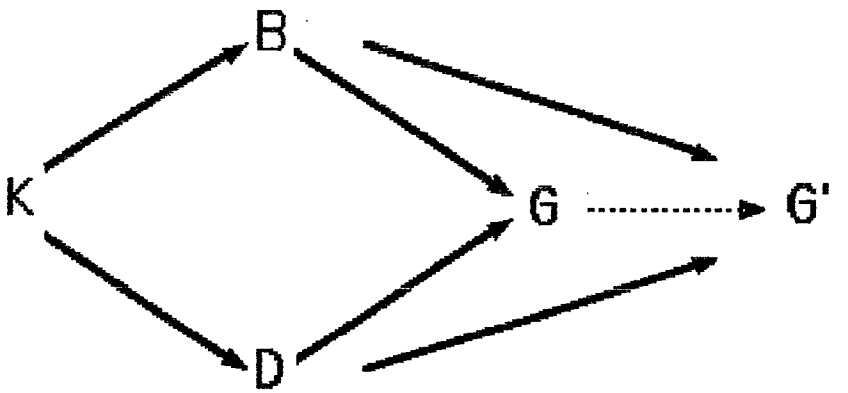
Dando continuidade à introdução de conceitos básicos, o diagrama $X \xrightarrow{f} Y \xrightarrow{g} Z$ denota duas funções, cuja composição g.f e as composições $f.id_X = f = id_Y.f$, se id_X e id_Y forem respectivamente as funções de identidade de X e de Y, tornam o *diagrama comutativo*, isto é, não importa o caminho seguido, (basta acompanhar a direção das setas e aplicar a função designada a cada transição, que o resultado geral se mantém).



Dados morfismos K → B e K → D, um *push-out* destes morfismos consiste em um objeto G e dois morfismos B → G e D → G, de forma que:

1. (comutividade): (K → B → G) = (K → D → G), portanto, o diagrama é comutativo e satisfaz a

2. (propriedade universal): para todos objetos G' e morfismos B → G' e D → G' satisfazendo (K → B → G') = = (K → D → G') existe um morfismo único G → G' tal que (B → G → G') = (B → G') e (D → G → G') = (D → G') como ilustrado pelo diagrama:



Das definições formais, deve se extrair essencialmente que, nas gramáticas de grafos por abordagem algébrica, os grafos componentes das produções são combinados através de diagramas comutativos e a aplicação das produções se dá através de morfismos em grafos. Grafos e seus morfismos definem uma categoria, chamada de categoria dos grafos. Os *diagramas de colagem*, mecanismo utilizado na aplicação das produções, são *push-outs* na categoria de grafos; com todas as vantagens teóricas advindas deste fato.

Seguem as definições de grafos (objetos), seus morfismos (K-morfismos) e composição destes.

Partindo de um alfabeto de cor, para arcos e nós respectivamente, C (C_A, C_N) um par de conjuntos, que serão fixos, define-se um *grafo (colorido)* $G = (G_A, G_N, s, t, m_A, m_N)$, consistindo dos conjuntos

G_A de arcos

G_N de nós,

e das funções seguintes, que indicam:

$s: G_A \to G_N$ as fontes,

$t: G_A \to G_N$ os sumidouros,

$m_A: G_A \to C_A$ a coloração de arcos,

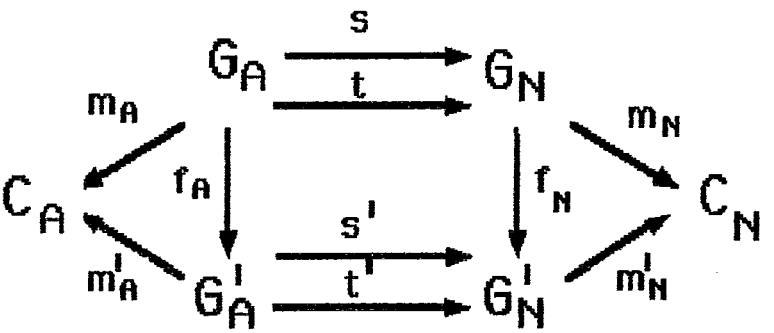
$m_N: G_N \to C_N$ a coloração de nós.

que são denotadas pelo diagrama

$G: C_A \xleftarrow{m_A} G_A \underset{t}{\overset{s}{\rightrightarrows}} G_N \xrightarrow{m_N} C_N$. Dizemos que G' é um subgrafo de G se $G' \subseteq G_A$, $G'_N \subseteq G_N$ e todos s', t', m'_A e m'_N são restrições dos mapeamentos de G correspondentes. Note que esta definição de grafos permite arcos paralelos, inclusive aqueles com mesma cor. Não distinguir entre arcos e nós, isto é, $x \in G$ significando que $x \in G_A$ e $x \in G_N$, equivale ao conceito de item.

Dados dois grafos G e G' um morfismo de grafo f: $G \to G'$, denotado por f ou $G \to G'$, é um par de mapeamen-

tos $f = (f_A: G_A \to G'_A, f_N: G_N \to G'_N)$ tal que $f_N\, s = s'f_A$, $f_N\, t = t'f_A$, $m'_A\, f_A = m_A$ e $m'_N\, f_N = m_N$, isto é, o seguinte diagrama comuta para os mapeamentos de fonte e sumidouro separadamente



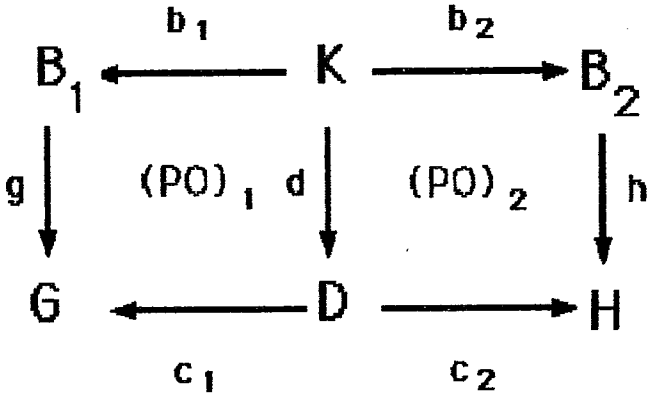
Um morfismo de grafo pode ser injetivo ou sobrejetivo respectivamente, se ambos f_A e f_N são mapeamentos injetivos ou sobrejetivos. E será um isomorfismo, se for injetivo e ao mesmo tempo sobrejetivo, existindo então um isomorfismo inverso.

A *composição* $f'f: G \to G''$ de *dois morfismos de grafos* $f = (f_A, f_N): G \to G'$ e $f' = (f'_A, f'_N): G' \to G''$ é definida por $f'f = (f'_A f_A, f'_N f_N)$.

Finalmente, as definições que compõem o cerne das gramáticas de grafos por abordagem algébrica: as definições de produção, derivações diretas, e construção de colagem.

Uma *produção* de grafo, denotada por p, é um par de morfismos de grafos, $p = (B_1 \leftarrow K \to B_2)$, onde grafos B_1, B_2 e K são chamados respectivamente lado esquerdo, lado direito e interface (designada por pontos de colagem, *gluing points*) de p. Uma produção é dita rápida se $K \to B_1$ e $K \to B_2$ são injetoras.

Dada uma produção p = ($B_1 \leftarrow K \rightarrow B_2$) e um grafo D, chamado contexto, juntamente com um morfismo de grafo K → D uma *derivação direta* consiste dos seguintes *push-out* $(PO)_4$ e $(PO)_2$, chamados de diagramas de colagem.



A derivação direta é denotada por $G \overset{p}{\Rightarrow} H$ (ou p: G ⇒ H), onde G é o grafo hospedeiro, a derivação pode ser compreendida como nas outras abordagens, com ligeira diferença quanto aos pontos de colagem, que por não serem retirados são depois fundidos na substituição, com os pontos de colagem correspondentes do grafo do lado direito. A aplicação da produção é dada pelo morfismo g: B_1 → G, chamado de *ocorrência* de p em G. Respectivamente, o morfismo h: B_2 → H é a ocorrência de p em H. Podendo ser assim compreendidos como mecanismos de embutimento. Todas as condições de colagem são obtidas diretamente através do diagrama de colagem e de suas propriedades, advindas da sua condição de push-out. Note também que se g é injetiva, não existirão nós aglutinados em um só, g será uma inclusão, isto é, B_1 será subgrafo de G, onde respectivamente, os conjuntos de vértices e arestas de B são subconjuntos e os mapeamentos restrições, dos correspondentes em G.

Gramática de Grafos (seqüencial) pode ser definida

como a ênupla GG = (C, T, prod, start), consistindo de um par de alfabetos de cor $C = (C_A, C_N)$ (podendo incluir em C um alfabeto de cor terminal $T = (T_A, T_N)$),um conjunto finito de produções e um grafo inicial finito. A *linguagem de grafo* L(GG) é definida como conjunto de todos os grafos (terminais) coloridos derivados do grafo inicial. Note que como a construção de push-out é somente única, a menos de isomorfismo, para cada derivação direta $G \Rightarrow H$ também $G \Rightarrow H'$, donde para cada $H \in L(GG)$ também $H' \in L(GG)$ para todos os grafos isomorfos a H. Assim podemos considerar L(GG) como uma linguagem de grafos abstratos.

Para se enunciar a propriedade de Church-Rosser, duas definições são necessárias: as derivações independentes paralelas e as derivações independentes seqüenciais.

Dada duas produções rápidas

$$P = (B_1 \xleftarrow{b_1} K \xrightarrow{b_2} B_2) \quad e \quad p' = (B_1' \xleftarrow{b_1'} K' \xrightarrow{b_2'} B_2'),$$

duas derivações diretas, $G \Rightarrow H$ via p baseada em g e $G \Rightarrow H'$ via p' baseada em g', são chamadas *independentes paralelas*, se a interseção de B_1 e B_1' (ocorrência de p e p' em G) consistir de itens comuns de colagem. O que significa precisamente

$$1.\ g\,B_1 \cap g'\,B_1' \subseteq g\,b_1\,K \cap g'\,b_1\,K'$$

de forma dual, a seqüência de derivações $G \Rightarrow H \Rightarrow X$ via (p,p') baseada em g e g' é chamada seqüência de *derivações indepen-*

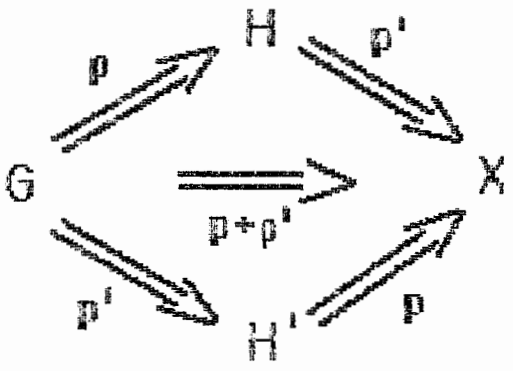
dentes seqüenciais.

$$2.\ h\ B_2 \cap g'\ B'_1 \subseteq h\ b_2\ K \cap g'\ b'_1\ K'$$

Destes conceitos desenvolve-se o conceito de *produções paralelas,*

$$p + p' = (B_1 + B'_1 \xleftarrow{b_1+b'_1} K + K' \xrightarrow{b_2+b'_2} B_2 + B'_2)$$

onde + denota a união disjunta de grafos e de morfismos de grafos, respectivamente. Note que p + p' se torna também uma produção rápida e que uma derivação G ⇒ X, via p + p', é chamada de derivação paralela. Segundo o *teorema de Church-Rosser*, sejam p e p' produções rápidas e G ⇒ H e G ⇒ H' derivações diretas independentes paralelas via p e p' respectivamente, então existem um grafo X e derivações diretas H ⇒ ⇒ X via p' e H' ⇒ X via p tal que $G \overset{p}{\Rightarrow} H \overset{p'}{\Rightarrow} X$ e $G \overset{p'}{\Rightarrow} H' \overset{p}{\Rightarrow}$ ⇒ X são seqüencialmente independentes.



E vice-versa, dada a seqüência de derivações independentes seqüenciais $G \overset{p}{\Rightarrow} H \overset{p'}{\Rightarrow} X$ (respectivamente $G \overset{p'}{\Rightarrow}$ ⇒ $H' \overset{p}{\Rightarrow} X$) tem-se também a seqüência de derivações independentes seqüenciais $G \overset{p'}{\Rightarrow} H' \overset{p}{\Rightarrow} X$ (respectivamente $G \overset{p}{\Rightarrow} H \overset{p'}{\Rightarrow}$ ⇒ X) tal que $G \overset{p}{\Rightarrow} H$ e $G' \overset{p'}{\Rightarrow} H'$ se tornam derivações independentes paralelas. Além disso, em todos os casos citados,

tem-se uma produção paralela $p + p'$, e uma derivação $G \Rightarrow X$ via $p + p'$, onde $B_i + B_i'$ é a união disjunta dos grafos B_i e B_i' para $i = 1,2$.

Outras restrições podem ser feitas às produções, resultando nas construções de concorrência e amalgamação, que podem ser estudadas detalhadamente em EHRIG [13], [14] e [15].

Como conclusão: o valor da abordagem algébrica, nas gramáticas de reescrita n-dimensionais, está na sua operacionalidade.

A abordagem algébrica das gramáticas de reescrita de grafos tem seu aspecto mais importante no resultado operacional, não devendo ficar a sua compreensão restrita à classificação hierárquica ou à determinação do grau de complexidade do embutimento destas gramáticas. Pois, para que um subgrafo seja substituído, não se faz necessário encontrar todas as arestas na regra e o embutimento tem em $K \rightarrow B_1$ e $K \rightarrow B_2$ morfismos injetores e que preservam cor, o que torna as gramáticas, quanto à hierarquia, elementares com uma monotonia adicional relativa às arestas de embutimento NAGL [5].

Esta abordagem trata o conjunto de produções como um sistema de reescrita confluente, estabelecendo uma seqüência de produções não arbitrária.

As propriedades de Church-Rosser para sistema de reescrita de termos são estendidas num teorema de Church-Rosser para um sistema de reescrita de grafos, estabelecendo meios sistemáticos de reduzir numa derivação o que seria uma cadeia de derivações, através de produções simplificadas.

CAPÍTULO III

MODELOS DE GRÁFICOS

Quando modelos formais tratam entidades pictóricas, proporcionando aos modelos computacionais a validade na interpretação gráfica, surge um novo conceito de modelagem e reconstrução da forma e movimento, com aplicações ilimitadas em várias áreas, particularmente na computação gráfica,visão por computador, engenharia de software, inteligência artificial e robótica.

III.1 - GRAMÁTICA DE FORMAS

Os primórdios dos métodos lingüísticos para problemas envolvendo processamento de figuras foi levantado por Miller e Shaw em 1968 [19]. Como os modelos computacionais para informações gráficas despertavam interesse e eram inúmeros, já em 1972 foi realizada uma conferência em Vancouver, sobre linguagens gráficas [20].

Neste capítulo, visando uma introdução comparativa, serão apresentadas algumas das gramáticas de "picture languages", das gramáticas conhecidas como "plex-grammars" e das gramáticas com coordendas, — de modo classificatório, por serem muito diversas as abordagens para a descrição e reconstrução de figuras. Estes modelos de descrição podem utilizar desde "arrays" numéricos de coordenadas até estruturas n-dimensionais, como grafos ou grafos com atributos; mas geralmente envolvem processos de segmentação hierárquica das figuras e uma decomposição final em pontos e retas.

Os modelos que geram as "picture languages" [21], aplicadas em reconhecimento de padrões, empregam códigos de cadeias, que descrevem uma figura digital através de pontos e sua vizinhança, de oito ou quatro vizinhos. E estabelecem uma relação de equivalência: a conexão de pontos. A complexidade das "picture languages" é estabelecida de acordo com a hierarquia de Chomsky; sendo propostas para o reconhecimento máquinas formais, como os autômatos celulares. A resolução por algoritmo do problema da equivalência para as "picture languages" regulares e o problema de pertinência das "picture languages", livres de contexto, na classe dos problemas NP completos, encontra-se em KIM e SUDBOROUGH [22].

Ainda um modelo de "picture languages", num trabalho com aplicações de valor artístico, KAMALA e ANINDYA [23] criam gramáticas para gerar os padrões de Kolam, que tradicionalmente são usados na decoração dos pisos das casas, no sul da Índia. As deformações no parquê são descritas usando gramáticas de "arrays" e matrizes, onde as derivações são definidas através de produções horizontais e verticais; onde os primitivos gerados (terminais) são substituidos por funções que modificam estruturas gráficas básicas, como círculos, hexágonos, ângulos, pontos e retas. Estas funções podem ser descritas em termos de outras funções, de acordo com regiões. Assim um terminal gerado, em qualquer passo da derivação, é definido como o valor de uma função no tempo. Estes terminais podem ser ênuplas ordenadas, numa extensão do conceito de atributos.

Entretanto, a característica nas descrições das

"picture languages" é a codificação de elementos gráficos em variáveis literais, ou em seqüência de variáveis literais, dispostas em "arrays" ou matrizes.

Assim, ainda incluido neste conceito sintático, e aplicado no reconhecimento de figuras parcialmente distorcidas, através de técnicas de correção de erros, o trabalho de YOU e FU [24] envolve manipulação simbólica e numérica. Para cada decomposição em subforma, é construída uma produção descrevendo a relação de concatenação. As subformas são decompostas até serem todas simples segmentos de curva, por sua vez então associados a um primitivo de curva e descritos por um conjunto de atributos.

As "plex-grammars" foram desenvolvidas por Narasimhan,Shaw,Feder, ver [19,20]. Numa extensão da teoria de Feder, os pesquisadores LIN e FU [25] propuseram a "3-D plex grammar", cuja idéia básica é considerar cada símbolo, terminal ou não terminal, como uma superfície primitiva ou composta, tendo um número n de curvas de conexão para a ligação a outras superfícies. Esta estrutura é chamada de NACE (n-attaching-curve entity) e as estruturas formadas pela interconexão destas entidades são chamadas de *plex*. Todas as curvas de conexão de uma NACE têm um identificador, e um identificador nulo para ser usado como marcador de posição das entradas dos campos de conexão, caso uma junta não estiver envolvida.

Uma produção irrestrita é da forma

$$\psi\ \Gamma_{\psi}\ \Delta_{\psi} \rightarrow \omega\ \Gamma_{\omega}\ \Delta_{\omega}$$

onde ψ e ω são chamadas as listas de componentes do lado esquerdo e direito, respectivamente, Γ as listas de interseção e Δ as listas de curvas de conexão, considerando-se os subíndices que designam esquerda e direita.

Por sua vez, as listas de componentes são seqüênciais de NACES unitários (componentes),e fornecem a ordem dos NACES conectados. A ligação das curvas de conexão de dois ou mais NACES formam as interseções e Γ especifica o modo como as listas de componentes se interligam. Estas listas de interseção são não ordenadas e divididas em campos, que especificam quais curvas de conexão, de qual NACE, são conectadas em cada interseção, onde um campo corresponde a uma interseção.

Segundo LIN e FU [25],as plex-grammars são um caso especial de gramáticas de atributos. Para aplicação prática foram usadas as "3D-plex grammars" livre de contexto; classificação que corresponde à de "plex grammars". Como aplicação atual de plex-grammars, WOODMAN, INCE, PREECE e DAVIES [26] desenvolvem um editor para notações gráficas.

Gramática de reescrita gráfica, a gramática com coordenadas de MILGRAM e ROSENFELD [27] mapeia, através de suas produções, um conjunto de símbolos,localizados por coordenadas dadas, em um novo conjunto de símbolos, cujas coordenadas são computadas por um conjunto de funções associadas à produção dada.

Pode ser formalmente definida como uma sêxtupla, na qual se incluem os conjuntos de símbolos terminais e não ter

minais, um domínio de coordenadas (e.g. conjunto dos inteiros), um índice da dimensão do espaço da posição dos símbolos, um símbolo inicial, e um conjunto finito de produções.

Nestas produções está a particularidade das gramáticas com coordenadas. Cada produção é uma quádrupla (Λ, Σ, Π, ϕ) sendo:

Λ - uma j-tupla de símbolos, para algum $j \geqslant 1$;

Σ - uma k-tupla de símbolos, para algum $k \geqslant 1$;

Π - um predicado com k argumentos, cada um dos quais é uma ênupla de coordenadas;

ϕ - uma j-tupla de funções, cada uma tendo k-argumentos; os argumentos e os valores da função são ênuplas de coordenadas.

onde Λ e Σ denotam os lados direito e esquerdo da regra, ϕ as funções de embutimento, e Π determina as condições de multiplicação e aglutinação de símbolos, na interface com o contexto; Π pode ser interpretado também como seqüência de transformações, de modo que as coordenadas dos símbolos de Λ, sejam um subconjunto das coordenadas das tuplas de símbolos hospedeiros.

Assim quanto ao embutimento, quando ϕ é uma relação, dispensa o predicado Π, e é reversível. Garantindo reversibilidade através da uniformidade na ampliação e redução do número de arestas.

O modelo de MILGRAM e ROSENFELD [27], para as gramáticas de reescritas gráficas, abrange o modelo das "shape grammars" de STINY [28].

No modelo de Stiny, uma regra tem especificação pictórica ou especificação formal. A regra formal $< \sigma , M_1, \ldots, M_n > \rightarrow < \sigma', M_1, \ldots, M_n >$ pode ser interpretada de acordo com o modelo de Rosenfeld, e onde apenas o primeiro componente é um símbolo terminal e a regra de formas se aplica sob uma seqüência de transformadas euclideanas.

Assim, em Stiny — onde as formas geradas e as operações são definidas sobre um universo incontável de formas, dado por um conjunto que contém uma linha reta e um operador estrela — torna-se necessária a criação de um índice n de formas, para diferenciar os universos gerados pelos símbolos terminais e não terminais. Também no modelo de Milgram e Rosenfeld para as gramáticas com coordenadas, a reescrita é feita em ênuplas de formas, dados os índices para as tuplas de símbolos do lado esquerdo e direito de uma regra. O universo de formas tratado fica vinculado diretamente aos argumentos do processo de derivação.

A figura foi descrita, em Stiny, em termos de formas retilíneas, bi-dimensionais, através de conjunto finito de pontos e de linhas, sendo derivada através de operações regularizadas de conjuntos, aplicadas a formas, num mesmo sistema de coordenadas. A substituição da forma correspondente ao lado esquerdo da regra na forma hospedeira, pela correspondente ao lado direito, realiza-se pelas operações de diferen

ças e união de formas, que são depois reduzidas (evitando duplicidade de semi-retas, numa operação que determina uma classe de equivalência). Portanto, acontecerá uma aglutinação de formas apenas se houver uma coincidência de pontos, determinantes das semi-retas envolvidas na substituição. O embutimento é, de fato, normal porque os pontos finais das semi-retas retiradas continuam na lista de pontos, e o conjunto permanece inalterado.

Para as gramáticas de reescrita gráficas (GRG), com coordenadas, inexistem as classes de Chomsky; o vocabulário é infinito e uma GRG n-dimensional com $n \geqslant 1$ pode ser simulada e simular uma máquina de Turing [27].

Outros avanços nos modelos das linguagens de reescrita gráficas envolvem estruturas n-dimensionais, como grafos. Geralmente, nas descrições de figuras que usam grafos [20] como modelos, os vértices representam formas como primitivos e as arestas relações entre estes primitivos. Tornando-se então necessária a adequação da escolha dos primitivos e das relações envolvidas, a um âmbito de figuras a serem geradas ou analisadas.

O modelo das graftais [29], que partilha conceitos de fractais e sistemas de partículas com gramáticas de grafos rotulados, descreve fenômenos naturais para a área de "computer imagery" (arte das imagens feitas por computador). Mas, assim como em [30], os modelos são os "L-systems", estendidos para grafos e conhecidos como gramáticas paralelas de figuras. Obtendo resultados reais em objetos sintéticos [31].

Terminando, incluem-se as abordagens motivadas pelas gramáticas de cláusulas e programação lógica, que resultam numa linguagem de especificação declarativa de figuras [32] e permitem aos programas analisar a estrutura das figuras.

III.2 - Gramáticas com Atributos

Diversas são as características e diverso é o emprego de atributos na literatura — aplicações em otimização de consultas a banco de dados relacionais por YANG e PFALTZ [35] ou asssociados a figuras como em HELM e MARRIOTT [32].

No entanto, como método de definição de linguagem, surgiu o conceito de gramática de atributos, introduzido por KNUTH [36] e reformulado dentro da semântica matemática por MAYOH [37], visando a especificação da semântica de linguagens, definidas pelas gramáticas livres de contexto.

Obtém-se uma gramática de atributos associando a uma gramática livre do contexto um conjunto de atributos e um conjunto de equações semânticas, também chamado de regras semânticas. O propósito dos atributos é transmitir informação entre nós da árvore de "parser" de um "input" dado.

Existem os atributos sintetizados (synthesized),que transmitem a informação na árvore de baixo para cima.E os herdados (inherited), que transmitem a informação na árvore de cima para baixo GALLIER [38].

Dada uma seqüência de símbolos de input, os valores

dos atributos do símbolo inicial são soluções de um sistema de equações, obtidas das regras semânticas associadas com certas instâncias de atributos na árvore de parser JALILI [39]. A noção de atributos associada à gramática de grafos, visando compreensão de desenhos esquemáticos, é encontrada em GÖTTLER [40,41,42,43] e BUNKE [44].

Pontos de interesse de ambas idéias, relativos a esta tese, serão relatados em linhas gerais a seguir.

Göttler desenvolveu uma ferramenta para criar "editores sintaticamente dirigidos", que usem qualquer técnica de diagramas. Estes diagramas modelam os objetos e funções que descrevem e estabelecem a especificação de requisitos para um sistema de "software". Nesta abordagem, os diagramas são representados por grafos, e informações tais como rótulo, forma e altura dos diagramas estão contidas nos atributos.

Em outras palavras, seguindo o processo usual de modelos de gráficos descritos através de grafos, visto no Capítulo III, Göttler descreve seus diagramas: os rótulos dos nós codificam os objetos gráficos (janelas e arestas do diagrama). Assim, cada nó do grafo modela uma entidade gráfica. E as arestas dos grafos modelam a relação "conectada com", entre as entidades gráficas. Deixando aos atributos dos nós do grafo os aspectos morfológicos, isto é, a forma e posição das entidades gráficas.

A manipulação dos diagramas (inserção ou supressão das entidades gráficas, tais como arestas, janelas, textos)

é formalmente descrita pelas gramáticas de grafos com atributos.

A aplicação de uma produção no grafo, em Göttler, exige uma composição entre as entidades de nós e rótulos como em NAGL [5], vide Capítulo II.Exemplo de produção P aplicada em um grafo hospedeiro H: procure em H um nó rotulado A, "insira o lado direito de P". A partir do lugar em H, em que o nó original rotulado por A foi encontrado, siga na direção de todas as arestas rotuladas por i (pode haver mais que uma ou nenhuma). Se, no final, forem achados um ou mais nós, parta destes nós encontrados, na direção contrária de uma aresta rotulada por j. Se, desta forma, um ou mais nós,rotulados por C, forem encontrados, desenhe uma aresta rotulada por k, partindo daí para o nó 4 do "lado direito da produção" já inserido [42]. Detalhe: a produção vem também apresentada em notação de diagrama. Em forma de Y, onde Göttler desenvolveu a teoria, ou em forma de X, por motivos práticos, ao deixar explícito o "contexto constante". Este artíficio evita a substituição redundante de contexto constante, tornando eficiente a atuação dos programas em lisp, que transformam as produções em código. A apresentação da regra em forma de diagramas não traz diferença teórica; sob este aspecto os conectores X ou Y são equivalentes.

Como os grafos têm atributos, as produções são associadas com funções computáveis ou com predicados. Os atributos são passados, na árvore de derivação, de modo similar às gramáticas de atributos do caso de "strings".

Como os nós dos grafos envolvidos na produção são rotulados, expressando um objeto gráfico, e associados a um conjunto de atributos que possuem uma fórmula de avaliação, — a aplicação de uma produção de grafo com atributos tem então dois efeitos: de uma mudança estrutural no grafo de representação e da construção de um programa, para a avaliação de atributos que requerem uma substituição simples. Ver GOTTLER [40,41,42,43].

Em Bunke [44], as gramáticas de grafo programadas, de atributos, usam em suas produções grafos sem atributos. Pois durante o processo de derivação, os atributos, definidos para nós e arestas, são juntados "ao lado direito da regra", de acordo com as funções de atributos.

A transformação de embutimento segue a abordagem de Schneider [8].

A derivação direta de um grafo envolve um predicado de condição de aplicação que, se for verdadeiro, desencadeia o processo de substituição e atribuição de valores, especificados pelas funções de atributos de nós e de arestas, respectivamente.

Estas considerações mostram a importância dos atributos na codificação das numerosas variações que podem ocorrer nas imagens a serem processadas.

CAPÍTULO IV

FORMULAÇÃO DO MODELO (GRAPHICS)

Graphics são grafos com atributos nos seus vértices. Gramáticas de *graphics* são extensões naturais de gramáticas de grafos e atributos, com regras que são a extensão com atributos das produções de gramáticas de grafos, aplicadas através de *push-outs*.

Neste capítulo, a noção das gramáticas de *graphics*, aspectos de implementação e vários exemplos serão apresentados e discutidos.

Aceitando *graphics* como uma generalização útil de grafos, é natural procurar uma definição que também generalize adequadamente ambos conceitos, de gramáticas de grafos e de atributos. As duas definições de gramáticas de grafos existentes na literatura preferiram generalizar uma forma de reescrita de grafo, que pode ser útil em certas aplicações mas ainda assim parece arbitrária. Nesta tese,em IV-1, a definição de gramática de *graphics* se baseia em morfismos de *graphics* e *push-outs*. Esta forma de reescrita de grafos parece ser a mais indicada para uma generalização de reescrita de *strings* por meio de gramáticas de atributos. Um *graphic* é uma entidade de representação nova,porque seus atributos podem ser termos de uma Σ-álgebra. Na formulação proposta, as gramáticas de *graphics* podem se beneficiar dos resultados da abordagem algébrica para gramáticas de grafos, assim como de resultados das Σ-álgebras.

IV.1 - Definição

A generalização natural de um grafo é também a generalização natural do conceito, em linguagens formais, de uma palavra ou *string*: um conjunto de vértices, totalmente ordenado, rotulado por símbolos de um alfabeto finito. Se for tomada uma relação simétrica em vez de uma ordenação total, tem-se:

Definição: um *graphic* G, numa álgebra A e um conjunto de rótulos W, é uma quádrupla G = (K, E, ℓ, f) onde

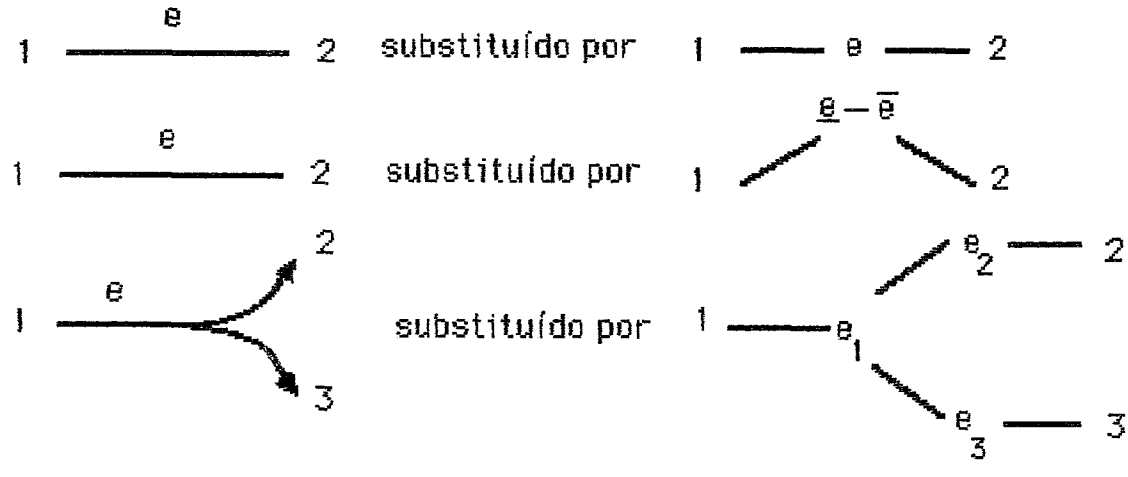
K é um conjunto finito não-vazio de elementos chamados vértices;

E é um conjunto finito de pares não ordenados de vértices chamados de arestas;

ℓ é uma função de K para W;

f é uma função de K para A.

Parece que se perde generalidade ao não permitir rótulos de arestas ou atributos, arestas direcionadas, arestas com mais de dois vértices, ou arestas múltiplas. Isto não acontece porque sendo



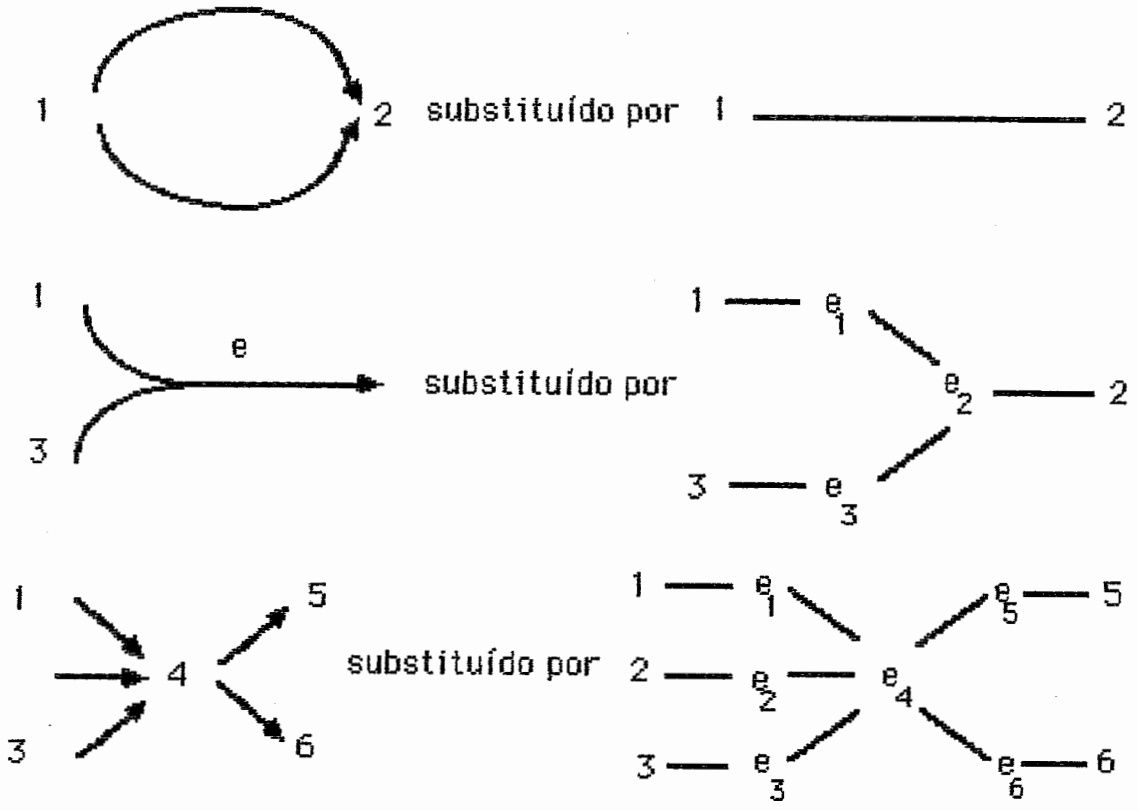


Fig. IV.1

foram introduzidos nós-arestas como novos vértices,cujos rótulos e atributos podem apreender informação de ordem. Considere as substituições como exemplos de novos nós, capturando as informações das arestas. É claro que as regras de substituição de arestas podem ser simuladas por produções de *graphics* e pode-se reconstituir o resultado de uma substituição de *graphic* revertendo o grafo, devolvendo-lhe as arestas direcionadas, rotuladas e com atributos.

IV.2 - Morfismos

Definição: Um morfismo de *graphic* g: G → G' consiste de mapeamentos ver: K → K', edge: E → E', att: A → A' tal que att é um homomorfismo.

edge (v_1, v_2) = (ver(v_1), ver(v_2))

ℓ'(ver(v)) = ℓ(v)

f'(ver(v)) = att(f(v))

Diz-se que G é um *subgraphic* de G' (g : G → G' for um embutimento) quando "ver" for uma injeção.

Convém notar que, com esta definição de morfismo, *graphics* formam uma categoria quando também as álgebras formam uma categoria. Especifica-se a categoria de *graphics* como uma subcategoria de STRUCT, ver [45].Morfismos de *graphics* são combinações naturais de morfismos de grafos e morfismos de álgebras. A definição de *subgraphic* é uma generalização natural da definição de subgrafo.

O caso que desperta maior interesse é aquele cuja categoria de álgebras é a Σ-álgebra para alguma *signature* Σ. Ver Capítulo II.

Exemplo: O *graphic* sobre a álgebra dos termos T(Σ ∪ V) — onde Σ = (R^2, +, x, $\sqrt{\ }$, sen, cos, 0, ...) e x, y, a, φ ε V —

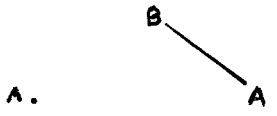


Fig. IV.2

K = {1,2,4}

ℓ(1) = A

ℓ(2) = A

ℓ(4) = B

E = {(2,4)}

f(1)=<x,y>

f(2)=<x + a cosφ, y + a senφ>

f(4)=<x + a cos(φ + 30)/$\sqrt{3}$,
y + a sen(φ + 30)/$\sqrt{3}$>

é um *subraphic* do *graphic* (sobre a álgebra $T(\Sigma \cup V)/\approx$ onde $\approx$ é dada pelas equações que avaliam +, x, sen, cos, ...)

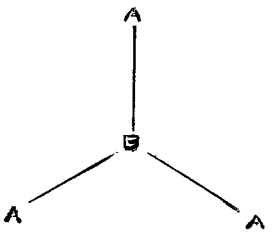


Fig. IV.3

$K' = \{1,2,3,4\}$

$\ell'(1) = A$

$\ell'(2) = A$

$\ell'(3) = A$

$\ell'(4) = B$

$E' = \{(1,4), (2,4), (3,4)\}$

$f'(1) = (0,0)$

$f'(2) = (a,0)$

$f'(3) = (a/2, a\sqrt{3}/2)$

$f'(4) = (a/2, a\sqrt{3}/6)$

porque se tem o morfismo de *graphic*

$$ver(v) = v \qquad att(u,v) = (Su, Sv)$$

onde S é dado substituindo-se 0 por x,y e ϕ. As expressões trigonométricas no *subgraphic* dão a projeção de uma aresta nos eixos x e y.

Todos os *graphics* exemplificados, em IV.2 e IV.3, terão dois atributos espaciais que serão dados pela representação do *graphic* como um diagrama. Deve-se notar que todos os *graphics* têm o graphic 0 como um *subgraphic*. O *graphic-vazio* é útil para remoção de vértices.

Definição: Um sistema de reescrita seqüencial de *graphics* $\Gamma = (V, \Sigma, I, P)$ consiste de

V, um conjunto finito de símbolos, não-vazio

Σ, uma *signature*

I, o *graphic* inicial sobre alguma Σ-álgebra

P, um conjunto de produções, não vazio .

Uma produção $p = B_1 \xleftarrow{B_1} K \xrightarrow{B_2} B_2$ é um par de morfismos que preservam atributos conectando três *graphics*, cujos vértices têm seus atributos com valores em $T(\Sigma \cup V)$ — termos formados de variáveis V e operações na *signature* Σ.

Comentário: A exigência de que os valores dos atributos em B_1, K, B_2 fossem dados por termos em $T(\Sigma \cup V)$ está ligada à escolha da reescrita de morfismos; isto garante que *pushouts* de grafos coincidam com *pushouts* de *graphic*. A exigência também permite aos valores dos atributos conduzirem informações das arestas; se um grafo dirigido for parcialmente ordenado, a ordenação das arestas pode ser obtida por

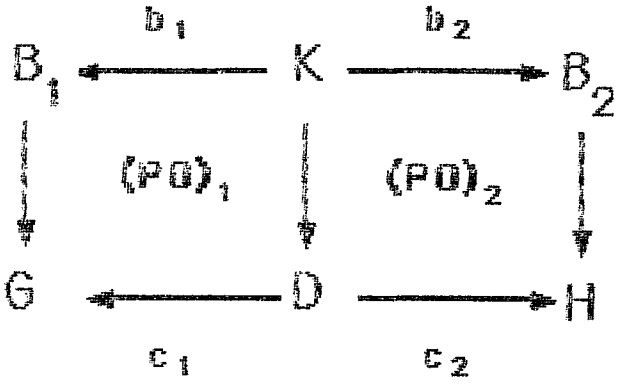
$< t >$ ——— $< t' + t >$ substitui ⟶

Não apenas "depois", "acima" e "à direita de" são exemplos disto, mas também várias ordenações na base de dados e redes semânticas de AI.

IV.3 - Sistema de Reescrita

Definição: Uma derivação direta $G \overset{p}{\Rightarrow} H$ é definida dando-se uma produção, um contexto de *graphic* D, um morfismo de *graphic* $K \to D$, e dois *pushouts* $(PO)_1$ e $(PO)_2$ na catego

ria de *graphics* e morfismos de *graphic* definidos acima.



Diz-se que o grafo H é derivado de G pela produção p, baseado no morfismo $g : B_1 \to G$.

Uma linguagem de *graphic* é definida como sendo o conjunto de todos os *graphics* derivados do *graphic* inicial por produções em P.

Comentário: Os componentes de atributos nos morfismos verticais g, d, h não precisam ser identidades, eles podem ser a mesma avaliação dos termos de $T(\Sigma \cup V)$ para uma Σ-álgebra A. Como foi explicado em EHRIG [13,pp.11], o *push-out* de dois morfismos de grafos $b : K \to B$ e $d : K \to D$ é dado "colando-se os itens b(k) em B e d(k) em D para cada item k em K". Para grafos rotulados pede-se

$$\ell_G([x]) = \textit{se } x \in B \textit{ então } \ell_B(x) \textit{ se não } \ell_D(x) \qquad \text{(IV.1)}$$

para cada vértice [x] em G, o grafo de *pushout* de b e d. Esta é uma definição adequada se $\ell_B(b(K)) = \ell_D(d(K))$ para todo k em K (satisfeita quando b e d são morfismos de *graphics*) e os atributos dos vértices ficam definidos no *graphic*

de *push-out* G por

$$f_G([x]) = \textit{se } x \in B \textit{ então } f_B(x) \textit{ se não } f_D(x) \qquad (IV.2)$$

para cada vértice [x] em G. O que funciona quando $f_B(b(K)) = f_D(d(K))$ para todo k em K (sendo satisfeito quando b e d preservam atributos).

Note-se que esta definição de uma derivação direta $G \overset{p}{\Rightarrow} H$ não depende de um questionamento sobre a existência de um *push-out* — simplesmente, se o *push-out* requisitado não existir, então hão haverá $G \overset{p}{\Rightarrow} H$.

Exemplo: Uma derivação direta:

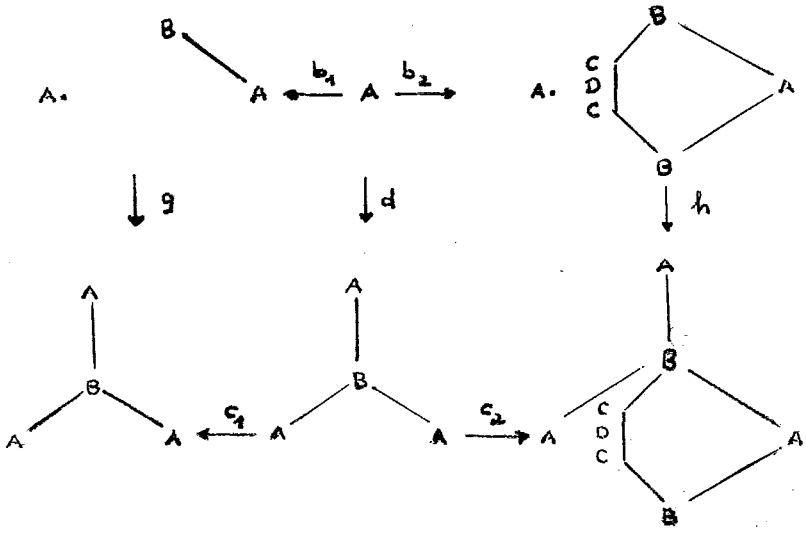


Fig. IV.4

Todos os morfismos neste diagrama são embutimentos (fontes são *subgraphics* de sumidouros); os dois morfismos superiores dão a produção e o morfismo de g foi tratado no exemplo anterior. O quadrado esquerdo é um pushout quando d atribui as coordenadas <0,0> ao nó rotulado A, situado no destino mais à direita. Com esta mesma atribuição de valores aos

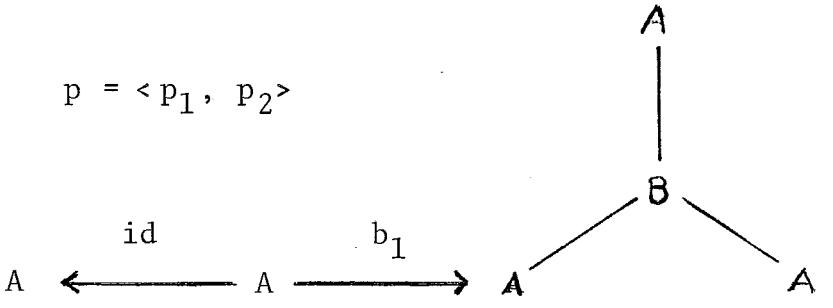
atributos o quadrado direito do diagrama é também um *pushout*. Os parâmetros a e ϕ do exemplo anterior servem para fixar os morfismos, b_1 e b_2 — eles parametrizam uma família de produções.

Exemplo: Uma gramática de *graphics* para descrever o padrão de "wall paper" com o "planegroup" p31m MACMILLAN [46]

$G = \langle V, I, P \rangle$ $W = \langle A, B, C, D \rangle$

$I = A\langle 0,0 \rangle$ $p = \langle p_1, p_2 \rangle$

p1:



$K = \{1\}$ $K = \{1,2,3,4\}$

$E = \{\ \}$ $E = \{\langle 1,4 \rangle, \langle 2,4 \rangle, \langle 3,4 \rangle\}$

$L(1) = A$ $L(1) = A\ L(2) = A\ L(3) = A\ L(4) = B$

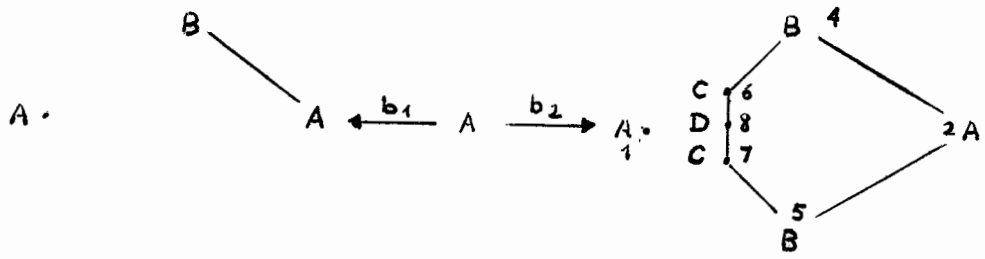
$f(1) = \langle x,y \rangle$ $f(1) = \langle x,y \rangle$ $f(2) = \langle x + a, y \rangle$

$f(3) = \langle x + a/2, y + \sqrt{3}\ a/2 \rangle$

$f(4) = \langle x + a/2, y + \sqrt{3}\ a/6 \rangle$

Fig. IV.5

p2:



K = {1,2,4}

K = {1,2,4,5,6,7,8}

E = {2,4}

E = {<2,4>, <4,6>, <6,8>, <8,7>, <7,5>, <5,2>}

L(1) = A L(2) = A L(4) = B L(5) = B L(6) = C

L(7) = C L(8) = D

f(1) = <x,y> f(2) = <x + a cosϕ, y + a senϕ>

f(4) = <x + a cos(ϕ + 30)/√3, y + a sen(ϕ + 30)/√3>

f(5) = <x + a cos(ϕ - 30)/√3, y + a sen(ϕ - 30)/√3>

f(8) = <x + a cosϕ/5, y + a senϕ/5>

f(6) = <x + a cosϕ/5 + a cos(ϕ + 90)√3/24,
y + a senϕ/5 + a sen(ϕ + 90)√3/24>

f(7) = <x + a cos(ϕ)/5 + a cos(ϕ - 90)√3/24,
y + a sen(ϕ - 90)√3/24>

Fig. IV.6

O uso repetido da produção p1 dá o padrão:

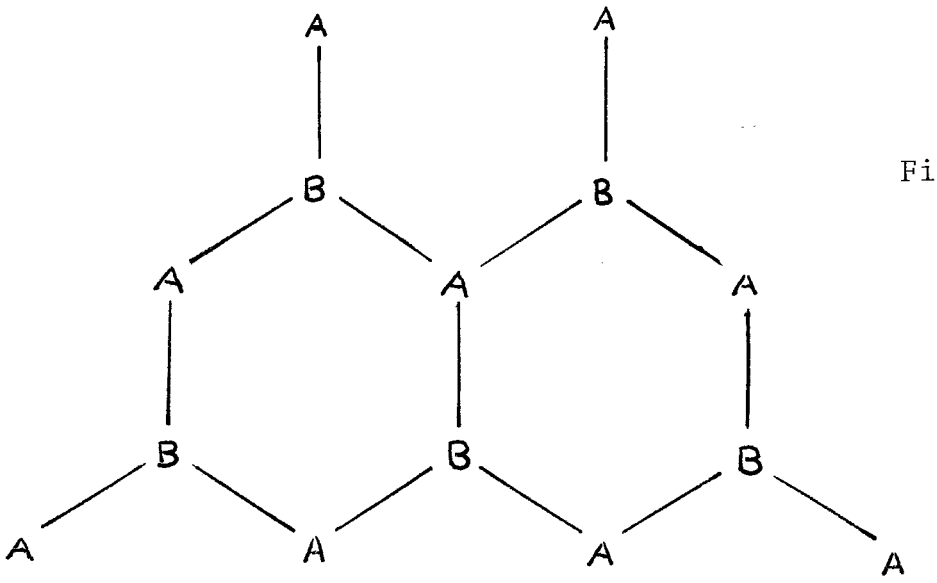


Fig. IV.7

e o uso repetido da produção p2 dará:

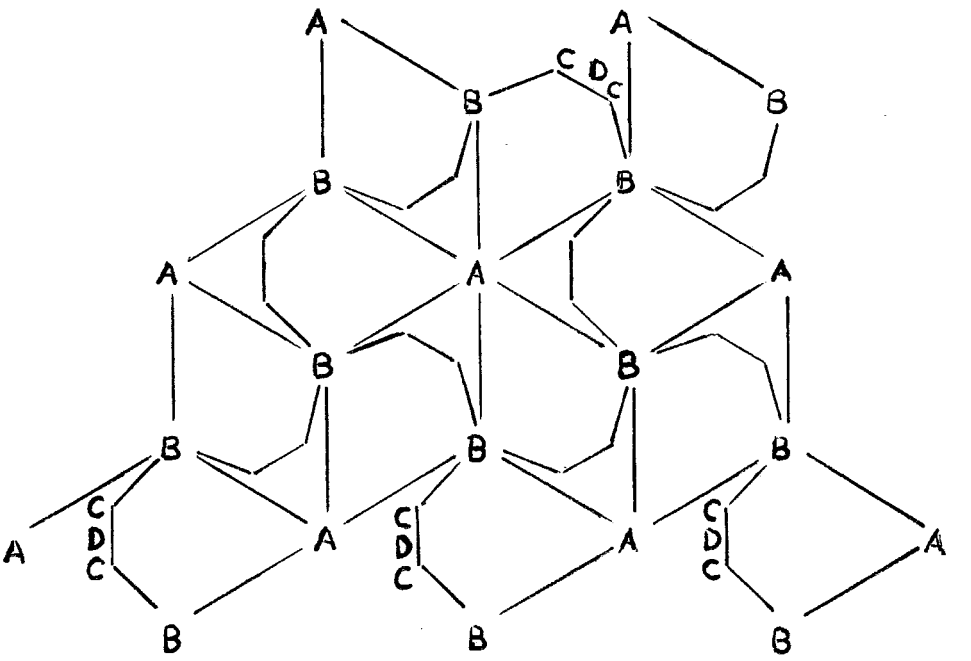


Fig. IV.8

A descrição da derivação de um *graphic* G em uma linguagem de *graphic* foi dada como se os atributos fossem avaliados durante o processo de derivação. Entretanto, não existe objeção a primeiro se derivar o grafo subjacente de G na linguagem de grafo, correspondente a L, e então derivar os valores dos atributos. O que corresponde ao método usual de avaliação em gramáticas de atributos para strings: primeiro derive o string na gramática livre de contexto subjacente, depois resolva as equações dos atributos.

Nota final: Se houver um Σ-morfismo μ da Σ-álgebra *graphic* inicial de uma gramática para qualquer Σ-álgebra A, então se obtém uma transformação da linguagem de *graphic* aplicando-se μ a cada passo de cada derivação na linguagem.O que apreende as transformações geométricas das figuras.

IV.4 - APLICAÇÕES

Serão os *graphics* modelos úteis da realidade?

Apresentaremos os *graphics* como modelos para organismos biológicos e para solucionar o problema de representação do conhecimento do mundo real, conforme proposto em "Sistemas Especialistas e Base de Dados". Também serão descritos vários modos de implementação de gramáticas de *graphics*.

Graphics têm sido definidos como grafos cujos vértices têm valores atribuidos. Quando se permite a atribuição de valores aos vértices, os modelos de grafos se tornam ainda mais úteis. Serão indicadas, a seguir, algumas das possíveis aplicações de *graphics*, bem como suas gramáticas em di-

versos autores.

Geração e análise de imagens:

Identificação de aeronaves [24].

Modelos estocásticos de terreno [32].

Análise de cenas [44].

Análise e geração de diagramas:

Estrutura de programas [40].

Figuras estruturadas [32].

Editor interativo de figuras [32].

Programas de animação [32].

Diagramas de circuito [40].

Arte [47].

Supervisor de janelas [32].

Análise Numérica:

Geração de grade para resolução de DDE's [54].

Modelos biológicos:

Morfogênesis [48].

Representação do conhecimento:

Esquemas conceituais em base de dados.

Redes semânticas em AI [49].

Os modelos de grafos se tornaram populares, mais precisos e convenientes quando seus vértices assumiram valores de atributos. Entretanto, a utilidade das gramáticas de grafos depende muito dos tipos de regras de reescrita que forem permitidos. A reescrita por *push-out* proporciona a flexibilidade que não possuem nem a reescrita de um único morfis-

mo (morfismos de identidade de K para B_1, e de D para G) nem as regras de trocas simples de vértices ou arestas.

Já ficou bem estabelecido que o desenvolvimento de organismos biológicos [48] pode ser modelado por grafos, e o exemplo 1 mostra que a extensão para *graphics* proporciona modelos mais exatos no desenvolvimento de sistemas especialistas, as redes semânticas são um modo popular de se representar "conhecimento do mundo real" [49] e o exemplo 2 mostra que *graphics* são um formalismo conveniente para redes semânticas.

Exemplo 1: "Phascum cuspidatum"

A gramática para o desenvolvimento do organismo "phascum cuspidatum" e o *graphic* de um estágio particular de vida são mostrados em

I: a

P: a ⇐ a ⇒ b - a

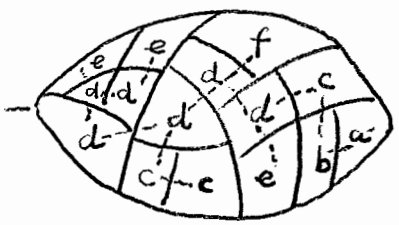
b ⇐ b ⇒ c

c ⇐ c ⇒ d - e

d ⇐ d ⇒ d

e ⇐ e ⇒ f

f ⇐ f ⇒ c - c



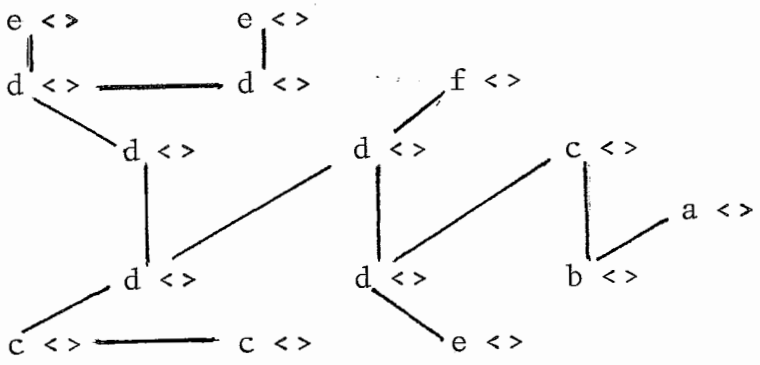


Fig. IV.9

As arestas no grafo reaparecem como linhas pontilhadas na figura. Como os atributos das células do "phascum cuspidatum" não são usados nesta gramática de *graphics* (exceto pela convenção de que o único vértice no *graphic* esquerdo de uma produção seja reescrito no vértice mais à esquerda do *graphic* direito desta produção), ela não é mais do que uma gramática de grafo tradicional. Entretanto, esta gramática ilustra dois pontos que são valiosos em modelagem biológica.

. As gramáticas trabalham melhor quando existe apenas um número finito de rótulos de células, mas o desenvolvimento de um organismo é mais naturalmente descrito em termos de um número infinitamente grande de estados, e os atributos de *graphics* podem apreender esta infinidade.

. As gramáticas dão comumente um resultado não ambíguo quando uma regra é aplicada, mas o desenvolvimento de um organismo é mais naturalmente descrito ao serem permitidas aplicações flexíveis de regras. Os atributos de *graphics* podem apreender esta flexibilidade.

Exemplo 2: "Redes semânticas"

Uma gramática para a representação dos interrelacionamentos humanos e o *graphic* para uma família particular são mostrados em

I: M(Adam,0) F(Eva,0)

P: M(mn,mt) F(fn,ft)⇐M(mn,mt) F(fn,ft)⇒M(mn,mt) F(fn,ft)

onde dt>mt+10 e dt>ft+10 F(dn,dt)

M(mn,mt) F(fn,ft)⇐M(mn,mt) F(fn,ft)⇒M(mn,mt) F(fn,ft)

onde st>mt+10 e st>ft+10 M(sn,st)

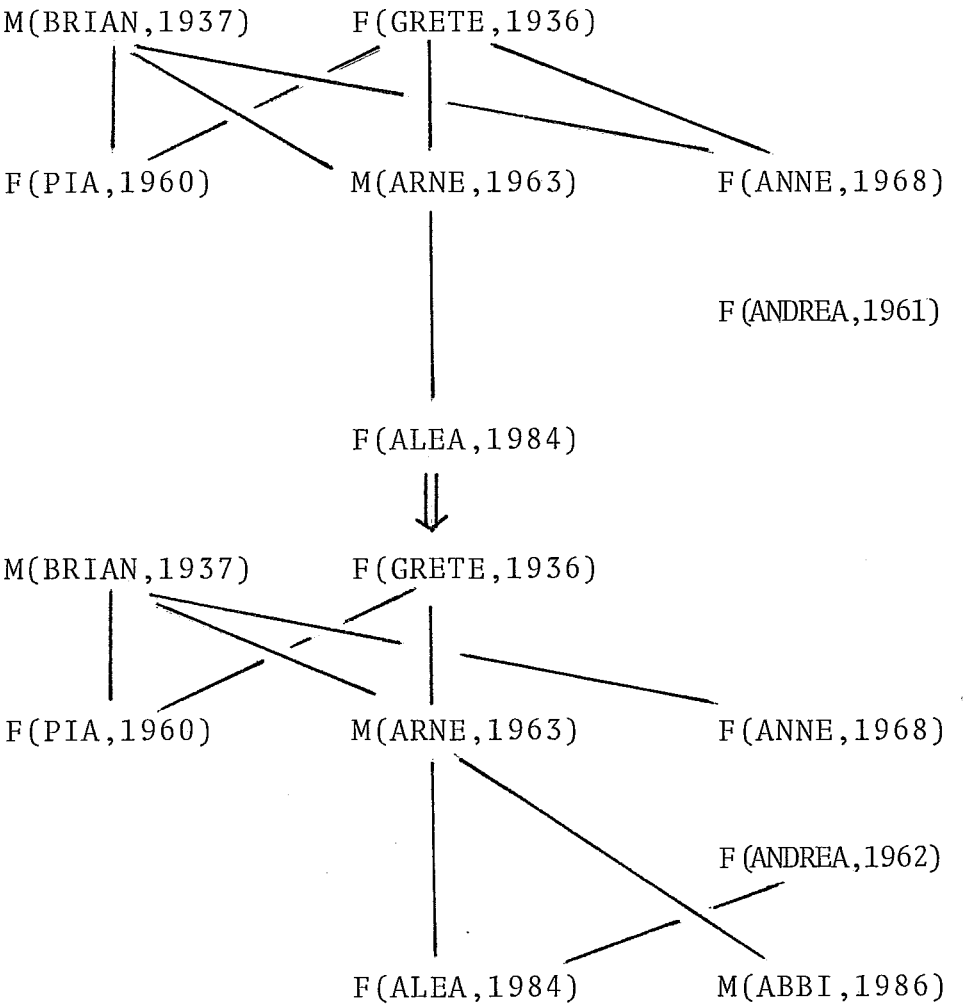


Fig. IV.10

A aplicação da segunda regra neste exemplo é flexível, porque a fixação dos parâmetros — mn = ARNE, mt = 1963, fn = ANDREA, ft = 1962 — não determinou os atributos: sn = = ABBI e st = 1968. O exemplo é instrutivo porque mostra como atributos de "tempo" permitem aos *graphics* apreender ordenações naturais, as quais seriam expressas por arestas direcionadas nos modelos de grafos. Outros exemplos ilustrariam como graphics podem apreender outras formas de informação de arestas, que os construtores de sistemas especialistas colocam em redes semânticas (reportar-se à referência sobre informação de arestas, em IV.1).

Exemplo 3: Apagamento

Em [49], Sowa introduziu uma forma de redes semânticas chamadas de grafos conceituais para a gramática de graphics do exemplo 2, e descreveu as operações adequadas à extração de informação e manipulação de grafos conceituais. Todas estas operações podem ser estendidas para *graphics*. Considere a seguinte adição de regras, quando então arestas e vértices poderão ser removidos.

M(mn,mt)⇐0⇒0 - remoção de vértice,

F(fn,ft)⇐0⇒0 usando o *graphic* vazio 0.

M(mn,mt)-F(fn,ft)⇐M(mn,mt) F(fn,ft)⇒M(mn,mt) F(fn,ft)

M(mn,mt)-M(n,t) ⇐ M(mn,mt) M(n,t) ⇒ M(mn,mt) M(n,t)

F(fn,ft)-F(n,t) ⇐ F(fn,ft) F(n,t) ⇒ F(fn,ft) F(n,t)

As duas primeiras regras dão a remoção de vértices, usando o *graphic* vazio 0. As últimas três regras proporcionam remoção de arestas.

Uma vez aceito que *graphics* e suas gramáticas são modelos úteis da realidade, passa-se a requerer uma implementação destes no computador; e a desejar um programa de análise, para reconhecer se um dado *graphic* pode ser gerado por uma gramática dada, e finalmente a desejar um programa de síntese para a geração de *graphics* a partir de gramáticas.

Os vários modos de implementação de *graphics* e suas gramáticas podem ser grupados em tradicional, concorrente, lógico e sintático. Desta forma serão mostrados respectivamente cada um destes grupos.

Implementação Tradicional

Se a linguagem escolhida para implementar graphics e suas gramáticas for uma linguagem imperativa como Pascal, ou uma linguagem orientada para objetos como Smalltalk, é natural que se represente *graphics* como "data object". É também natural se representar uma regra da gramática por um par de rotinas: uma rotina de reconhecimento *bottom-up* para substituir o lado direito pelo lado esquerdo, e uma rotina de geração *top-down* para substituir o lado esquerdo pelo lado direito.

```
type  Vertex
      access record; colour: V; x co, y co: Real
      end record;

type  Edge  is record source, sink: vertex.
            end record;

procedure Generate p2(in i1,i2,i4: Vertex; e: Edge; a,phi: Real
                      out i5,i6,i7,i8: Vertex;success: Boolean
                          e1,e2,e3,e4,e5: Edge);
          x,y,u,v: Real;

  begin x   :=i1.x co;y:=i1.y co;
    success :=(i1.colour=A)and(e.source=i2)and(e.sink=i4)
              and(i2.colour=A)and(i4.colour=B)
              and(i2.x co=x + a × cos phi)
              and(i2.y co=x + a × sen phi)
              and(i4.x co=x + a × cos(phi + 30)/√3)
              and(i4.y co=x + a × sen(phi + 30)/√3);

  if success then
     i5 := new Vertex(colour⇒B;x co⇒x + a cos(phi-30)/√3);
           y co⇒x + a sen(phi-30)/√3);
     u  := x + a × cos(phi)/5;v :=y + a × sen(phi)/5;
     i8 := new Vertex(colour⇒D,x co⇒u,y co ⇒ v);
     i6 := new Vertex(colour⇒ C,
                                x co⇒u + a × cos(phi+90)× sqrt(3)/24,
                                y co⇒v + a × sen(phi+90)× sqrt(3)/24);
     i7 := new Vertex(colour ⇒ C,
                                x co⇒u + a × cos(phi-90)× sqrt(3)/24,
                                y co⇒v + a × sen(phi-90)× sqrt(3)/24);
     e1 :=(source⇒i2,sink⇒i5);
     e2 :=(source⇒i4,sink⇒i6);
     e3 :=(source⇒i6,sink⇒i8);
     e4 :=(source⇒i7,sink⇒i8);
     e5 :=(source⇒i5,sink⇒i7);
   end if;
end Generate p2;
```

Fig. IV.11

Usando-se recursividade decrescente e outros métodos usuais em escrita de compiladores não é difícil escrever programas de análise e síntese, uma vez que se tenha rotinas para "escolha de uma produção" e "encontro de ocorrência". A única diferença entre um programa de análise e um programa tradicional de *parsing* é que o teste para "alcance do *graphic* inicial I de uma gramática" pode ser bastante complicado. A distinção entre símbolos terminais e não-terminais como em linguagens formais é irrelevante.

Notar que a rotina para se encontrar uma ocorrência de uma regra $B_1 \leftarrow K \rightarrow B_2$ corresponde a se encontrar um morfismo $B_1 \rightarrow G$. O que corresponde na implementação a estabelecer alguns dos parâmetros atuais de uma chamada da função para a regra (os outros parâmetros podem ser estabelecidos pelo usuário). As implementações de base de dados e sistemas especialistas poderiam ser incluídas neste grupo, como implementações "tradicionais" de *graphics* e suas gramáticas.

Implementações concorrentes

Se a linguagem escolhida para implementar *graphics* e suas gramáticas for uma linguagem concorrente como OCCAM/CSP com seus processos, CCS com seus agentes,ou ADA com suas tarefas, é natural a identificação de *vértices* em um *graphic* com processos/agentes/tarefas. As arestas em um graphic podem ser dadas por *links/channels*; os atributos de um vértice podem ser dados por variáveis locais de um processo e o rótulo de um vértice corresponde ao tipo de seu processo. Desde que uma linguagem concorrente pode apreender a não determina

ção inerente das gramáticas, não é difícil escrever programas de análise e síntese, uma vez que se tenha rotinas de geração e análise para as regras da gramática. Como as aplicações mostram, precisa-se de uma linguagem de programação bastante flexível, de modo a escrever rotinas de reconhecimento e geração.

```
operation Generate p22(i1,i2,i4,i5,i6,i7,i8,a,φ)
condition old  Task(i1,A,x,y);
          old  Task(i2,A,x + a cosφ;y + a senφ)
          old  Task(i4,B,x + a cos(φ+30)/√3,y + a sen(φ+30)√3)
          old  Link(i2,i4)
action    new  Task(i5,B,x + a cos(φ-30)/√3,y+ a sen(φ-30)/√3)
          new  Task(i8,D,u:=x + a cos(φ)/5,v:=y + a sen(φ)/5)
          new  Task(i6,C,u + a cos(φ+90)√3/24;v + a sen(φ+90)√3/24)
          new  Task(i7,C,u + a cos(φ-90)√3/24,v +a sen(φ-90)√3/24)
          new  Link(i2,i5)new Link(i6,i8);
          new  Link(i7,i8)new Link(i5,i7)
end operation

operation Recognise p2(i1,i2,i4,i5,i6,i7,i8,a,φ)
condition old  Task(i1,A,x,y);
          old  Task(i2,A,x + a cosφ,y + a senφ)
          old  Task(i4,B,x + a cos(φ+30)/√3,y + a sen(φ+30)/√3)
          old  Task(i5,B,x + a cos(φ-30)/√3,y + a sen(φ-30)/√3)
          old  Task(i8,D,u:=x + a cosφ/5,v:=y + a senφ/5)
          old  Task(i6,C,u + a cos(φ+90)√3/24,v + sen(φ+90)√3/24)
          old  Task(i7,C,u + a cos(φ-90)√3/24,v + sen(φ-90)√3/24)
          old  Link(i2,i4)old Link(i2,i5)old Link(i4,i6)
          old  Link(i6,i8)old Link(i7,i8)old Link(i5,i7)
action    drop Link(i2,i5)drop Link(i4,i6)
          drop Link(i6,i8)drop Link(i7,i8)drop Link(i5,i7)
          drop Task(i5)drop Task(i6)drop Task(i7)drop Task(i8)
end operation
```

Fig. IV.12

Implementação lógica

Se a linguagem escolhida para implementar graphics e suas gramáticas for uma linguagem lógica como PROLOG, é natural que se represente *graphics* como um conjunto de fórmulas atômicas. Um modo de se apreender arestas em *graphics* é indexar os vértices e conectar os índices por fórmulas atômicas.

Visto que as regras de gramática produzem novos *graphics* a partir dos velhos, as fórmulas que foram verdadeiras antes da aplicação da regra podem ser falsas depois. Esta não monotonicidade pode ser apreendida de três modos:

(A) Introduzindo um novo atributo de "estágio" em todas as fórmulas, e expressando a regra da gramática por uma implicação lógica.

(B) Usando operadores não lógicos do PROLOG, como *assert* e *retract*.

(C) Mudando para um "metanível".

Devido ao problema enfocado o modo (A) torna-se inviável, assim ilustra-se (B) e (C) através do exemplo de rede semântica.

(A) Current graphic P(1,A,0).P(2,A,6,0).
P(3,A,3,$3\sqrt{3}$).P(4,B,3,$\sqrt{3}$)
E(1,4).E(2,4).E(3,4).

(B) Generate-p2(i1,i2,i4,a,ϕ):-P(i1,A,x,y).P(i2,A,x+a cosϕ,y+a senϕ)
P(i4,B,x+a cos(ϕ+30)/$\sqrt{3}$,y + a sen(ϕ+30)/$\sqrt{3}$)
E(i2,i4)
assert P(<i1,i2,i4,5>,B,x+a cos(ϕ-30)/$\sqrt{3}$, y+a sen(ϕ-30)/$\sqrt{3}$)
assert P(<i1,i2,i4,8>,D,x+a cosϕ/5,y + a senϕ/5)
assert P(<i1,i2,i4,6>,C,x+a cosϕ/5 + a cos(ϕ + 90)$\sqrt{3}$/24,
y + a senϕ/5 + a sen(ϕ + 90)/$\sqrt{3}$/24)
assert P(<i1,i2,i4,7>,C,x+a cosϕ/5 + a cos(ϕ - 90)/$\sqrt{3}$/24),
y + a senϕ/5 + a sen(ϕ - 90)/$\sqrt{3}$/24)
assert E(<i1,i2,i4,6>,<i1,i2,i4,8>)
assert E(<i1,i2,i4,7>,<i1,i2,i4,8>)
assert E(<i1,i2,i4,5>,<i1,i2,i4,7>)

Recognise-p2(i1,i2,i4,i5,i6,i7,i8,a,ϕ):-P(i1,A,x,y)
P(i2,A,x + a cosϕ,y + a senϕ)
P(i4,B,x + a cos(ϕ + 30)/$\sqrt{3}$, y + a sen(ϕ + 30)/$\sqrt{3}$) E(i2,i4)
P(i5,B,x + a cos(ϕ - 30)/$\sqrt{3}$, y + a sen(ϕ - 30)/$\sqrt{3}$) E(i2,i5)
u is x + a cosϕ/5 v is y + a senϕ/5, P(i8,D,u,v)
P(i6,C,u + a cos(ϕ + 90)$\sqrt{3}$/24, v + a sen(ϕ + 90)$\sqrt{3}$/24)
P(i7,C,u + a cos(ϕ - 90)$\sqrt{3}$/24, v + a sen(ϕ - 90)$\sqrt{3}$/24)
E(i6,i8) E(i8,i7) E(i5,i7)
retract P(i5,B,u + a cos(ϕ + 30)/$\sqrt{3}$, y + a sen(ϕ + 30)/$\sqrt{3}$)
retract P(i8,D,u,v).retract E(i2,i5) retract E(i5,i7)
retract P(i6,C,u + a cos(ϕ + 90)$\sqrt{3}$, v + a sen(ϕ + 90)$\sqrt{3}$/24)
retract P(i7,C,u + a cos(ϕ - 90)$\sqrt{3}$, v + a sen(ϕ - 90)$\sqrt{3}$/24)
retract E(i4,i6) retract E(i6,i8) retract E(i8,i7)

(C) Gen-p2(G,Union(G,Assertions)):-In(G,Union(P(i1,A,x,y)...E(i2,i4))).
Rec-p2(Union(G,Retractions),G):-In(G,Union(P(i1,A,x,y)...E(i5,i7))).

Fig. IV.13

Implementação sintática

Se a gramática de graphic for implementada por um sistema de reescrita de termos, como em REVE, ver [50], um graphic é representado por um termo com componentes para vértices e arestas. Uma regra da gramática é representada por uma regra de reescrita "condicional"; reescrevendo em uma direção se produz uma rotina de geração; regras de reescrita para reorganizar termos são usadas para encontrar ocorrências de produções da gramática.

- Regras de geração

P(i1,A,x,y); P(i3,A, x + a cosϕ, y + a senϕ);
P(i4,B,x + a cos(ϕ + 30)/$\sqrt{3}$, y + a sen(ϕ + 30)/$\sqrt{3}$); E(i2,i4)
$\Rightarrow$P(i1,A,x,y)...E(i1,i4)
P(<i1,i2,i4,5>,B,x + a cos(ϕ - 30)/$\sqrt{3}$, y + a sen(ϕ - 30)) ...
E(<i1,i2,i4,5>,<i1,i2,i4,7>)

- Regras de reorganização

E(i,j); E(k,ℓ) $\Rightarrow$ E(k,ℓ); E(i,j) E(i,j) $\Rightarrow$ E(j,i)
E(i,j); P(k,ℓ, x,y) $\Rightarrow$P(k, ℓ,x,y); E(i,j)
P(k,ℓ,x,y); P(kk,$\ell\ell$,xx,yy) $\Rightarrow$ P(kk, $\ell\ell$,xx,yy); P(k,ℓ,x,y)

Fig. IV.14

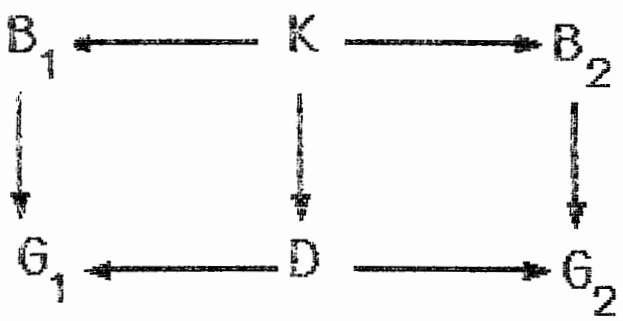
As regras de reescrita foram apresentadas na direção de síntese. As últimas quatro regras não alteram o *graphic* subjacente, mas precisam ser incluídas no sistema de

reescrita para simular todas as derivações da gramática de *graphic*. Se todas as regras de reescrita forem revertidas e interrompido o processo quando o graphic I for encontrado, pode-se analisar qualquer *graphic* na linguagem dada pela gramática.

A afirmativa de que qualquer gramática de *graphic* pode ser simulada tão precisamente por um sistema de reescrita de termos é verdadeira se for mostrado que as gramáticas de *graphics* são gramáticas de tipos de dados algébricos [51]. Para tanto, torna-se necessário:

. Achar uma especificação de tipo abstrato de dados cuja álgebra inicial seja isomorfa ao conjunto de *graphics*.

. Descrever como as produções de *graphic* $B_1 \leftarrow K \rightarrow B_2$ podem ser convertidas para um par $< t_1, t_2 >$ de Σ-termos de modo que:



Se e somente se $G_1 = [D(t_1)]$ e $G_2 = [D(t_2)]$

Para algum Σ-termo $D(x)$.

A versão de gramática de grafos, que foi mostrada

como sendo uma gramática de tipo abstrado de dado algébrico, em [53], é a seguinte:

1. Arestas são seqüências de nós com rótulos.

2. Nós podem ter rótulos e atributos em forma de arestas unárias.

3. Todas as produções têm um grafo discreto finito $\overline{n}$ como K.

Graphics satisfazem (1), (2), "V_1 V_2 é uma aresta → não é rotulada &V_1 V_2 é uma aresta" e "apenas arestas unárias e binárias". Não se requer que as produções nas gramáticas de *graphics* satisfaçam (3), mas o poder de geração das gramáticas de *graphics* não é afetado mesmo sendo exigida a restrição (3), retirando-se todas as arestas de K (na aplicação da produção estas arestas desaparecem de D mas reaparecem em G_1 e G_2 porque estão ainda presentes em B_1 e B_2).

Pode-se implementar *graphics* e suas gramáticas quando se tem acesso a um sistema eficiente de base de dados ou a um "sistema para construção de sistemas especialistas", que existe disponível no mercado. Nestes sistemas *graphics* podem ser representados por bases de conhecimento, e as regras de gramáticas podem ser representadas por transações. Num sistema de base de dados uma transação é uma seqüência para achar, inserir, deletar e modificar ações; em um sistema especialista uma transação é uma regra de produção da forma "*se* condição *então* ação".

CAPÍTULO V

CONCLUSÕES

Nesta tese foi apresentado o conceito de *graphics* na extensão desse termo em inglês, que significa também a arte de desenhar figuras, como em arquitetura ou engenharia,de acordo com regras matemáticas, implicando que sobre estas figuras podem ser calculados esforços e visualizados sistemas dinâmicos. Assim, associando planos a gráficos amplia-se por conseguinte sua definição — *graphics* são modelos gráficos para descrição de mecanismos de raciocínio e processos de conhecimento.

Um *graphic* é uma nova entidade representacional porque seus atributos são termos de uma Σ-álgebra. A definição das gramáticas de *graphics* é baseada em morfismos de *graphics* e *pushouts*. Esta forma de reescrita de grafos é a extensão mais direta das gramáticas de *strings*, tendo os resultados desta abordagem algébrica de grafos beneficiado a reescrita de *graphics*. Especificando:ao absorver o valor da abordagem algébrica, esta forma de reescrita de *graphics* trata o conjunto de produções como um sistema de reescrita confluente, estabelecendo uma seqüência de produções não arbitrária.

A representação por *graphics* traz a vantagem de uma modelagem em dois níveis — mais precisamente, no nível de grafo e no nível de atributos, símbolos variáveis e operadores de uma Σ-álgebra —, através da maleabilidade conceitual

obtida com a aplicação de diferentes Σ-álgebras isomorfas, seguindo a concepção do exemplo em IV.3 e dos programas inclusos no apêndice, nos quais os atributos de um *graphic* tomam valores alternadamente no espaço n-dimensional e nos grupos de simetria.

Como primeiras pesquisas, espera-se aplicar *graphics* na obtenção de padrões de simetria e como auxílio na catalogação de padrões de fibras, vasos e células da madeira, visualizados no corte de seus eixos tangencial e radial.

Sendo qualquer morfismo de *graphic* intrinsecamente um morfismo de grafo e um homomorfismo de Σ-álgebras, como trabalho futuro, propõe-se obter todas as vantagens dos isomorfismos de um *graphic*, o que permitirá a transferência de resultados obtidos, de uma álgebra para outra.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] EHRIG, H., VOLKER, C., ROZENBERG, G., Eds., Proc. 1st Int. Workshop on Graph Grammars and their Application to Computer Science and Biology, *Lecture Notes in Computer Science* 73, Springer Verlag, Berlim, 1979.

[2] EHRIG, H., NAGL, M., ROZENBERG, G., Eds., Proc. 2nd Int. Workshop on Graph Grammars and their Application to Computer Science, *Lecture Notes in Computer Science* 153, Springer Verlag, Berlim, 1983.

[3] EHRIG, H., NAGL, M., ROZENBERG, G., Eds., Proc. 3nd Int. Int. Workshop on Graph Grammars and their Application to Computer Science, *Lecture Notes in Computer Science* 291, Springer Verlag, Berlim, 1987.

[4] PFALTZ, J. e ROSENFELD, A., "Web Grammars", Proc. Int, Joint Conf. Artificial Intelligence, Washington, pp. 609-619, 1969.

[5] NAGL, M. "A Tutorial and Bibliographical Survey on Graph Grammars", *Lecture Notes in Computer Science* 73,Springer Verlag, Berlim, pp. 70-125, 1979.

[6] MONTANARI, U.G., "Separable Graphs, Planar Graphs, and Web Grammars", *Information and Control* 16, pp. 243. 267, 1970.

[7] ROSENFELD, A. e MILGRAM, D., "Web Automata and Web Grammars", Tech. Report TR 181, University of Maryland, Computer Science Center, Maryland, 1972.

[8] SCHNEIDER, H.J., *Chomsky-Like Systems for Partialy Ordered Symbol Sets*, Tech. Rep. 2/2/71, Informations-Verarbeitung II, TU Berlim, 1971.

[9] NAGL, M., "Formal Languages of Labelled Graphs", *Computing* 16, pp. 113-137, Fasc. 1-2, Springer Verlag, Berlim, 1976.

[10] GOGUEN, J.A., THATCHER, J.W. e WAGNER, E.G., "An Initial Algebra Approach to the Specification, Correcteness and Implementation of Abstract Data Types", em Yeh, R.T. Ed., *Current Trends in Programming Methodology*, Vol IV, *Data Structuring* Prentice Hall, Englewood Cliffs, New Jersey, 1978.

[11] HUET, G. e OPPEN, D.C., "Equations and Rewrite Rules. A Survey", *Formal Language Theory Perspectives and Open Problems.*, Book R., Ed., Academic Press, Inc., New York, 1980.

[12] EHRIG, H. e MAHR, B., *Fundamentals of Algebraic Specification* 1: Equations and Initial Semantics, Springer Verlag, Berlim, 1985.

[13] EHRIG, H., "Introduction to the Algebraic Theory of Graph Grammars. (A Survey)", *Lecture Notes in Computer Science* 73, Springer Verlag, Berlim, pp. 1-89, 1979.

[14] EHRIG, H. e KREOWSKI, H.J., "Applications of Graph Grammar Theory to Consistency Synchronization and Scheduling in Data Base Systems",*Information Systems*, Pergamon Press, Grã-Bretanha, Vol 5, pp. 225-238, 1980.

[15] EHRIG, H. e ROSEN, Barry K., "Decomposition of Graph Grammar Productions and Derivations", *Lecture Notes in Computer Science* 73, Springer Verlag, Berlim, 1979.

[16] BORROW, L. e ARBIB, M., *Discrete Mathematics: Applied Algebra for Computer and Information Science*, Hemisphere PU. Co., Saunders Company, 1974.

[17] SZWARCFITER, Jayme L., *Grafos e Algoritmos Computacionais*, Rio de Janeiro, Editora Campus, 1984.

[18] FURTADO, A.L. e MYLOPOULOS, J., "Using Graph Grammars to Define Sets of Diagraphs", *Infor*, Vol. 17, No. 3, Agosto, 1979 e em Tech. Rep. 1/75 PUC/RJ, 1975.

[19] MILLER, W.F. e SHAW, A.C.,"Linguistic Methods in Picture Processing - A Survey", Fall Joint Computer Conference Dec 9-11, San Francisco, Cal 1968, Afips Conference Proceedings, Vol. 33, Part 1, 1968.

[20] NAKE, F. e ROSENFELD, A., *Graphic Languages*,Proceedings of the Ifip Working Conference on Graphic Languages, Vancouver Canada May 22-26, 1972 North-Holland, Amsterdam, 1972.

[21] ROSENFELD, Azriel, *Picture Languages - Formal Models for Picture Recognition*, Academic Press, New York, 1979.

[22] KIM, C. e SUDBOROUGH, I.H., "The Membership and Equivalence Problems for Picture Languages", *Theoretical Computer Science* 52, pp. 177-191, 1987.

[23] KRITHIVASAN, Kamala e DAS, Anĩndya, "Terminal Weighted Grammars and Picture Description", *Computer Vision, Graphics and Image Processing* 30, pp. 13-31, 1985.

[24] YOU, K.C. e FU, K.S., "Distorted Shape Recognition Using Attributed Grammars and Error - Correcting Techniques", *Computer Graphics and Image Processing*, 13, pp. 1-16, 1980.

[25] LIN, Wei-Chung e FU, King-Sun, "3D-Plex Grammars", *Information Sciences* 34, pp. 1-24, Elsevier, New York, 1984.

[26] WOODMAN, M., INCE, D., PREECE, J. e DAVIES, G., "A Grammar Formalism as a Basis for the Syntax-Directed Editing of Graphical Notations", *Workstations and Publication Systems*, Ed. Earnshaw e Rae, A., Springer-Verlag, New York, 1987.

[27] ROSENFELD, A. e MILGRAM, D.L., "A Note on Grammars with Coordinates", *Graphic Languages*, Proceedings of the Ifip Working Conference on Graphic Languages,Amsterdam, 1972.

[28] STINY, N., *Pictorial and Formal Aspects of Shape and Shape Grammars and Aestetic Systems*, Ph.D. Thesis, UCLA, Los Angeles,1975.

[29] SMITH, Alvy Ray, "Pants,Fractals,and Formal Languages", *Computer Graphics*, Vol. 18, No. 3, ACM, 1984.

[30] PRUSINKIEWICZ, P., "Applications of L-Systems to Computer Imagery", *Lecture Notes in Computer Science* 291, Springer Verlag, Berlim, pp. 534-548, 1987.

[31] PRUSINKIEWICZ, P., LINDENMAYER, A. e HANAN, J., "Developmental Models of Herbaceous Plants for Computer Imagery Purposes", *Computer Graphics*, Vol. 22, No. 4, ACM, pp. 141-150, 1988.

[32] HELM, R. e MARRIOT, K., "Declarative Graphics", Third International Conference in Logic Programming,*Lecture Notes in Computer Science* 225, Springer Verlag, Berlim, 1986.

[33] HESS, Lilia A., "Process of Reducing Figures into Graphic Structures; Minimization and Continuity of Graphic Paths", Series: Monographs in Comp. Sci. and Comp. Appl., No. 8/71, PUC, Rio de Janeiro, 1971.

[34] BALLARD, D.H. e BROWN, M.C., *Computer Vision*, Prentice Hall, New Jersey, 1982.

[35] YANG, D. e PFALTZ, J., "Attribute Domain Partitions, a Tool for Efficient, Optimization", Sig. Mod Conference, 1987.

[36] KNUTH, D.E., "Semantics of Context Free Languages", *Math. System Theory* 2, pp. 127-145, 1968, com correção em *Siam J. Computing* 5, pp. 95, 1971.

[37] MAYOH, Brian H., "Attribute Grammars and Mathematical Semantics", *Siam J. Computing*, Vol. 10, No. 3, pp. 503-518, Agosto, 1981.

[38] GALLIER, J., "An Efficient Evaluator for Attribute Grammars with Conditional Rules", Tr. Dep. Comp. and Info Sciences, University Pennsylvania, MS. CIS 83-86, Filadelfia, Pennsylvania, Maio, 1984.

[39] JALILI, F. e GALLIER, J.H., "A General Incremental Evaluator for Attribute Grammars", Tr. Dep. Comp. and Info Sciences, University Pennsylvania, MS. CIS 81-11, Filadelfia, Pennsylvania, Março, 1983.

[40] GÖTTLER, H., "Attributed Graph Grammars for Graphics", *Lecture Notes in Computer Science* 153, Springer-Verlag, Berlim, pp. 130-142, 1983.

[41] GÖTTLER, H., "Implementation of Attributed Graph Grammars", Proceedings of the WG'84 "Graphtheoretic Concepts in Comp. Sci.", Universitätsverlag Rudolf Trauner,Linz, 1984.

[42] GÖTTLER, H., "Graph Grammars and Diagram Editing", *Lecture Notes in Computer Science* 291, Springer Verlag, Berlim, 1987.

[43] GÖTTLER, H., "Graph Grammars, a New Paradigm for Implementing Visual Languages",*Lecture Notes on Rewriting Systems* , pp. 152-166, Berlim, 1989.

[44] BUNKE, H.,"Graph Grammars as a Generative Tool in Image Understanding", *Lecture Notes in Computer Science* 153, Springer Verlag, Berlim, pp. 8-19, 1983.

[45] EHRIG, H., KREOWSKI, H.J., MAGGIOLO-SCHETTINI, A., ROSEN, B.K. e WINKOWSKI, J., "TRANSFORMATION OF STRUCTURES: An Algebraic Approach", *Math. System. Theory* 14, pp. 305-334, 1981.

[46] LOCKWOOD, E.H. e MACMILLAN, R.H., *Geometric Symmetry*, Cambridge Univ. Press, 1978.

[47] NAGL, M., *Graph-Grammatiken*, Braunschweig,Vieweg, 1979.

[48] ROZENBERG, G., SALOMAA, Eds., *The Book of L*, North-Holland, 1986.

[49] SOWA, J.F., *Conceptual Structures*,Addison-Wesley,1984.

[50] LESCANNE, P., "Computer Experiments with the REVE Term Rewriting Systems Generator", Proc. 10th. Symp. Pr. Prog. Lang., pp. 99-108, 1983.

[51] EHRIG, H., HABEL, A., HUMMART, U. e BOEHM, P., "Towards Algebraic Datatype Grammars: A Junction Between Algebraic Specifications and Graph Grammars", *Bulletin Eacts* 29, pp. 22-27, 1986.

[52] MACGILLAVRY, C.H., *Fantasy & Symmetry. The Periodic Drawings of M.C. Escher,* Abrahams, New York, 1976.

[53] BAUDERON, M. e COURCELLE, B., "An Algebraic Formalism for Graphs", em CAAP'86, *Lecture Notes in Computer Science* 214, Springer Verlag, Berlim, 1986.

[54] KALABA, R. e CASTI, J.L., "Numerical Grid Generation", *Applied Math. Computation,* pp. 1-895, 1982.

[55] PADAWITZ, P., "Graph Grammars and Operational Semantics", *Theoretical Computer Science* 19, pp. 117-141, 1982.

[56] HESS, L. e MAYOH, B.H., "Graphics and their Grammars", *Lecture Notes in Computer Science* 153, Springer Verlag, Berlim, pp. 232-249, 1987 e em *Daimi* PB - 223, AArhus Univ., Março 1987.

APÊNDICE

Os atributos dos *graphics* deste apêndice tomam valores nas Σ-álgebras:

do espaço n-dimensional (*n-tuple space*), onde:

F campo de escalares

V conjunto de objetos chamados vetores (conjunto de n-uplas de escalares em F)

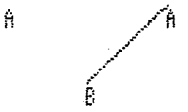
+, . símbolos (denotando soma e produto)

e dos grupos de simetria, onde:

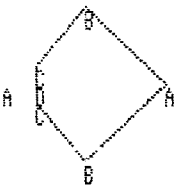
movimentos de congruência

sucessão destes movimentos (denotando produto)

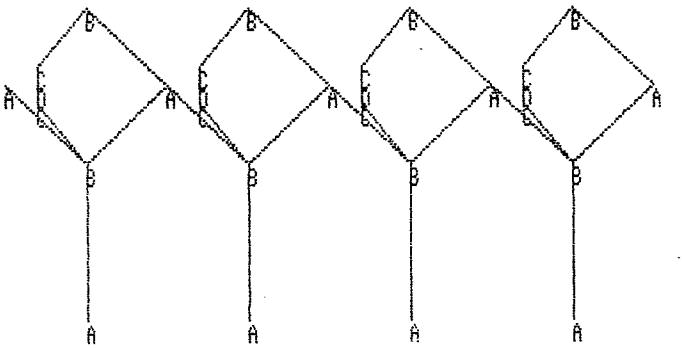
As figuras a seguir mostram a regra de reescrita e aplicação desta na geração de um padrão p31m. Os respectivos programas são listados em seqüência.



Lado Esquerdo da Regra



Lado Direito da Regra



Geracao Automatica da Figura IV.8

```
 1    PROGRAM Lilia1;
 2
 3    USES Graph, Crt;
 4
 5    TYPE
 6        VERTEX = RECORD
 7            colour      : CHAR;
 8            x_co, y_co : REAL;
 9        END;
10
11        EDGE = RECORD
12            source : VERTEX;
13            sink   : VERTEX;
14        END;
15
16    VAR
17        NewV              : VERTEX;
18      (* IN *)
19        V1,V2,V3,V4       : VERTEX;
20        E                 : EDGE;
21      (* OUT *)
22        V5,V6,V7,V8       : VERTEX;
23        E1,E2,E3,E4,E5 : EDGE;
24
25        ok : BOOLEAN;
26        GraphDriver, GraphMode : INTEGER;
27        ch : CHAR;
28        a  : REAL;
29
30
31
32    PROCEDURE Generate_P2 (i1,i2,i4 : VERTEX; e : EDGE; a,phi : REAL;
33        VAR i5,i6,i7,i8 : VERTEX; VAR success : BOOLEAN;
34        VAR e1,e2,e3,e4,e5 : EDGE);
35
36        VAR
37            x,y,u,v : REAL;
38
39    BEGIN
40        x := i1.x_co;
41        y := i1.y_co;
42        phi := phi * Pi / 180;
43
44        success :=  ((i1.colour = 'A') AND
45        (e.source.colour = i2.colour) AND
46        (e.source.x_co = i2.x_co) AND
47        (e.source.y_co = i2.y_co) AND
48        (e.sink.colour = i4.colour) AND
49        (e.sink.x_co = i4.x_co) AND
50        (e.sink.y_co = i4.y_co) AND
51        (i2.colour = 'A') AND
52        (i4.colour = 'B') AND
53        (i2.x_co = x + a * Cos(phi)) AND
54        (i2.y_co = y + a * Sin(phi)) AND
55        (i4.x_co = x + a * Cos(Pi / 6) / Sqrt(3)) AND
```

```
56          (i4.y_co = y + a * Cos(Pi / 6) / 3));
57
58          IF (success)
59        THEN BEGIN
60             WITH NewV DO
61           BEGIN
62                colour := 'B';
63                x_co   := x + a * Cos(phi - Pi / 6) / Sqrt(3);
64                y_co   := y + a * Sin(phi - Pi / 6) / Sqrt(3);
65           END;
66             i5 := NewV;
67             u  := x + a * Cos(phi) / 5;
68             v  := y + a * Sin(phi) / 5;
69             WITH NewV DO
70           BEGIN
71                colour := 'D';
72                x_co   := u;
73                y_co   := v;
74           END;
75             i8 := NewV;
76             WITH NewV DO
77           BEGIN
78                colour := 'C';
79                x_co   := u + a * Cos(phi + Pi / 2) * Sqrt(3) / 24;
80                y_co   := v + a * Sin(phi + Pi / 2) * Sqrt(3) / 24;
81           END;
82             i6 := NewV;
83             WITH NewV DO
84           BEGIN
85                colour := 'C';
86                x_co   := u + a * Cos(phi - Pi / 2) * Sqrt(3) / 24;
87                y_co   := v + a * Sin(phi - Pi / 2) * Sqrt(3) / 24;
88           END;
89             i7 := NewV;
90
91             e1.source := i2;
92             e1.sink   := i5;
93
94             e2.source := i4;
95             e2.sink   := i6;
96
97             e3.source := i6;
98             e3.sink   := i8;
99
100            e4.source := i7;
101            e4.sink   := i8;
102
103            e5.source := i5;
104            e5.sink   := i7;
105       END
106         ELSE Write(chr(7));
107   END;
108
109   BEGIN
110   a := 200. * Cos(Pi/3);
```

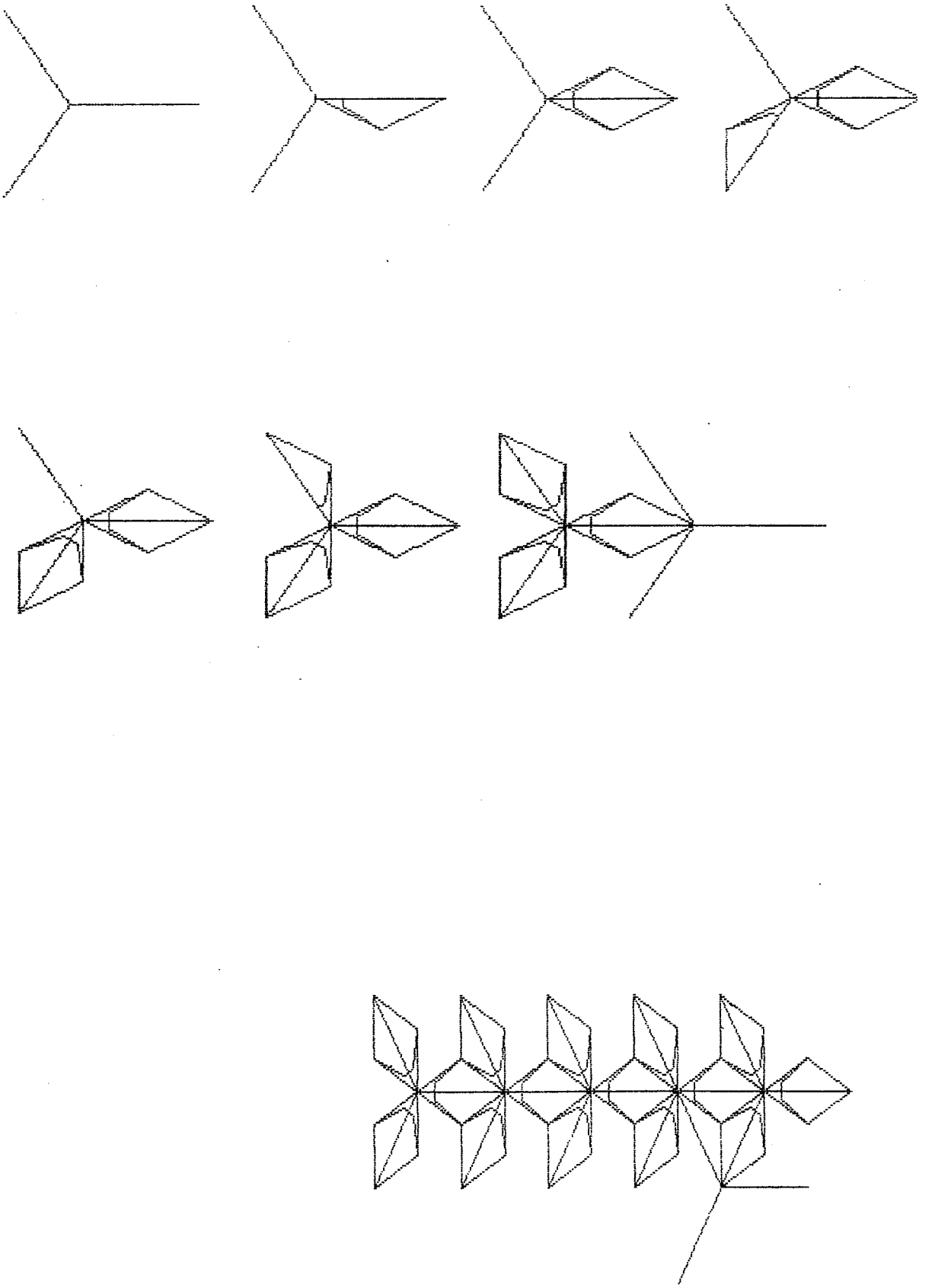
```
111
112     V1.colour := 'A'; V1.x_co := 0.;     V1.y_co := 0.;
113     V2.colour := 'A'; V2.x_co := a;      V2.y_co := 0.;
114     V3.colour := 'A'; V3.x_co := a / 2; V3.y_co := a * Sqrt(3) / 2;
115     V4.colour := 'B'; V4.x_co := a / 2; V4.y_co := a * Sqrt(3) / 6;
116
117     E.source := V2;
118     E.sink   := V4;
119
120     Generate_P2(V1,V2,V4,E, a, 0., V5,V6,V7,V8, ok , E1,E2,E3,E4,E5);
121
122     GraphDriver := DETECT;
123     InitGraph(GraphDriver, GraphMode, '');
124     SetTextStyle(SmallFont, HorizDir, 4);
125
126     SetViewPort(120,150,639,349, ClipOff);
127     SetColor(EGAWhite);
128     OutTextXY(300,80,'Lado Direito da Regra');
129     OutTextXY(0,80,'Lado Esquerdo da Regra');
130     OutTextXY(Round(V1.x_co),Round(V1.y_co),V1.colour);
131     OutTextXY(Round(V2.x_co),Round(V2.y_co),V2.colour);
132     OutTextXY(Round(V4.x_co),Round(V4.y_co),V4.colour);
133     SetColor(EGARed);
134     Line(Round(E.source.x_co),Round(E.source.y_co),
135     Round(E.sink.x_co),  Round(E.sink.y_co));
136
137     SetViewPort(420,150,639,349, ClipOff);
138     SetColor(EGAWhite);
139     OutTextXY(Round(V1.x_co),Round(V1.y_co),V1.colour);
140     OutTextXY(Round(V2.x_co),Round(V2.y_co),V2.colour);
141     OutTextXY(Round(V4.x_co),Round(V4.y_co),V4.colour);
142     SetColor(EGAYellow);
143     Line(Round(E.source.x_co),Round(E.source.y_co),
144     Round(E.sink.x_co),  Round(E.sink.y_co));
145
146     SetColor(EGAWhite);
147     OutTextXY(Round(V5.x_co),Round(V5.y_co),V5.colour);
148     OutTextXY(Round(V6.x_co),Round(V6.y_co),V6.colour);
149     OutTextXY(Round(V7.x_co),Round(V7.y_co),V7.colour);
150     OutTextXY(Round(V8.x_co),Round(V8.y_co),V8.colour);
151     SetColor(EGAYellow);
152     Line(Round(E1.source.x_co),Round(E1.source.y_co),
153     Round(E1.sink.x_co),  Round(E1.sink.y_co));
154     Line(Round(E2.source.x_co),Round(E2.source.y_co),
155     Round(E2.sink.x_co),  Round(E2.sink.y_co));
156     Line(Round(E3.source.x_co),Round(E3.source.y_co),
157     Round(E3.sink.x_co),  Round(E3.sink.y_co));
158     Line(Round(E4.source.x_co),Round(E4.source.y_co),
159     Round(E4.sink.x_co),  Round(E4.sink.y_co));
160     Line(Round(E5.source.x_co),Round(E5.source.y_co),
161     Round(E5.sink.x_co),  Round(E5.sink.y_co));
162
163     ch := ReadKey;
164     CloseGraph;
165     END.
```

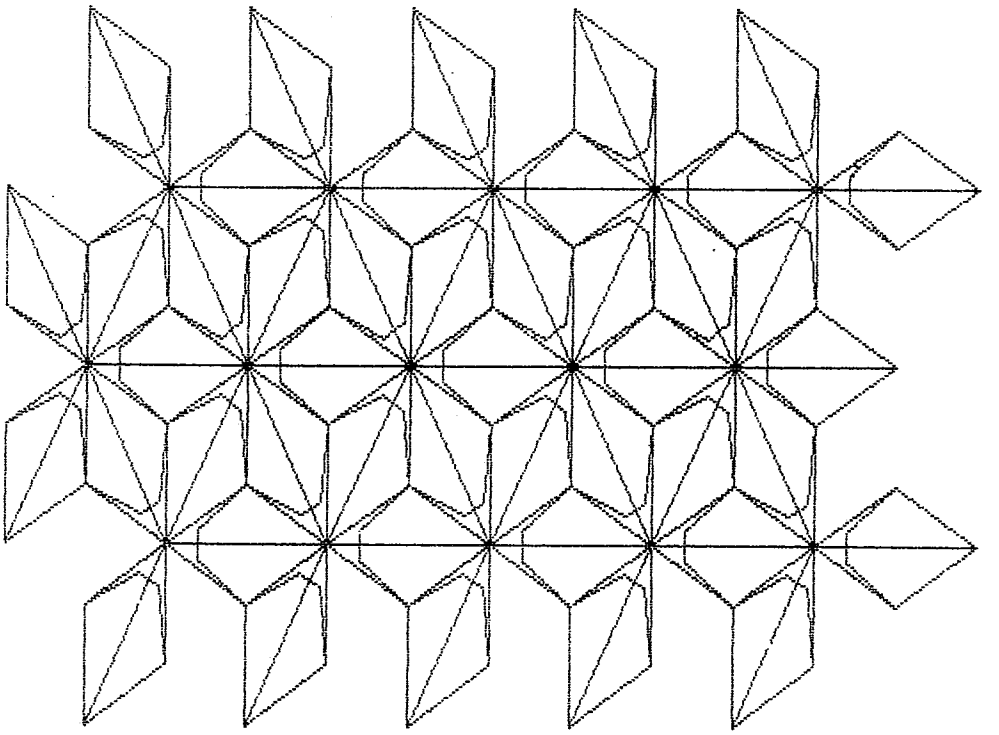
```
 1      PROGRAM Lilia2;
 2
 3      USES Graph, Crt;
 4
 5      TYPE
 6          VERTEX = RECORD
 7              colour     : CHAR;
 8              x_co, y_co : REAL;
 9          END;
10
11          EDGE = RECORD
12              source : VERTEX;
13              sink   : VERTEX;
14          END;
15
16      VAR
17          NewV             : VERTEX;
18        (* IN *)
19          V1,V2,V3,V4      : VERTEX;
20          E,EE,EEE         : EDGE;
21        (* OUT *)
22          V5,V6,V7,V8      : VERTEX;
23          E1,E2,E3,E4,E5 : EDGE;
24
25          ok : BOOLEAN;
26          GraphDriver, GraphMode : INTEGER;
27          ch : CHAR;
28          a  : REAL;
29
30
31  ----PROCEDURE Generate_P1 (V1 : VERTEX; VAR V2,V3,V4 : VERTEX;
      VAR e1,e2,e3 : EDGE);
32  ----BEGIN
33          V2.colour := 'A';
34          V2.x_co := V1.x_co + a;      V2.y_co := V1.y_co;
35          V3.colour := 'A';
36          V3.x_co := V1.x_co + a / 2; V3.y_co := V1.y_co + a * Sqrt(3)
             / 2;
37          V4.colour := 'B';
38          V4.x_co := V1.x_co + a / 2; V4.y_co := V1.y_co + a * Sqrt(3)
             / 6;
39
40          e1.source := V1;
41          e1.sink   := V4;
42          e2.source := V2;
43          e2.sink   := V4;
44          e3.source := V3;
45          e3.sink   := V4;
46          SetColor(EGAWhite);
47          OutTextXY(Round(V1.x_co),Round(V1.y_co),V1.colour);
48          OutTextXY(Round(V2.x_co),Round(V2.y_co),V2.colour);
49          OutTextXY(Round(V3.x_co),Round(V3.y_co),V3.colour);
50          OutTextXY(Round(V4.x_co),Round(V4.y_co),V4.colour);
51          SetColor(EGARed);
52          Line(Round(E1.source.x_co),Round(E1.source.y_co),
```

```
53          Round(E1.sink.x_co),  Round(E1.sink.y_co));
54          Line(Round(E2.source.x_co),Round(E2.source.y_co),
55          Round(E2.sink.x_co),  Round(E2.sink.y_co));
56          Line(Round(E3.source.x_co),Round(E3.source.y_co),
57          Round(E3.sink.x_co),  Round(E3.sink.y_co));
58      END;
59
60
61  PROCEDURE Generate_P2 (i1,i2,i4 : VERTEX; e : EDGE; a,phi : REAL;
62          VAR i5,i6,i7,i8 : VERTEX; VAR success : BOOLEAN;
63          VAR e1,e2,e3,e4,e5 : EDGE);
64
65          VAR
66              x,y,u,v : REAL;
67
68  BEGIN
69          x := i1.x_co;
70          y := i1.y_co;
71          phi := phi * Pi / 180;
72
73          success :=  ((i1.colour = 'A') AND
74          (e.source.colour = i2.colour) AND
75          (e.source.x_co = i2.x_co) AND
76          (e.source.y_co = i2.y_co) AND
77          (e.sink.colour = i4.colour) AND
78          (e.sink.x_co = i4.x_co) AND
79          (e.sink.y_co = i4.y_co) AND
80          (i2.colour = 'A') AND
81          (i4.colour = 'B') AND
82          (i2.x_co = x + a * Cos(phi)) AND
83          (i2.y_co = y + a * Sin(phi)) AND
84          (i4.x_co = x + a * Cos(Pi / 6) / Sqrt(3)) AND
85          (i4.y_co = y + a * Sin(Pi / 6) / Sqrt(3)) );
86
87          IF (success)
88      THEN BEGIN
89              WITH NewV DO
90              BEGIN
91                  colour := 'B';
92                  x_co   := x + a * Cos(phi - Pi / 6) / Sqrt(3);
93                  y_co   := y + a * Sin(phi - Pi / 6) / Sqrt(3);
94              END;
95              i5 := NewV;
96              u  := x + a * Cos(phi) / 5;
97              v  := y + a * Sin(phi) / 5;
98              WITH NewV DO
99              BEGIN
100                 colour := 'D';
101                 x_co   := u;
102                 y_co   := v;
103             END;
104             i8 := NewV;
105             WITH NewV DO
106             BEGIN
107                 colour := 'C';
```

```
108          x_co   := u + a * Cos(phi + Pi / 2) * Sqrt(3) / 24;
109          y_co   := v + a * Sin(phi + Pi / 2) * Sqrt(3) / 24;
110        END;
111        i6 := NewV;
112        WITH NewV DO
113        BEGIN
114          colour := 'C';
115          x_co   := u + a * Cos(phi - Pi / 2) * Sqrt(3) / 24;
116          y_co   := v + a * Sin(phi - Pi / 2) * Sqrt(3) / 24;
117        END;
118        i7 := NewV;
119
120        e1.source := i2;
121        e1.sink   := i5;
122
123        e2.source := i4;
124        e2.sink   := i6;
125
126        e3.source := i6;
127        e3.sink   := i8;
128
129        e4.source := i7;
130        e4.sink   := i8;
131
132        e5.source := i5;
133        e5.sink   := i7;
134      END
135      ELSE Write(chr(7));
136  END;
137
138  PROCEDURE Generate(phi : REAL);
139  BEGIN
140
141      SetColor(EGAWhite);
142      OutTextXY(Round(V5.x_co),Round(V5.y_co),V5.colour);
143      OutTextXY(Round(V6.x_co),Round(V6.y_co),V6.colour);
144      OutTextXY(Round(V7.x_co),Round(V7.y_co),V7.colour);
145      OutTextXY(Round(V8.x_co),Round(V8.y_co),V8.colour);
146      SetColor(EGAYellow);
147      Line(Round(E1.source.x_co),Round(E1.source.y_co),
148      Round(E1.sink.x_co),Round(E1.sink.y_co));
149      Line(Round(E2.source.x_co),Round(E2.source.y_co),
150      Round(E2.sink.x_co),Round(E2.sink.y_co));
151      Line(Round(E3.source.x_co),Round(E3.source.y_co),
152      Round(E3.sink.x_co),Round(E3.sink.y_co));
153      Line(Round(E4.source.x_co),Round(E4.source.y_co),
154      Round(E4.sink.x_co),Round(E4.sink.y_co));
155      Line(Round(E5.source.x_co),Round(E5.source.y_co),
156      Round(E5.sink.x_co),Round(E5.sink.y_co));
157
158  END;
159
160    BEGIN
161    GraphDriver := DETECT;
162    InitGraph(GraphDriver, GraphMode, '');
```

```
163     SetTextStyle(SmallFont, HorizDir, 4);
164
165     SetViewPort(120,150,639,349, ClipOff);
166     OutTextXY(100,140,'Geracao Automatica da Figura IV.8');
167
168     a := 200. * Cos(Pi/3);
169
170     V1.colour := 'A'; V1.x_co := 0.;    V1.y_co := 0.;
171     Generate_P1(V1,V2,V3,V4,E,EE,EEE);
172     Generate_P2(V1,V2,V4,E, a, 0.0, V5,V6,V7,V8, ok , E1,E2,E3,E4,E5);
173     Generate(0.0);
174     Generate_P1(V2,V1,V3,V4,E,EE,EEE);
175     Generate_P2(V2,V1,V4,E, a, 0.0, V5,V6,V7,V8, ok , E1,E2,E3,E4,E5);
176     Generate(0.0);
177     Generate_P1(V1,V2,V3,V4,E,EE,EEE);
178     Generate_P2(V1,V2,V4,E, a, 0.0, V5,V6,V7,V8, ok , E1,E2,E3,E4,E5);
179     Generate(0.0);
180     Generate_P1(V2,V1,V3,V4,E,EE,EEE);
181     Generate_P2(V2,V1,V4,E, a, 0.0, V5,V6,V7,V8, ok , E1,E2,E3,E4,E5);
182     Generate(0.0);
183
184     ch := ReadKey;
185     CloseGraph;
186     END.
```





```
    1   PROGRAM Lilia3;
    2
    3   USES Graph, Crt;
    4
    5   TYPE
    6       VERTEX = RECORD
    7           colour     : STRING[2];
    8           x_co, y_co : REAL;
    9       END;
   10
   11       EDGE = RECORD
   12           source : VERTEX;
   13           sink   : VERTEX;
   14       END;
   15
   16   VAR
   17       NewV             : VERTEX;
   18     (* IN *)
   19       V0,V1,V3,V5      : VERTEX;
   20       E,EE,EEE         : EDGE;
   21     (* OUT *)
   22       V7,V12,V13,V18,V19 : VERTEX;
   23       E1,E2,E3,E4        : EDGE;
   24       EE1,EE2,EE3,EE4    : EDGE;
   25
   26       ok : BOOLEAN;
   27       Xasp, Yasp : WORD;
   28       GraphDriver, GraphMode, ii : INTEGER;
   29       ch : CHAR;
   30       a : REAL;
   31
   32  ---PROCEDURE Linha(x1,y1,x2,y2 : INTEGER);
   33  ---BEGIN
   34       y1 := Round(y1 * Xasp / Yasp);
   35       y2 := Round(y2 * Xasp / Yasp);
   36       Line(x1,y1,x2,y2);
   37  ---END;
   38
   39  ---PROCEDURE Roda(Vert0 : VERTEX; ang : REAL; VAR Vert1 : VERTEX);
   40       VAR
   41           V : VERTEX;
   42
   43  ---BEGIN
   44       ang := ang * Pi / 180.0;
   45       Vert1.x_co := Vert1.x_co - Vert0.x_co;
   46       Vert1.y_co := Vert1.y_co - Vert0.y_co;
   47       V.x_co := Vert1.x_co * Cos(ang) - Vert1.y_co * Sin(ang);
   48       V.y_co := Vert1.x_co * Sin(ang) + Vert1.y_co * Cos(ang);
   49       Vert1.x_co := V.x_co + Vert0.x_co;
   50       Vert1.y_co := V.y_co + Vert0.y_co;
   51  ---END;
   52
   53  ---PROCEDURE Generate_P1 (V0 : VERTEX; VAR V1,V3,V5 : VERTEX;
   54       VAR e1,e2,e3 : EDGE);
   55  ---BEGIN
```

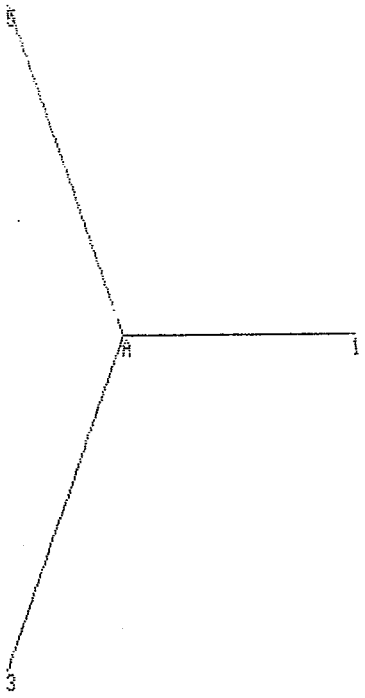
```
56          V1.colour := '1';
57          V1.x_co := V0.x_co + a;
58          V1.y_co := V0.y_co;
59          V3.colour := '3';
60          V3.x_co := V0.x_co + a * Cos(2*Pi/3);
61          V3.y_co := V0.y_co + a * Sin(2*Pi/3);
62          V5.colour := '5';
63          V5.x_co := V0.x_co + a * Cos(2*Pi/3);
64          V5.y_co := V0.y_co - a * Sin(2*Pi/3);
65
66          e1.source := V0;
67          e1.sink   := V1;
68          e2.source := V0;
69          e2.sink   := V3;
70          e3.source := V0;
71          e3.sink   := V5;
72          SetColor(EGAWhite);
73      (*
74        OutTextXY(Round(V0.x_co),Round(V0.y_co),V0.colour);
75        OutTextXY(Round(V1.x_co),Round(V1.y_co),V1.colour);
76        OutTextXY(Round(V3.x_co),Round(V3.y_co),V3.colour);
77        OutTextXY(Round(V5.x_co),Round(V5.y_co),V5.colour);
78      *)
79          SetColor(EGARed);
80          Linha(Round(E1.source.x_co),Round(E1.source.y_co),
81          Round(E1.sink.x_co),  Round(E1.sink.y_co));
82          Linha(Round(E2.source.x_co),Round(E2.source.y_co),
83          Round(E2.sink.x_co),  Round(E2.sink.y_co));
84          Linha(Round(E3.source.x_co),Round(E3.source.y_co),
85          Round(E3.sink.x_co),  Round(E3.sink.y_co));
86      END;
87
88
89  PROCEDURE Generate_P2 (i0,i1 : VERTEX; e : EDGE; a : REAL;
90          VAR i7,i13,i19 : VERTEX; VAR e1,e2,e3,e4 : EDGE);
91
92          VAR
93              x,y,u,v : REAL;
94
95  BEGIN
96          x := i0.x_co;
97          y := i0.y_co;
98          IF (TRUE)
99      THEN BEGIN
100             WITH NewV DO
101         BEGIN
102                 colour := '7';
103             x_co   := x + a / 2;
104             y_co   := y + a * Sqrt(3) / 6;
105         END;
106             i7 := NewV;
107             u  := x + a / 5;
108             v  := y + a * Sqrt(3) / 24;
109             WITH NewV DO
110         BEGIN
```

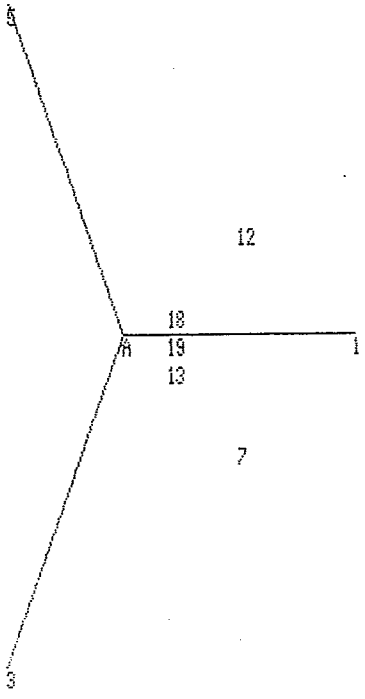
```
111         colour := '13';
112         x_co   := u;
113         y_co   := v;
114       END;
115       i13 := NewV;
116       WITH NewV DO
117       BEGIN
118         colour := '19';
119         x_co   := x + a / 5;
120         y_co   := y;
121       END;
122       i19 := NewV;
123
124       e1.source := i1;
125       e1.sink   := i7;
126
127       e2.source := i7;
128       e2.sink   := i0;
129
130       e3.source := i7;
131       e3.sink   := i13;
132
133       e4.source := i13;
134       e4.sink   := i19;
135     END
136     ELSE Write(chr(7));
137   END;
138
139   (*
140   Var
141     x,y,u,v : REAL;
142
143     begin
144     x := i0.x_co;
145     y := i0.y_co;
146     if (True)
147        then begin
148             With NewV do
149               begin
150               colour := 'B';
151               x_co   := x + a / 2;
152               y_co   := y + a * Sqrt(3) / 6;
153               end;
154             i7 := NewV;
155             u  := x + a / 5;
156             v  := y + a * Sqrt(3) / 24;
157             With NewV do
158               begin
159               colour := 'C';
160               x_co   := u;
161               y_co   := v;
162               end;
163             i13 := NewV;
164             With NewV do
165               begin
```

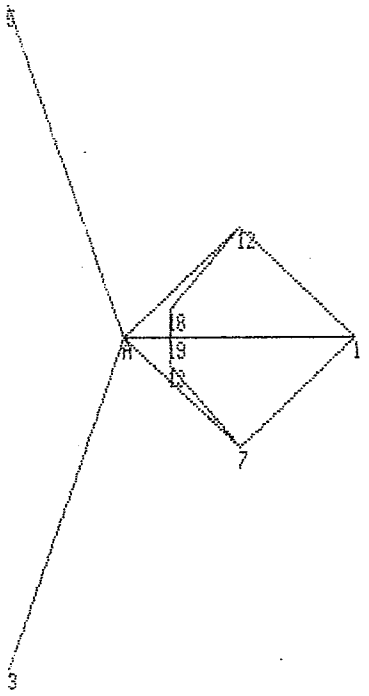
```
166                   colour := 'D';
167                   x_co   := x + a / 5;
168                   y_co   := y;
169                   end;
170                 i19 := NewV;
171
172                 end
173           else Write(chr(7));
174       end;
175     *)
176   PROCEDURE Generate;
177   BEGIN
178
179       SetColor(EGAWhite);
180     (*
181       OutTextXY(Round(V7.x_co), Round(V7.y_co), V7.colour);
182       OutTextXY(Round(V13.x_co),Round(V13.y_co),V13.colour);
183       OutTextXY(Round(V19.x_co),Round(V19.y_co),V19.colour);
184       OutTextXY(Round(V12.x_co),Round(V12.y_co),V12.colour);
185       OutTextXY(Round(V18.x_co),Round(V18.y_co),V18.colour);
186     *)
187       SetColor(EGAYellow);
188
189       e1.source := V1;
190       e1.sink   := V7;
191
192       e2.source := V7;
193       e2.sink   := V0;
194
195       e3.source := V7;
196       e3.sink   := V13;
197
198       e4.source := V13;
199       e4.sink   := V19;
200
201       ee1.source := v1;
202       ee1.sink   := v12;
203
204       ee2.source := v12;
205       ee2.sink   := v0;
206
207       ee3.source := v12;
208       ee3.sink   := v18;
209
210       ee4.source := v18;
211       ee4.sink   := v19;
212
213       Linha(Round(E1.source.x_co),Round(E1.source.y_co),
214       Round(E1.sink.x_co),  Round(E1.sink.y_co));
215       Linha(Round(E2.source.x_co),Round(E2.source.y_co),
216       Round(E2.sink.x_co),  Round(E2.sink.y_co));
217       Linha(Round(E3.source.x_co),Round(E3.source.y_co),
218       Round(E3.sink.x_co),  Round(E3.sink.y_co));
219       Linha(Round(E4.source.x_co),Round(E4.source.y_co),
220       Round(E4.sink.x_co),  Round(E4.sink.y_co));
```

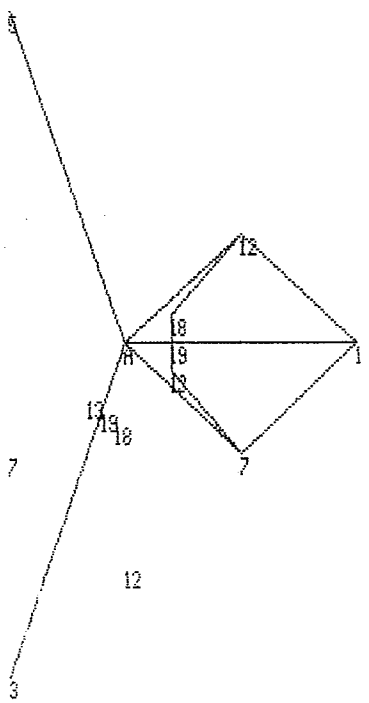
```
221         Linha(Round(EE1.source.x_co),Round(EE1.source.y_co),
222         Round(EE1.sink.x_co),  Round(EE1.sink.y_co));
223         Linha(Round(EE2.source.x_co),Round(EE2.source.y_co),
224         Round(EE2.sink.x_co),  Round(EE2.sink.y_co));
225         Linha(Round(EE3.source.x_co),Round(EE3.source.y_co),
226         Round(EE3.sink.x_co),  Round(EE3.sink.y_co));
227         Linha(Round(EE4.source.x_co),Round(EE4.source.y_co),
228         Round(EE4.sink.x_co),  Round(EE4.sink.y_co));
229     END;
230
231 PROCEDURE Mirror(VAR V0,V1,V7,V13,V19 : VERTEX; VAR V12,V18 :
      VERTEX);
232 BEGIN
233         V12.colour := '12';
234         V12.x_co := V7.x_co;
235         V12.y_co := (- V7.y_co + V0.y_co) + V0.y_co;
236
237         V18.colour := '18';
238         V18.x_co := V13.x_co;
239         V18.y_co := (- V13.y_co + V0.y_co) + V0.y_co;
240
241 END;
242
243     BEGIN
244     GraphDriver := DETECT;
245     InitGraph(GraphDriver, GraphMode, '');
246     GetAspectRatio(Xasp, Yasp);
247     SetTextStyle(SmallFont, HorizDir, 4);
248
249     SetViewPort(120,75,639,349, ClipOff);
250
251     a := 100.0;
252
253     FOR ii := 0 TO 4 DO
254 BEGIN
255         V0.colour := 'A'; V0.x_co := ii*a;     V0.y_co := 0.;
256         Generate_P1(V0,V1,V3,V5,E,EE,EEE);
257         Generate_P2(V0,V1,E, a, V7,V13,V19, E1,E2,E3,E4);
258         Mirror(V0,V1,V7,V13,V19, V12,V18);
259         Generate;
260
261         Roda(V0, 120, V1); (* V3 *)
262         Roda(V0, 120, V7);
263         Roda(V0, 120, V12);
264         Roda(V0, 120, V13);
265         Roda(V0, 120, V18);
266         Roda(V0, 120, V19);
267         Generate;
268
269         Roda(V0, 120, V1); (* V5 *)
270         Roda(V0, 120, V7);
271         Roda(V0, 120, V12);
272         Roda(V0, 120, V13);
273         Roda(V0, 120, V18);
274         Roda(V0, 120, V19);
```

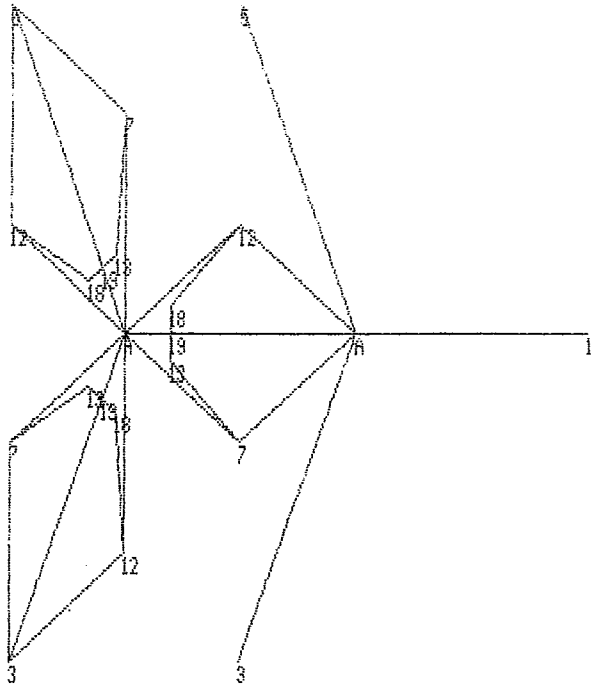
```
275         Generate;
276     END;
277
278     V0 := V3;
279
280     FOR ii := 0 TO 4 DO
281     BEGIN
282
283         Generate_P1(V0,V1,V3,V5,E,EE,EEE);
284         Generate_P2(V0,V1,E, a, V7,V13,V19, E1,E2,E3,E4);
285         Mirror(V0,V1,V7,V13,V19, V12,V18);
286         Generate;
287
288         Roda(V0, 120, V1); (* V3 *)
289         Roda(V0, 120, V7);
290         Roda(V0, 120, V12);
291         Roda(V0, 120, V13);
292         Roda(V0, 120, V18);
293         Roda(V0, 120, V19);
294         Generate;
295
296         Roda(V0, 120, V1); (* V5 *)
297         Roda(V0, 120, V7);
298         Roda(V0, 120, V12);
299         Roda(V0, 120, V13);
300         Roda(V0, 120, V18);
301         Roda(V0, 120, V19);
302         Generate;
303         V0.colour := 'A'; V0.x_co := V0.x_co - a;
304     END;
305
306     V0 := V3;
307
308     FOR ii := 0 TO 4 DO
309     BEGIN
310         V0.colour := 'A'; V0.x_co := V0.x_co + a;
311         Generate_P1(V0,V1,V3,V5,E,EE,EEE);
312         Generate_P2(V0,V1,E, a, V7,V13,V19, E1,E2,E3,E4);
313         Mirror(V0,V1,V7,V13,V19, V12,V18);
314         Generate;
315
316         Roda(V0, 120, V1); (* V3 *)
317         Roda(V0, 120, V7);
318         Roda(V0, 120, V12);
319         Roda(V0, 120, V13);
320         Roda(V0, 120, V18);
321         Roda(V0, 120, V19);
322         Generate;
323
324         Roda(V0, 120, V1); (* V5 *)
325         Roda(V0, 120, V7);
326         Roda(V0, 120, V12);
327         Roda(V0, 120, V13);
328         Roda(V0, 120, V18);
329         Roda(V0, 120, V19);
330         Generate;
331     END;
332
333     ch := ReadKey;
334     CloseGraph;
335     END.
```

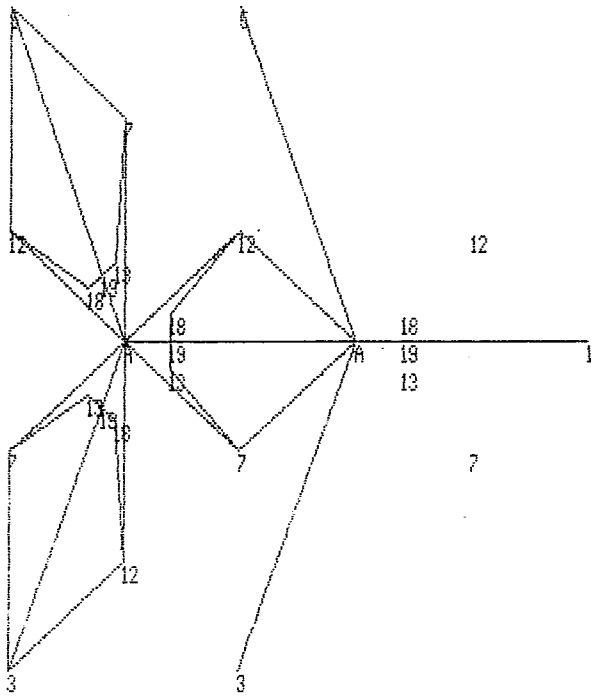

5
A
1
3


5
12
18
A
19
1
13
7
3


12
8
9
10
1
7
3


12
8
9
10
1
7
7
12
3


5
12
18
19
13
1
7
12
3
3


5
12
12
18
18
19
19
13
1
7
7
12
3
3