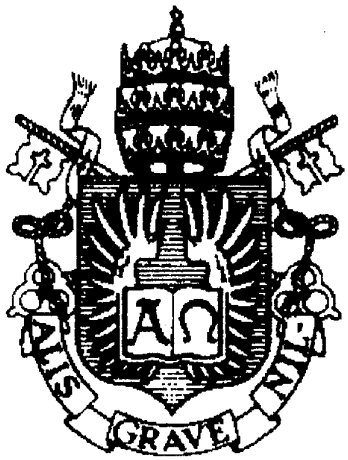
PUC



ISSN 0103-9741

Monografias em Ciência da Computação
nº 36/94

Fundamentos de Projeto para uma Linguagem Visual de Programação Estruturada

Clarisse Sieckenius de Souza
Deller James Ferreira

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, N° 36/94

Editor: Carlos J. P. Lucena Dezembro, 1994

Fundamentos de Projeto para uma Linguagem Visual de Programação Estruturada *

Clarisse Sieckenius de Souza

Deller James Ferreira

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br

Fundamentos de Projeto para uma Linguagem Visual de Programação Estruturada

Clarisse Sieckenius de Sousa
Deller James Ferreira

ICAD
Departamento de Informática - PUC-Rio
Email: clarisse@icad.puc-rio.br e deller@inf.puc-rio.br

Abstract

This work proposes to create a Visual Programming Language, where control structures are visually represented by geometric forms suggestive of the data flow control. Such geometric forms are formalized according to L-systems rewrintig rules. L-systems support other kinds of grammars due to generality and simplicity - they allow for rules to be applied in paralel and for easy mapping from syntax on to visual representations. By associating L-systems to the so called "turtle geometry" control structures can be visualized. Moreover L-systems provide for a syntactic formalization necessary to every programming language. The proposed Visual Programming Language is based on Pascal and called Lvpascal.

Resumo

O presente trabalho se propõe a criar uma Linguagem de Programação Visual, onde as estruturas de controle sejam representadas visualmente segundo formas geométricas que sugiram o controle, sobre o fluxo de dados, de cada estrutura. Tais formas geométricas possuem uma formalização associada, os L-systems. Os L-systems se destacam sobre os vários tipos de gramáticas por sua generalidade, permitindo que regras sejam aplicadas em paralelo e pela facilidade do mapeamento da sintaxe na representação visual através da geometria da tartaruga. Os L-systems associados à geometria da tartaruga são utilizados na visualização das estruturas de controle, além de proverem a formalização sintática necessária a qualquer linguagem de programação. A Linguagem Visual de Programação se baseia na linguagem PASCAL e é chamada de LvPASCAL.

I- Introdução

O primeiro passo do trabalho foi um estudo de vários tipos de gramáticas, incluindo o trabalho de Lilia Hess [1], buscando fundamentação teórica para o trabalho. A parte relativa a gramáticas de grafos encontra-se do item II.

Após estarmos inclinados na escolha dos L-systems como representação formal, realizamos um estudo dos trabalhos de Lindenmayer e Prusinkiewicz, contido no item III.

Finalmente, no item IV, foram elaboradas a repesentações visuais das estruturas de controle(contidas na linguagem pascal), de uma forma cognitiva, e construídos os L-systems equivalentes.

II - Gramáticas de Grafos

Gramáticas de Grafos são extensões das Gramáticas de Strings para grafos. Há duas abordagens para gramáticas de grafos, uma através das gramáticas de Chomsky(sistemas de reescrita sequenciais e outra baseando-se nos Lsystems(sistemas de reescrita paralelos).

II.1 - Gramáticas de Grafos Sequenciais

Em um passo de reescrita sequencial, ou seja, em um passo de derivação direta de uma gramática de grafo sequencial, um subgrafo de um grafo hospedeiro é substituido por outro. Dessa forma, uma produção de grafo é uma tripla p = (g_l, g_r, E), onde g_l é o subgrafo a ser substituído(lado esquerdo), g_r é o subgrafo a ser inserido(lado direito) e E é a função de Encaixe. A função de Encaixe é o principal critério de diferenciação entre as várias gramáticas de grafos.

II.1.1 - Função de Encaixe

A primeira definição formal do problema de Encaixe foi formulada por Schneider:

Seja p = (g_l, g_r, E) uma produção

Sejam In(g_l, g) e Out(g_l, g) o conjunto de todas as arestas com rótulo *i* indo para g_l de g_l - g e saindo de g_l para g_l - g, respectivamente.

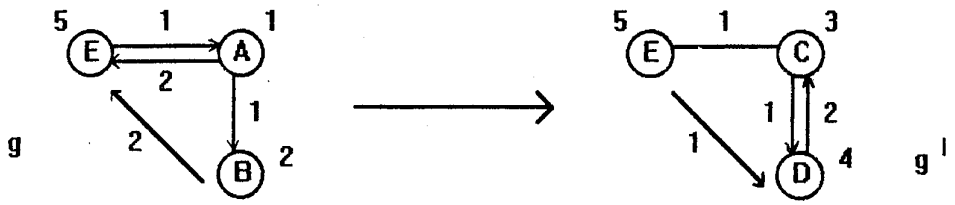
A função de encaixe E é definida como uma 2n-upla (π_{-n}, ...,π_{-1},π_1,...,π_n) com π_{-1},π_1 sendo a relação entre os conjuntos de nós k_l e k_r, ie, $\pi_{-1} \subseteq k_l \times k_r$, $\pi_1 \subseteq k_l \times k_r$, $1 \leq i \leq n$. O grafo g' a derivação direta de g, é definido como:

(g_l - g) $\cup$ g_r mais as seguintes arestas de encaixe

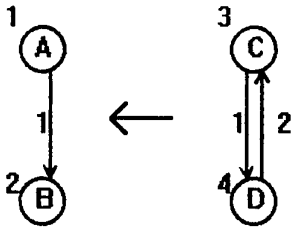
$In_i(g_r, g') \leftarrow \pi_{-i} \circ In_i(g_l, g)$

$Out_i(g_r, g') \leftarrow Out_i(g_l, g) \circ \pi_i^{-1}$

Exemplo:



Aplicando:



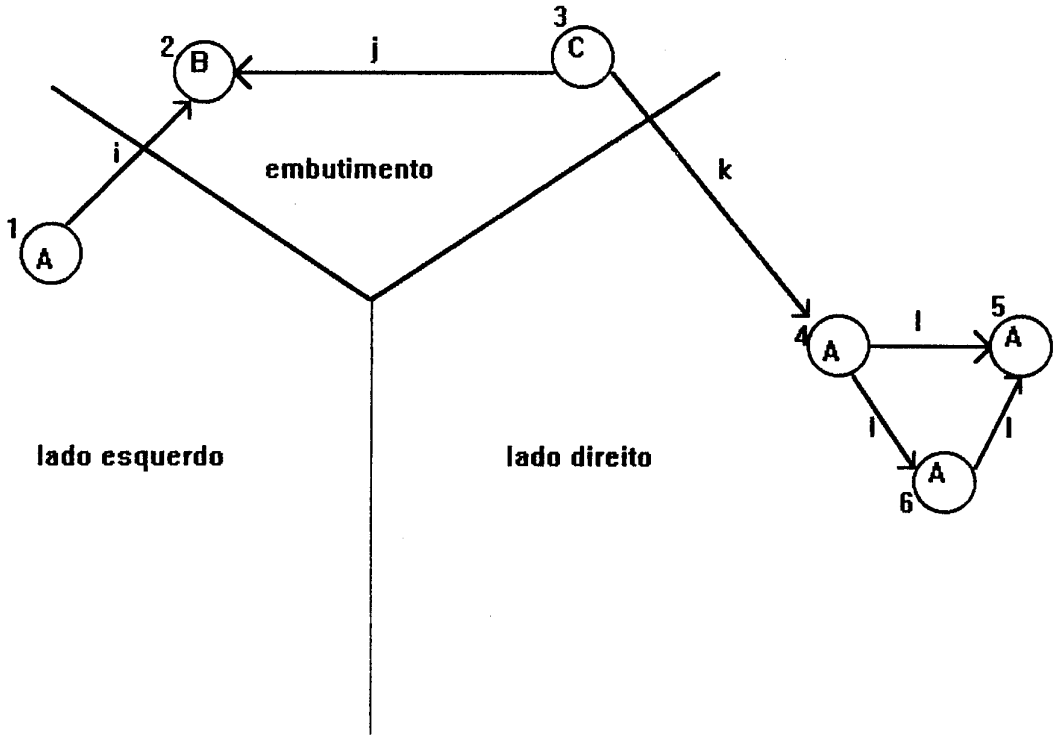
π-1 = {(1,3),(1,4)}

π1 = {(1,4),(2,4)}

$\pi_2 = \pi_{-2} = \varnothing$

Uma outra forma de função de encaixe é apresentada por Göttler[4]:

Uma produção P é apresentada graficamente na figura abaixo.



A produção P da figura anterior é aplicada a um grafo hospedeiro H da seguinte forma:

- Procure um nó em H rotulado por A.

- Insira o lado direito de P.

- Comece pelo ponto onde estava o nó rotulado por A indo em direção de todas as arestas rotuladas por i. Se for encontrado:

(a) Nós rotulados pos B
Vá nas direções contrárias a aresta j e caso encontre

(b) Nós rotulados por C desenhe uma aresta rotulada por K desse nó ao nó 4 do lado direito da regra que foi inserido.

A terceira e última abordagem a ser discutida é a abordagem algébrica. A teoria algébrica de gramáticas é um método de descrever gramáticas de grafos usando morfismos de grafos e construções de colagem. Morfismos de grafos são mapeamentos preservando as relações entre os nós e as arestas. As contruções de colagem são uma generalização da operação de concatenação de strings e casos especiais de "pushouts", dessa forma técnicas de diagrama universal podem ser usadas na maioria das construções e provas. Infelizmente, conceitos de álgebra universal são desconhecidos à maioria dos pesquisadores de ciência da computação e biologia, os mais interessados em gramáticas de grafos.

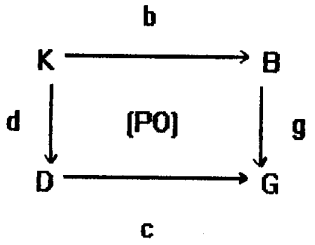
Fazendo uma analogia com a operação de concatenação de strings e a operação de colagem em grafos, consideremos uma produção de Chomsky da forma p = (u,v). Então, para cada contexto x e y obtemos uma derivação direta xuy => xvy para que u seja substituido por v. Mais precisamente, há uma substituição de u por v e uma conexão do primeiro símbolo v1 de v = v1...vs com o último símbolo xn de x = x1 ... xn e do u'timo vs de v com o primeiro y1 de y = y1 ... yt.

No caso de gramáticas de grafos u,v e (x,y) tornam-se grafos arbitrátios B_1, B_2 e D respectivamente. Precisamos considerar também um grafo K chamado de interface.

A string xuy corresponde a colagem de B_1 e D através de K e xvy a colagem de B_2 e D através de K. O que significa que uma derivação direta em grafos consiste de duas construções de colagem.

No caso especial em que K é exatamente a interseção de B e D, a colagem de B e D através de K é simplesmente a união de B e D. Em geral, temos a situação em que B,D e K são disjuntos e a conexão entre eles é dada por morfismos de grafos b : K -> B e d: K :->D.

A idéia principal da abordagem algébrica é a caracterização do conceito algébrico de *pushout* (PO). O *pushout* é a construção de colagem. Na colagem não é considerado apenas o grafo G, também consideramos a conexão com B e D através dos morfismos d: K :->D e c: D -> G em conjunto com os morfismos b : K -> B e g: B :->G de acordo com a figura abaixo.



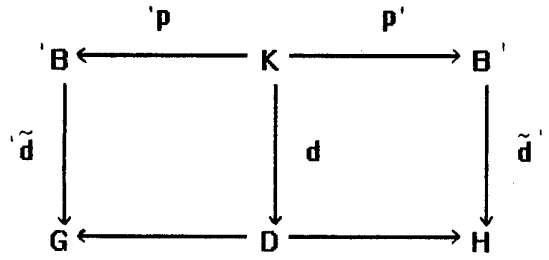
Uma gramática sequencial segundo a abordagem algébrica é definida a seguir.

Um grafo rotulado sobre um alfabeto de nós Σ_v e um alfabeto de rótulos de arestas Σ_e é definido por G = (V, E, q, mv,me), com V conjunto de nós , E conjunto de arestas (ambos finitos), mapeamentos q, z : E -> V (fonte e destino), e funções de rotulações mv : V -> Σ_v e me : V -> Σ_e. Uma produção é definida como:

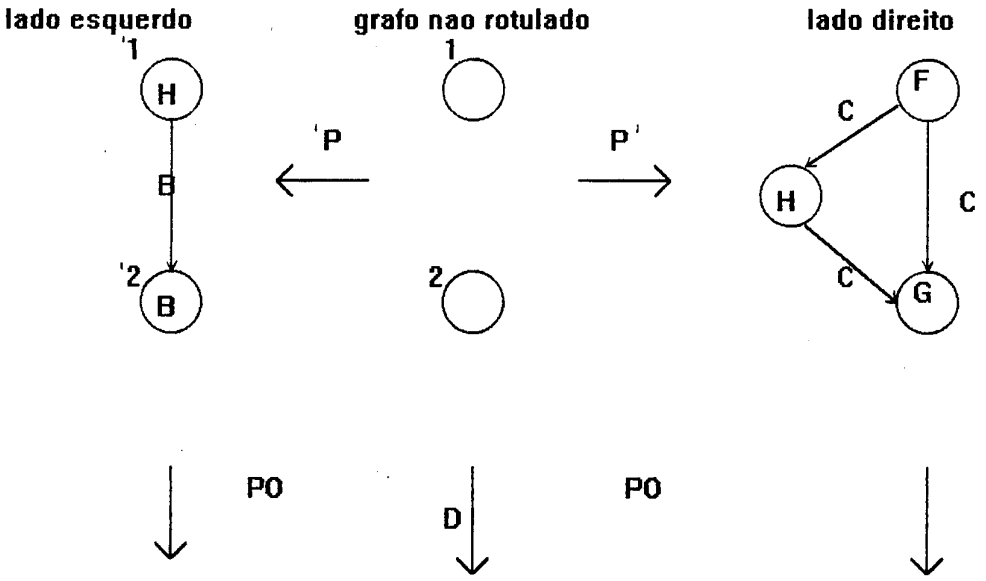
$$p = (\text{'B} \overset{\text{'p}}{\leftarrow} K \overset{\text{p'}}{\rightarrow} \text{B'})$$

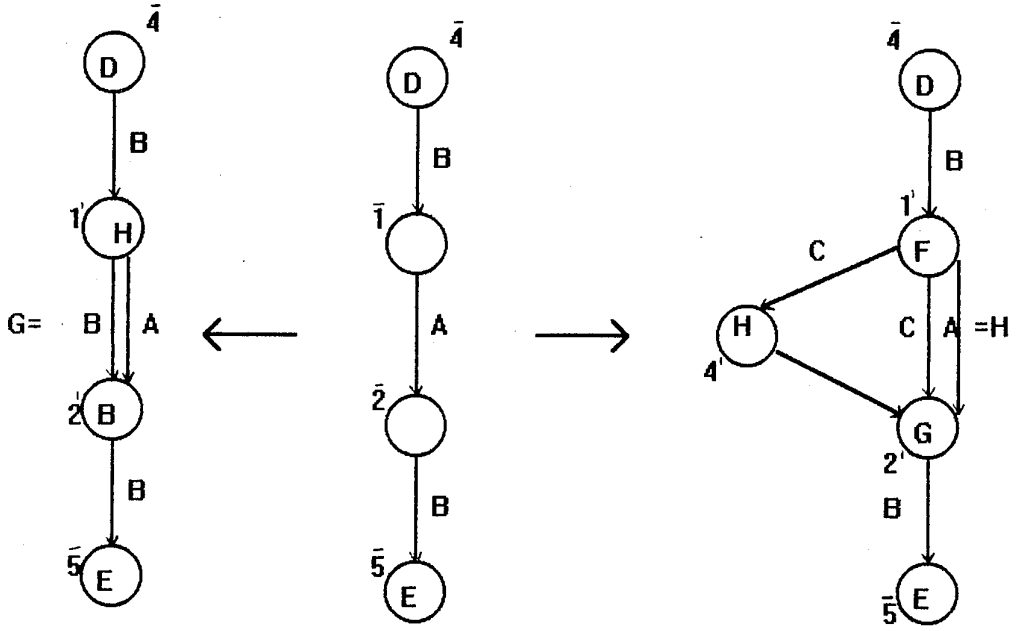
Onde 'B e B'são grafos rotulados, K um grafo não rotulado, 'p e p' morfismos. Um grafo H é uma derivação direta do grafo G se e somente se existe um morfismo de grafo injetivo d: K :->D(D é parcialmente rotulado), tal que o diagrama da figura abaixo comuta.

Um diagrama é comutativo se K->B->G = K->D->G.



Exemplo:


lado esquerdo
grafo nao rotulado
lado direito
'1
H
B
'2
B
'P
1
2
P'
F
C
H
C
C
G
PO
D
PO



grafo parcialmente rotulado

II.2 Gramáticas de Grafos paralelas

O mecanismo de reescrita nas derivações paralelas não é local, todo grafo hospedeiro é reescrito em um passo. Dessa forma as produções são aplicadas mais de uma vez em cada passo de derivação.

Sistemas de reescrita de grafos paralelos podem ser considerados como uma generalização dos L-systems para grafos rotulados, podendo ser chamados de graph L-systems.

III - L-systems

A aplicação de gramáticas na modelagem do desenvolvimento de organismos vivos levou a descoberta de uma família de gramáticas chamadas de L-systems. Os L-systems foram descobertos por Aristid Lindenmayer (1925-1989), com a proposta de descrever o crescimento de organismos multicelulares.

Através dos L-systems, o desenvolvimento de organismos multicelulares é modelado aplicando-se regras de substituição ou regras de reeescrita em pralelo a cada célula a cada passo no tempo. A reescrita pode ser livre de contexto(desenvolvimento sem interação) ou sensível ao contexto(desenvolvimento com interações). Tais sistemas diferem das gramáticas de Chomsky pelo fato que nenhuma distinção é feita entre símbolos terminais e não terminais, e a cada passo da derivação, as produções são aplicadas paralelamente aos símbolos.

Os L-systems são usados como ferramenta para modelagem de desenvolvimento celular uni-dimensional, bi-dimensional e tri-dimensional. Sua interpretação é também topológica, fornecendo relacionamentos de vizinhança entre as células. Particularmente representam bem as estruturas ramificadas das plantas controlando sua topologia.

Há dois tipos básicos de L-systems, um gerando uma linguagem de strings e outro uma linguagem de grafos.

III.1 L-systems como uma gramática de strings

Um L-systemsem interação(0L-system) é uma tripla G=(∑,P,ω), onde ∑ é um conjunto finito não vazio(da célula de estados) , P é um mapeamento de ∑ em ∑*, e ω é um elemento de ∑*. Escreve-se um elemento de P(uma produção da célula de estados), P (uma produção da célula de estados) como a->x, onde a ∈ ∑ e x ∈ ∑ *. Nós dizemos que a string x deriva diretamente da string y no 0L-system G e escrevemos x⇒y, se existe um inteiro n ≥ 1, existem os símbolos ai e strings pi, para 1≥i≥n, tal que G

$$x=a_1a_2\ldots a_n$$
$$y=p_1p_2\ p_n$$

E também para cada i , ai →pi é uma produção de G. Além disso, dizemos que uma string x deriva a string y em G, e escrevemos $\overset{*}{\underset{G}{\Rightarrow}}$ y , se existem strings q0,q1,...,qn para algum inteiro n≥0, tal que

$$q_0 \Rightarrow q_1 \Rightarrow q_2 \Rightarrow \ldots \Rightarrow q_n \quad , x=q_0 \text{ e } y=q_n.$$

A linguagem gerada por um 0L-system G = (∑,P,ω) , denotado como L(G) é:

$$L(G)=\{x/x \overset{*}{\underset{G}{\Rightarrow}} y\}.$$

A definição dos L-systems interativos(IL-systems) leva em consideração um certo número de vizinhos de esquerda e da direita da célula de estados e dos simbolos em ambs extremidades da string. Um Sistema é um (k,l) L-system se K os vizinhos da esquerda e L os vizinhos da direita são levados em conta na determinação da substituição de determinada célula, em adição ao estado da célula.

Um (K,L) L-system é uma quatro-upla $G=(\Sigma,P,g,\omega)$, onde Σ é um conjunto de símbolosfinito não vazio, P é um mapeamento de $(\Sigma \cup \{g\})^k \times \Sigma \times (\Sigma \cup \{g\})^l$ em Σ^*, g é um símbolo fora de Σ e $\omega \in \Sigma^*$. Os elementos de P(produções) são escritos na forma $(\omega 1,a,\omega 2) \rightarrow \omega 3$, onde $a \in \Sigma$, $\omega 1 \in (\Sigma \cup \{g\})^k$, $\omega 2 \in (\Sigma \cup \{g\})^l$ e $\omega 3 \in \Sigma^*$.

As strings $\omega 1$ e $\omega 2$ devem satisfazer as seguintes condições:

(i) Se $\omega 1 = \omega 1' g \omega 1''$ para algum $\omega 1'$ e $\omega 1'' \in (\Sigma \cup \{g\})^*$,

então $\omega 1' \in \{g^*\}$.

(ii) Se $\omega 2 = \omega 2' g \omega 2''$ para algum $\omega 2'$ e $\omega 2'' \in (\Sigma \cup \{g\})^*$,

então $\omega 2' \in \{g^*\}$.

III.2 L-systems como uma gramática de grafos

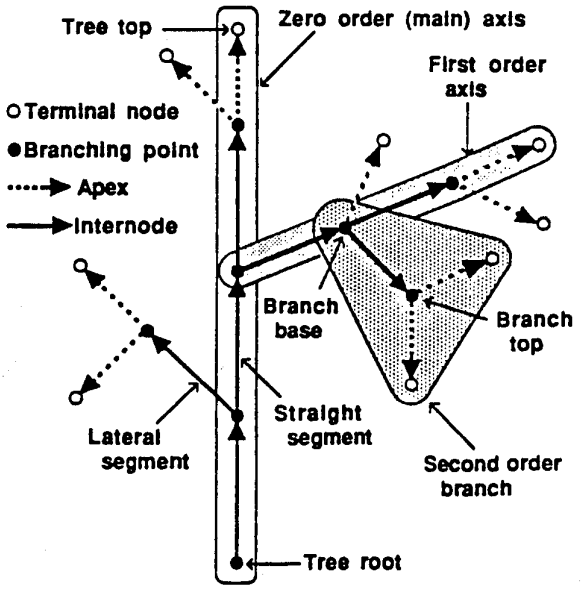
Os L-system segundo uma abordagem de grafos apresentados no trabalho são os Tree L-systems. Os Tree L-systems são um método de geração de estruturas ramificadas, que são representadas através de Axial Trees.

III.2.1 Axial Trees

Uma "Axial Tree"possui arestas rotuladas e direcionadas. No conceito biológico as aresta são ramificações. Um segmento seguido por pelo menos mais de um segmento é chamado de "internode". Um segmento terminal é chamado de "apex". O que também caracteriza uma "axial tree" é um segmento reto unindo alguns nós possuindo segmentos laterais. Uma sequência de segmentos é chamada de "axis"se:

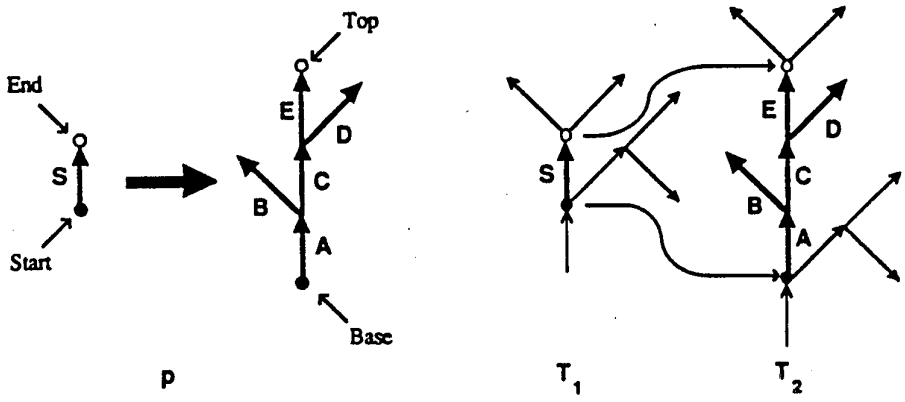
- o primeiro segmento na sequência é originado na raiz da árvore ou é um segmento lateral.

- cada segmento subsequente é um segmento reto.



III.2.2 Tree L-systems

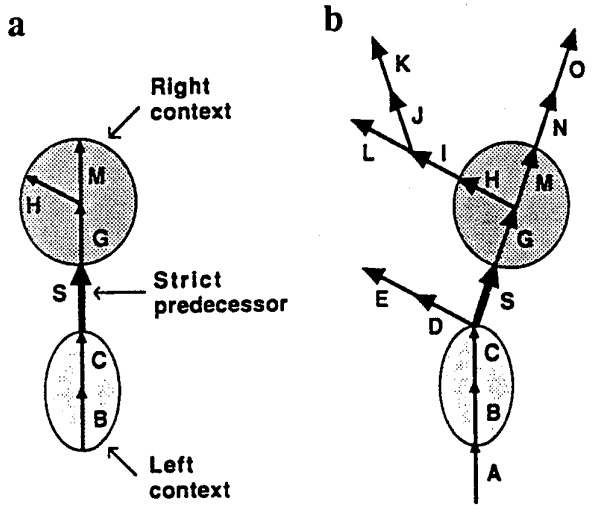
O processo de desenvolvimento das plantas se dá pela transformação de segmentos em estruturas mais complexas. Segundo a teoria de gramática de grafos o conceito correspondente é o de reescrita de aresta. Uma regra de reescrita de um Tree L-system, substitui uma aresta especificada como predecessor por uma Axial Tree chamada de sucessor, de talforma que o nó inicial do predecessor é identificado com o topo do sucessor de acordo com a figura abaixo:



No caso da reescrita livre de contexto, o rótulo da aresta a ser substituída determina a produção a ser aplicada.

Um "Tree 0L-system" G é especificado por três componentes: um conjunto de rótulos de arestas V, uma árvore inicial w com rótulos em V e um conjunto de produções ("tree productions"). Dado o L-system G, uma "axial tree" T2 é derivada diretamente de uma árvore T1 (T1⇒T2), se T2 é obtida de T1 pela aplicação simultânea de cada aresta em T1 pelo seu sucessor de acordo com o conjunto de produções P. Uma árvore T é gerada por G em uma derivação de tamanho n se existe uma sequência de árvores T0,T1,T2,...,Tn tal que T0 = W, Tn=T e T0⇒... ⇒Tn.

Uma produção sensível ao contexto requer contexto, ou seja, arestas vizinhas da aresta a ser substituída devem ser testadas. desta forma um predecessor de uma produção sesível ao contexto p consiste de três componentes: uma Axial Tree l chamada de contexto da esquerda, uma aresta S chamada de predecessor estrito e uma Axial Tree r chamada de contexto da direita. Uma produção p encaixa-se a uma dada ocorrência de uma aresta S em uma árvore T se l é um caminho em T terminando no nó inicial de S e r é uma subárvore de T originando-se no nó final de S. A produção pode ser aplicada pela substituição de S pela Axial Tree especificada como sucessor da produção como na figura abaixo:



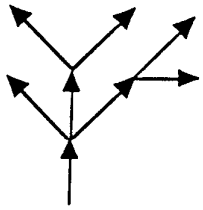
III.2.2.1 L-systems com colchetes

Uma das possibilidades de representação dos Tree L-systems é como uma string com colchetes. Dois símbolos são utilizados para determinar um ramo:

[- fornece a interpretação necessária para o início de um ramo.

] - fornece a marcação para o final de um ramo.

Ex:



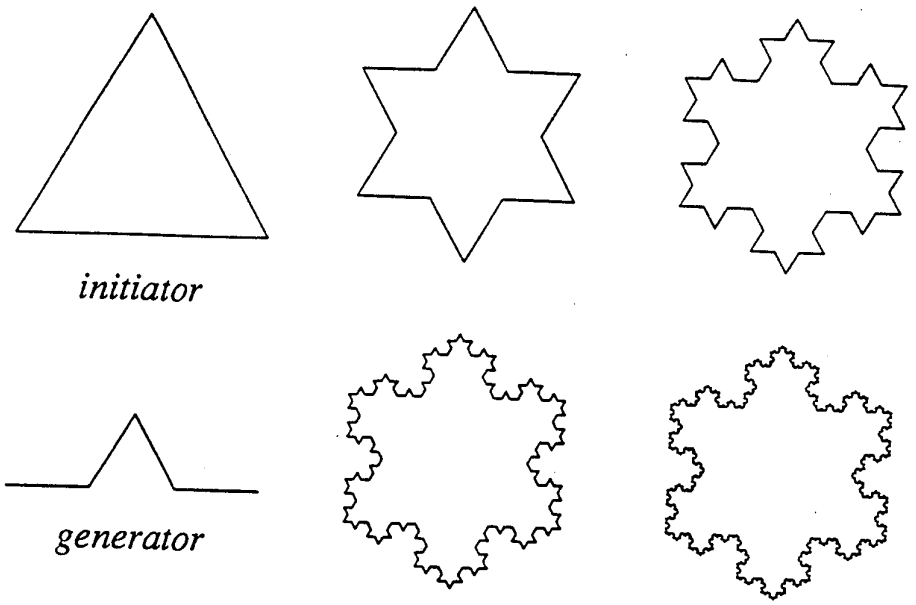
F[+F][-F[-F]F]F[+F][-F]

III.3 Reescrita de aresta

O método de reescrita de aresta usado na aplicação das produções é uma extensão das construções Koch. Em um Tree L-system a reescrita de aresta é a substituição de uma aresta por uma Axial Tree. Em um L-system de strings um caracter é substituído por uma string.

III.4 Construção Koch

A partir de duas formas, o "initiator"e o "generator", a cada estágio define-se n subdivisões iguais do "initiator" e substitui-se cada divisão por uma cópia do "generator".

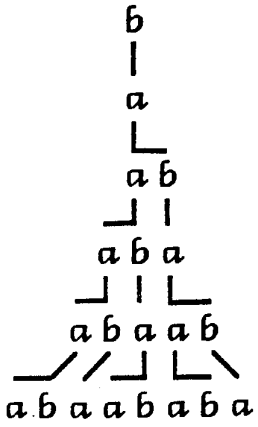


Exemplo de um L-system de strings:

axioma(initiator) : b

regras de reescrita(os sucesores das regras são generators) : a→ab
b→a

derivação: figura a seguir.



III.5 Interpretação geométrica da tartaruga

A teoria dos L-Systems está diretamente e exclusivamente relacionada com a topologia das entidades criadas. A interpretação gráfica e, portanto geométrica dos L-systems é um passo de certa forma fora da teoria. Vários autores propuseram interpretações.

O método explorado no trabalho e o mais difundido é o da interpretação dos L-Systems, formulado por Prusinkiewicz, através da geometria da tartaruga proposta por Abelson e diSessa[].

III.5.1 Geometria da tartaruga

A idéia básica da interpretação da tartaruga é dada a seguir. O estado da tartaruga é definido como uma tripla (x,y,α), onde as coordenadas cartesianas (x,y) representam a posição da tartaruga, e o ângulo α, que é o posicionamento da cabeça da tartaruga, sendo interpretado como a direção para onde a tartaruga está olhando. Dado um tamanho de passo d e um ângulo de incremento δ, a tartaruga pode responder aos comandos representados pelos seguintes símbolos:

F Mova-se para frente em um tamanho de passo d.
O estado da tartaruga altera-se para (x',y',α), onde $x'= x +d\cos\alpha$ e $y'=y+d\text{sen}\alpha$.
Um segmento de linha entre os pontos (x,y) e (x',y') é desenhado.

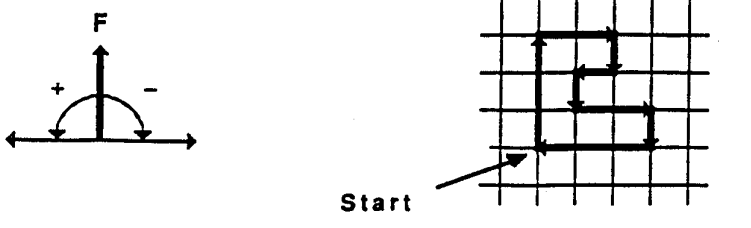
f Mova-se para frente em um paso de tamanho d sem desenhar a linha.

\+ Rode para a esquerda de acordo com o ângulo δ.
O próximo estado da tartaruga é$(x,y,\alpha+\delta)$.

\- Rode para a direita de acordo com o ângulo δ.
O próximo estado da tartaruga é$(x,y,\alpha-\delta)$.

Dada uma string v, o estado inicial da tartaruga $(xo,yo,\alpha o)$ e os parâmetros fixos d e δ, a interpretação da tartaruga é a figura(conjunto de linhas) desenhadas pela tartaruga em resposta a string v como na figura abaixo:

a **b**



FFF-FF-F-F+F+FF-F-FFF

III.5.1.1 Processo de ramificação

Com relação aos Tree L-systems há a necessidade da introdução dos símbolos [e] , cuja interpretação é a seguinte:

[Coloca estado corrente da tartaruga em uma pilha. A informação salva na pilha contém a posição e orientação da tartaruga.

] Retira um estado da pilha e o faz o estado corrente. Nenhuma linha é desenhada, embora a posição da tartaruga se altere.

III.5.1.2 Modificadores de âgulo e escala

Às vezes tona-se necessário o uso de modificadores do ângulo e da escala do traçado.

VI Criação das representações visuais das estruturas de controle de LvPASCAL

A forma utilizada na criação das formas geométricas que representam visualmente as estruturas de controle não segue nenhume metodologia, mas sim uma motivação cognitiva e perceptiva. A seguir será discutida cada representação visual proposta e apresentada sua especificação formal que chamaremos de parser visual.

Há dois tipos de representação para as estruturas de controle. Uma que pretente ser mais geral , e consequentemente mais abstrata e outra mais específica ,e portanto menos abstrata.

VI.1 Elaboração das estruturas de controle

1) FOR

O comando for é uma iteração de n passos, sendo suas características a repetição e o retorno do controle ao ponto inicial formando um LOOP. A representação abstrata escolhida é um círculo representando o LOOP, mas que não deixa claro o número de iterações. A representação menos abstrata é um polígono regular de n lados.

Obs: Os casos nos quais n=1 ou n=2 são particulares e optou-se por conservar uma noção mais geométrica, sobrepujando o conceito de LOOP.

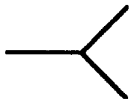
2) DO While e Repeat Until

A estrutura Do While caracteriza-se por um teste booleano seguido das instruções e a estrutura Repeat Until por um teste booleano após as instruções, portanto foram introduzidos os testes, representados como segmentos de reta verticais antes e depois do LOOP(estrutura análoga ao for)respectivamente.

Do While **Repeat Until**

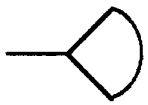
3)IF

O comando if é representado por uma estrutura ramificada, devido a sua característica de desvio do fluxo para uma de duas direções.



4)CASE

O comando case é uma forquilha, pois o fluxo é desviado em uma de várias direções. Sua representação mais abstrata é um leque.



5) PROGRAM

O programa(program) é uma estrutura modular que tudo contém como uma caixa, daí sua repesentação por um quadrado.

Obs: A estrutura program não é uma estrutura de controle e possui características intrínsecas. Mas não podemos deixá-la para um tratamento posterior , por ser indispensável.

6)Procedure

A chamada de um procedimento é o desvio para um trecho comparável a um programa de pequenas dimensões, logo a representação é um quadrado pequeno pendurado.

7)Simple Statement

A característica principal do comando simples é a ausência de subcomponentes, portanto uma representação que descrevesse um único elemento foi escolhida. Há implícito o conceito de sequêcia. A representação é um traço horizontal.

8)Recursion

A recursão é uma estrutura que referencia a si própria. Achamos que uma figura auto similar seria adequada, portando escolhemos uma espiral.

IV.2 Parser Visual

Ao escolhermos um botão contendo a representação visual de uma estrutura de controle, são disparados dois procedimentos, um que exibe o texto em pascal equivalente e outro que chama o L-system correspondente.

L-systems são a formalização das representações visuais, sendo a visualização orientada por regras. Um L-system é usado para construir o feedbak visual de cada botão.

Os L-systems são utilizados na especificação das estruturas de controle, levando-se em consideração o tamanho do traçado, controle do estado e de ângulo.

1) Program

O programa é representado por um quadrado preenchido. Como todas as estruturas de controle possuem características puramente geométricas, a estrutura program é representada por um L-system que faz o preenchimento do espaço limitado pelo quadrado.

axioma - A
regra - A -> +FA+FA+FA+FA
regra - F ->)F
ângulo - 90
tamanho de F inicial - tamanho do traçado do programa
parada - ordem = (tamanho do traçado/2)-2
incremento escalar - 1/2

2)Procedure

axioma - FF[FF]+FF+f-ffA
regra - A->+HA+HA+HA+HA
regra - H->)F
regra - H->)H
ângulo - 90
tamanho F - tamanho do traçado/4
parada - (tamanho de F/2) -2
incremento escalar - 1/2

3)Repeat Until
axioma - [⟩A]C) - B
regra - A->F+A
regra - B->FF
regra - C->f
incremento escalar - cos (179/2)
incremento angular - 89
ângulo de rotação - 1
tamanho de F - tamanho do traçado/2
parada - número de lados

4)IF
axioma - AB
regra - A → ⟩F
regra - B → [+⟩F][-⟩F]
ângulo de rotação - 45°
incremento escalar - 1/2
tamanho de F - tamanho do traçado

5)Case
axioma - ABC
regra - A → ⟩F(para n par) ou regra - A → ⟩F[⟩F](para n+1 ímpar)
regra - B →[-D][-B]
regra - C →[+D][+C]
regra - D -> ⟩F
parada - n/2(n é o número de casos)
ângulo de rotação - 90°/(n-1)(para n e n+1, n > 2)
incremento escalar - 1/2
tamanho de F - tamanho do traçado

6) Simple Statement
axioma - A
regra - A → F
tamanho de F - tamanho do traçado

7)For

axioma - A
regra - A → F + A
parada - n (número de lados)
ângulo de rotação - 1
tamanho de F - tamanho do traçado x cos(179/2)

8) Do While
axioma1 - +F][-F]([〉A]
regra1 - A → F + A
parada - n(número de lados)
ângulo de rotação - 1
incremento angular - 89
tamanho de F - tamanho do traçado/2
incremento escalar - cos(179/2)

9)Recursion

axioma - A
regra - A-> -F -- 〉A
parada - número de chamadas
ângulo de rotação - 45
incremento escalar - 2/3
tamanho de F - tamanho do traçado

Adcionando controle de estado

O controle de estado tem por por objetivo permititr a continuidade do fluxo do programa, ou seja, permite que as estruturas de controle sejam posicionadas de forma sequencial.

É importante dizer que há duas visualizações das estruturas de controle, uma visualização estática e uma visualização dinâmica. A visualização estática é a repesentação visual do código do programa. A visualização dinâmica procura mostrar o fluxo dos dados do programa em execução.

No trabalho, até o momento, não ficou clara a visualização dinâmica, portanto a discussão se restringe a visualização estática.

A representação das estruturas de controle que formam o código do programa é estruturada por meio de uma visão top-down, os sucessivos refinamentos do código são visualizados em planos diferentes; os níveis de detalhamento do programa são apresentados por uma interface 2 1/2 D.

Para cada plano de visualização devemos incluir no L-system de cada estrutura um controle de estado. Esse controle é realizado por meio de uma convenção do estado no início e no fim de cada estrutura e implementado com o uso de colchetes(pilha de estados).

A convenção é expressada no exemplo abaixo:

qi=(x,y,0)
qf=(x+tamanho do traçado,y,0)

Logo, para cada estrutura de controle, deve-se incorporar ao L-system equivalente os seguintes controles de estado:

1) Program

[L-system1]f, onde o tamanho de f é igual ao tamanho do traçado do programa.

Obs: Para o restante das estruturas to tamanho de f = tamanho do traçado.

2)Procedure

〉f[L-system2]f〉f
incremento escalar - 1/tamanho do traçado

3)Repeat Until

[+L-system3]f

4) If

〉f[L-system4]f

incremento angular = 1/tamanho do traçado

5)Case

〉f[L-system5]f

incremento angular = 1/tamanho do traçado

6)Single Statement

〉f[L-system6]〉f

incremento angular = 1/tamanho do traçado

7)For

[+L-system7]f
ângulo de rotação = 90

8)Do While

[L-system8]f

9) Recursion

[〉fL-system6]f

incremento escalar - $\sqrt{2}/18$

Referências

1. Prusinkiewics, P. and Lindenmayer, A.'1990 *The Algorithmic Beauty of Plants*
   Springer Verlag

2. Abelson, H. and diSessa, A. A.'1982 *Turtle Geometry*
   MIT Press Series

3. Chang'1990 *Principles of Visual Programming Systems*
   SHI-KUO Chang Editor

4. Shu,N. C. '1988 *Visual Programming*
   Van Nostrand Reinhold Company Inc.

5. de Souza, C. S. '1993 *Regularidade e Generalização em Interfaces Gráficas*
   Anais do SIBGRAPI VI
   Recife, Pe. pp 183-191

6. de Souza, C.S.'1994 *Structure and Expressiveness of Visual Computer Languages.*
   Aguardando publicação

7. Hofstadler, D.'1979 *Gödel, Escher and Bach: an Eternal Golden Braid*
   New York, Basic Books