

31006-8

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM SISTEMA DE MODELAMENTO  
DE OBJETOS FUNCIONAIS

por

*Glaucius Décio Duarte*

Dissertação submetida como requisito parcial para  
a obtenção do grau de Mestre em  
Ciência da Computação

*Prof. Anatólio Laschuk*  
orientador

*Prof. Benamy Turkienicz*  
co-orientador

Porto Alegre, maio de 1990.



SABi



UFRGS

05221931

UFRGS  
INSTITUTO DE INFORMÁTICA  
BIBLIOTECA

CATALOGAÇÃO NA FONTE

Duarte, Glaucius Décio

Um sistema de modelamento de objetos funcionais. Porto Alegre, PGCC da UFRGS, 1990.

1 v.

Diss. (mestr. ci. comp.) UFRGS-PGCC, Porto Alegre, BR-RS, 1990.

Dissertação: Computação Gráfica: Geração de  
Objetos Funcionais: Geração de Objetos Tri-  
dimensionais

UFRGS  
INSTITUTO DE INFORMÁTICA  
BIBLIOTECA

### AGRADECIMENTOS

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo auxílio financeiro, recebido em forma de bolsa, durante o curso de Pós-Graduação, e ao corpo docente do Curso de Pós-Graduação em Ciência da Computação da UFRGS, em especial ao meu orientador, *Anatólio Laschuk*, ao meu co-orientador *Benamy Turkienicz* (prof. do Departamento de Urbanismo da Faculdade de Arquitetura da UFRGS), e à *profa. Carla Maria Dal Sasso Freitas* pelo apoio e pelas sugestões valiosas que muito me auxiliaram na elaboração deste trabalho.

## SUMARIO

SUMÁRIO .....	04
LISTA DE ABREVIATURAS .....	07
LISTA DE FIGURAS .....	08
LISTA DE TABELAS .....	11
RESUMO .....	12
ABSTRACT .....	13
1. INTRODUÇÃO .....	14
1.1 <u>Utilização de regras de "shape grammars" na análise e criação de projetos</u> .....	19
1.2 <u>Geração de objetos tridimensionais a partir da sugestão de utilização de regras de "shape grammars"</u> .....	23
2. UTILIZAÇÃO DE REGRAS NA CRIAÇÃO DE VOLUMES .....	27
3. FORMALIZANDO A GERAÇÃO DE VOLUMES .....	35
3.1 <u>Volume padrão</u> .....	35
3.2 <u>Volume inicial</u> .....	36
3.3 <u>Aplicando operações de transformação para gerar os diversos estágios do processo gerativo</u> .....	36
4. CRIANDO UM MECANISMO DE REPETIÇÃO PARA ACELERAR O PROCESSO DE GERAÇÃO .....	41
5. CRIANDO MECANISMOS ALTERADORES PARA AUMENTAR AS ALTERNATIVAS DE PROJETO .....	44

5.1	<u>Alterando as dimensões de um volume</u> .....	45
5.2	<u>Deslocando um volume para outra posição</u> .....	49
5.3	<u>Trocando o volume de referência</u> .....	52
5.4	<u>Alterando a textura e a cor das faces de um volume</u> .....	53
5.5	<u>Eliminação de um volume</u> .....	54
5.6	<u>Deformação de um volume existente</u> .....	55
5.7	<u>Rotação de um volume</u> .....	58
5.8	<u>Individualizando volumes de mesmo nome, para permitir alterações somente em alguns deles</u> .....	59
5.9	<u>Alterando o nome de um volume</u> .....	62
6.	ACOPLAMENTO DE UNIVERSOS .....	64
7.	UTILIZAÇÃO DE REGRAS DE GERAÇÃO PARA AGILIZAR OS ESTÁGIOS DO PROCESSO PROJETIVO .....	71
7.1	<u>Geração de um vocabulário de elementos</u> .....	72
7.1.1	A linguagem de ações .....	73
7.1.2	O editor de vocabulário .....	74
7.1.2.1	O editor de ícones .....	75
7.1.2.2	Utilização de um procedimento gráfico-interativo para acompanhar o processo de edição .....	77
7.2	<u>Incorporando um mecanismo para entrada de parâmetros</u> .....	78
7.3	<u>Um mecanismo para operações aritméticas</u> .....	80
7.4	<u>Atribuição</u> .....	82
7.5	<u>Exibição de mensagens</u> .....	84
7.6	<u>Variáveis auxiliares</u> .....	86
7.7	<u>Observações e comentários</u> .....	89
8.	MECANISMO DE CONTROLE .....	91
9.	RELAÇÕES FUNCIONAIS .....	101
9.1	<u>Como definir a estrutura funcional de um projeto</u> .....	101

9.2	<u>Criando classes de objetos de projeto</u> .....	104
9.2.1	Utilizando uma estrutura de dados relacio- nal para representar os objetos de um pro- jeto .....	105
9.2.2	Geração de classes de objetos utilizando relações .....	108
9.2.3	O Mecanismo para criação de uma classe ...	112
9.3	<u>Representação das ocorrências de uma classe</u> ....	114
9.4	<u>Incorporando os relacionamentos funcionais</u> .....	116
9.4.1	Inerências .....	120
9.4.2	Geração dos relacionamentos .....	123
9.5	<u>Mecanismos para leitura de atributos</u> .....	125
9.6	<u>Realizando alterações nas relações de entidade</u> .	126
9.7	<u>Eliminação de uma entidade</u> .....	129
9.8	<u>Eliminação de relacionamentos</u> .....	130
9.9	<u>Leitura dos relacionamentos de uma entidade</u> ....	131
9.10	<u>Mecanismo que fornece o nome da entidade que possui um determinado conjunto de atributos</u> ...	133
9.11	<u>Transferência do fluxo de ações</u> .....	134
10.	APLICAÇÕES .....	141
10.1	<u>Biologia</u> .....	141
10.2	<u>Química</u> .....	147
10.3	<u>Matemática</u> .....	150
11.	CONCLUSÕES .....	156
	ANEXOS - Exemplos de objetos gerados pelo SMOF .....	161
	BIBLIOGRAFIA .....	172
	OUTRAS FONTES .....	175

## LISTA DE ABREVIATURAS

CAD	Computer Aided Design
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
Cont.	Continuação
Deslocam.	Deslocamento
G+	Geminação positiva
G-	Geminação negativa
Identif.	Identificação
J+	Justaposição positiva
J-	Justaposição negativa
Mod.	Módulo
Quant.	Quantidade
S+	Superposição positiva
S-	Superposição negativa
UFRGS	Universidade Federal do Rio Grande do Sul

## LISTA DE FIGURAS

Figura 1.1 - Condição inicial de uma organização espacial .....	21
Figura 1.2 - Regras de uma gramática de layout .....	22
Figura 1.3 - Derivação de layouts pela aplicação da Regra 1 da Figura 1.2 .....	23
Figura 2.1 - Volume inicial .....	27
Figura 2.2 - Geminação positiva .....	29
Figura 2.3 - Geminação negativa .....	30
Figura 2.4 - Justaposição positiva .....	30
Figura 2.5 - Justaposição negativa .....	30
Figura 2.6 - Superposição positiva .....	30
Figura 2.7 - Superposição negativa .....	31
Figura 2.8 - Nomenclatura dos vértices utilizados como referência nas operações do tipo geminação positiva .....	32
Figura 2.9 - Quatro tipos de geminação positiva (vista lateral) .....	32
Figura 2.10 - Perspectiva da Figura 2.9(a) / geminação positiva com referencial em G1 .....	33
Figura 2.11 - Caso da Figura 2.9(a) quando as dimensões do volume a ser criado forem maiores que as da semente .....	33
Figura 2.12 - Perspectiva da Figura 2.11 .....	33
Figura 3.1 - Resultado do Exemplo 3.1 .....	39
Figura 4.1 - Resultado do Exemplo 4.1 .....	43
Figura 5.1 - Ilustração esquemática do Exemplo 5.1 .....	47
Figura 5.2 - Perspectiva do projeto gerado pela aplicação dos passos definidos no Exemplo 5.1 ...	48
Figura 5.3 - Ilustração esquemática do Exemplo 5.2 .....	51



Figura 5.4 - Perspectiva do projeto final do Exemplo 5.2 .....	52
Figura 5.5 - Deformação de um paralelepípedo .....	55
Figura 5.6 - Códigos das arestas em um volume .....	56
Figura 5.7 - Duas alternativas de um projeto .....	58
Figura 5.8 - Quatro volumes gerados com o mesmo nome (V) .....	59
Figura 5.9 - Uma alteração em um dos volumes individualizados .....	61
Figura 7.1 - Exemplo da definição de um ícone para representar o elemento janela definido no Exemplo 6.1 .....	76
Figura 7.2 - Matriz bidimensional de volumes, gerada a partir de uma seqüência de ações, com utilização do mecanismo de entrada de parâmetros .....	80
Figura 9.1 - Uma relação de materiais de uma instalação sanitária .....	106
Figura 9.2 - Elementos de uma estrutura de edifício ...	109
Figura 9.3 - Independência entre o sistema principal e os dados funcionais .....	112
Figura 9.4 - Tipos de relacionamento .....	118
Figura 9.5 - Esquema diagramático para representação de relacionamentos com uma visão de refinamento funcional .....	120
Figura 9.6 - Inerências entre as classes funcionais de uma estrutura de edifício .....	122
Figura 9.7 - Aplicação de uma geminação positiva a partir de uma laje inicial .....	128
Figura 9.8 - Idem Figura 9.7(b) (geminação +), com eliminação dos objetos P2 e P4 .....	130

Figura 9.9 - Diagramas de ações para as diversas decomposições de ações funcionais em um projeto, com desvio do fluxo estabelecido pelo mecanismo de controle da linguagem de ações .....	132
Figura 9.10 - Organização modular para geração da laje inicial .....	135
Figura 9.11 - Casos de uma geminação positiva .....	137
Figura 9.12 - Organização modular para uma geminação positiva, a partir da semente laje inicial .....	138
Figura 10.1 - Relacionamento entre duas entidades da classe semente amarela .....	142
Figura 10.2 - Relacionamento entre duas entidades da classe semente verde .....	143
Figura 10.3 - Relacionamento entre uma entidade da classe semente amarela e uma verde (primeira geração) .....	143
Figura 10.4 - Relacionamento entre duas sementes amarelas, na segunda geração .....	144
Figura 10.5 - Representação tridimensional de um cruzamento entre dois indivíduos amarelos híbridos .....	145
Figura 10.6 - Configuração espacial do metano .....	148
Figura 10.7 - Conseqüência funcional de um triplo relacionamento entre entidades moleculares ..	149
Figura 10.8 - Unidade algébrica computacional .....	152
Figura 10.9 - Representação 3-D para os termos algébricos da equação $2 \cdot X + 2 = 10$ .....	152
Figura 10.10 - Representação 3-D para os termos $X^2$ e $X^3$ .....	153
Figura 10.11 - Relacionamentos em uma matriz algébrica tridimensional .....	154

## LISTA DE TABELAS

Tabela 2.1 - Resumo das operações de acoplamento de volumes .....	34
Tabela 3.1 - Valores de RG .....	38
Tabela 3.2 - Resumo dos mecanismos e símbolos do capítulo .....	40
Tabela 4.1 - Resumo dos mecanismos e símbolos do capítulo .....	43
Tabela 5.1 - Códigos de texturas para preenchimento (pintura) das faces dos volumes .....	53
Tabela 5.2 - Códigos de cores .....	53
Tabela 5.3 - Resumo dos mecanismos e símbolos do capítulo .....	63
Tabela 6.1 - Códigos para o parâmetro TA, a serem utilizados no mecanismo para acoplamento de universos $\theta$ .....	67
Tabela 6.2 - Resumo dos mecanismos e símbolos do capítulo .....	70
Tabela 7.1 - Resumo dos mecanismos e símbolos do capítulo .....	90
Tabela 8.1 - Resumo dos mecanismos e símbolos do capítulo .....	100
Tabela 9.1 - Resumo dos mecanismos e símbolos do capítulo .....	139

## RESUMO

Este trabalho apresenta uma metodologia computacional para a geração, manipulação e visualização de elementos tridimensionais cuja forma geométrica básica é o paralelepípedo. Os entes geométricos são agrupados em classes funcionais de objetos e suas ocorrências são representadas em uma estrutura de dados funcional, criando-se uma representação relacional dos atributos funcionais das respectivas entidades. Além disso, também são consideradas as relações funcionais que podem ser produzidas entre os objetos modelados, estabelecendo-se uma linguagem de ações que permite levar em consideração as consequências funcionais produzidas por uma determinada ação de projeto aplicada em um determinado estágio de projeto.

**ABSTRACT**

This work presents a computational methodology for generating, handling and rendering tridimensional elements using parallelepipedous as their basic geometric form. The entities geometric are grouped in functional classes of objects and their occurrences are represented in a functional data structure where functional attributes for every entity are included using relational representations. Furthermore, functional relations among modelled objects are also considered and an action language is established in order to consider the functional consequences triggered by design actions during the different stages of the design process.

## 1. INTRODUÇÃO

A elaboração de um projeto utilizando-se um sistema computacional pode ser vista como um processo de interação contínua entre o projetista e o objeto de projeto, assim sendo, pode-se dizer que *projetar é uma série de ações ou operações que levam à construção de um produto final*. Muitas variáveis devem ser consideradas em um problema de projeto e os valores envolvidos sofrem várias alterações durante o processo. Assim, um projeto apresenta soluções preliminares (*projeto esquemático*), até atingir aproximações de alto nível (*projeto preliminar e final*). Portanto, pode-se afirmar que *um projeto pode ser elaborado por meio de uma sequência de ações a serem aplicadas, preservando-se a interatividade com o projetista, de forma a permitir um correto "feedback", caso seja necessário /ALV 87/*.

Computadores constituem-se num meio eficiente de assistência na tomada de decisões na elaboração de um projeto, seja ele, uma nova peça mecânica, uma estrutura de edifício, uma torre de alta tensão ou até mesmo um novo modelo de embarcação. Assim, o projetista pode decidir melhor se possuir um ambiente em que possa criar seu próprio universo de trabalho, i. é, um ambiente que permita a concepção, representação, refinamento e avaliação das idéias de projeto. Um ambiente adequado para desenvolvimento de projetos em arquitetura deve, portanto, permitir o uso de representações gráficas alternativas e manipulação de transformações geométricas /ALV 87/.

Os profissionais envolvidos na tarefa de concepção de um projeto geralmente utilizam representações gráficas ou simbologias escritas, i. é, *convenções*. As con-

venções auxiliam tanto os profissionais envolvidos no projeto como a outras pessoas a entenderem melhor as qualidades e características dos objetos projetados. Assim, um ambiente computacional de trabalho adequado, deverá ter condições de representar e entender informações, tais como, tamanhos, proporções, materiais, temperatura, peso, cores, formas e volumes /ALV 87/.

Muitos sistemas do tipo CAD (*Computer Aided Design*) surgiram a partir da metade dos anos 60, principalmente devido ao advento de computadores eficientes e periféricos para exibição gráfica de melhor nível. Inicialmente, os sistemas eram basicamente editores gráficos para elaboração de desenhos em duas dimensões (2D), que ofereciam aos usuários um limitado conjunto de primitivas gráficas /FOL 82/, tais como, pontos, linhas, círculos e arcos. No final da década de 60, novos sistemas surgiram, incorporando recursos limitados para modelagem de objetos em três dimensões, elaborados a partir de construções fundamentadas em pontos, linhas e arcos. Este tipo de representação passou a ser conhecido desde então como modelo "wireframe", visto que os objetos modelados por este processo pareciam ser construídos pela interligação de finos arames. Durante a década de 70 os sistemas de CAD passaram a incorporar técnicas para representação de objetos por meio de suas faces e como consequência, começaram a surgir processos para remoção de linhas e faces ocultas /FOL 82/ /ROG 85/, permitindo dessa forma uma representação mais realista e elaborada dos objetos projetados. Também, ao longo dos anos 70 e início da década de 80, problemas como a representação de faces curvas e o modelamento de objetos sólidos /MOR 85/ começaram a ser estudados pelos pesquisadores que atuam na área de computação gráfica /SMI 89/, inclusive o grupo de modelagem geométrica da UFRGS. Atualmente, essas técnicas estão implementadas em diversos pacotes do tipo CAD,

disponíveis no mercado internacional de *software*.

Enquanto as técnicas apresentadas acima limitavam-se a uma descrição geométrica dos objetos, novas tendências começaram a surgir, baseadas em novas técnicas que objetivam uma forma de representação que também leva em consideração os relacionamentos funcionais, como pode ser visto em /GRA 89/. Assim, os objetos modelados, além de existirem geometricamente dentro de um universo de trabalho, terão uma função vital dentro do referido universo, e a ocorrência de um objeto poderá apresentar algum tipo de *relacionamento* com os outros objetos. Note-se que um *relacionamento* poderá ser uma ligação ou um conhecimento recíproco de objetos. Assim, ao ativar-se uma operação alteradora ou de inserção, deve existir um meio para estabelecer as *conseqüentes* alterações nos *relacionamentos funcionais* produzidas por essa ação, examinando-se também a existência ou não de *conflitos* ou *restrições*.

Como exemplo, suponha-se que um projetista está gerando objetos em um sistema de CAD que leva em consideração, além da *representação geométrica* dos objetos, as *relações funcionais* reais que ocorrem entre os entes projetados. Então, um objeto inicial é gerado e, após a geração desse primeiro elemento, um novo objeto é criado, sendo conectado ao primeiro. Os dois objetos são constituídos de um tipo de poliestireno expandido, conhecido como *isopor*, que é um material combustível. Ao ser criado um terceiro elemento, conectado ao segundo e constituído de material *ferro* previamente aquecido a uma temperatura de 1000°C, os dois primeiros são volatilizados. Note-se, que se o sistema utilizado para geração dos três objetos fosse um sistema convencional de CAD, i. é, que leva em consideração apenas as características geométricas dos elementos modelados, tal *conseqüência funcional* talvez



não tivesse sido considerada. Portanto, pode-se concluir que ao projetarmos alguma coisa, uma representação geométrica convencional pode não ser suficiente para representar os objetos envolvidos no processo, visto que outros fatores, de caráter *funcional*, devem ser também considerados.

Os referidos fatores *funcionais* (*funções vitais*) de um projeto, podem ser inseridos em um sistema de CAD por meio de mecanismos *condicionais* ou de *controle*, aliados a descrições adequadas sobre os objetos envolvidos no projeto. Os mecanismos *condicionais* podem ser vistos como regras a serem seguidas durante os estágios do processo de projeto. Uma regra pode ser vista como sendo uma *ação*, que pode ser aplicada a um estágio inicial (*um objeto inicial / uma pré-condição*), e cujo objetivo é atingir um estágio final (*um resultado / uma pós-condição*), que será o produto de uma composição das várias ações aplicadas.

Uma regra poderá ser aplicada a um objeto, para conceber um estágio mais avançado do projeto, paralelamente a um ou mais *testes de validação das regras*, i. é, ao ser aplicada uma regra, o sistema deverá verificar se a referida ação é permitida (*regra válida*) ou não (*regra inválida*). Assim, por exemplo, o plano diretor e o código de edificações de uma cidade estabelecem uma série de *normas reguladoras* que visam disciplinar as novas construções, estabelecendo as condições mínimas que os ambientes devem satisfazer quanto a estabilidade/segurança, higiene e conforto ambiental. Tais instrumentos de controle poderiam ser vistos como regras que devem ser seguidas pelos profissionais envolvidos neste tipo de atividade. Portanto, um sistema computacional de auxílio a projetos deverá ter condições de identificar uma ação que seja incompatível com as regras estabelecidas. Para possibilitar isso, o sistema deverá possuir uma linguagem

própria, de tal forma que permita ao projetista conceber um programa computacional, utilizando uma linguagem de programação própria para projetos, a qual será chamada no decorrer do trabalho de *linguagem de ações*, como proposto em /COY 86/. Note-se, ainda, que as regras, por exemplo, estabelecem valores mínimos, i. é, o projetista deverá ter liberdade para utilizar valores maiores ou iguais aos estabelecidos, de tal forma que possa aplicar ao projeto um *desenho próprio*, se assim o desejar.

Além disso, é necessário mencionar mais uma vez, que os elementos de projeto deverão estar *vinculados*, de tal forma que uma alteração em um deles, poderá ativar uma *reação em cadeia*, provocando assim, alterações nos demais, se for necessário. Portanto, o sistema deverá ser *sensível* às modificações, reagindo às ações aplicadas.

A proposta deste trabalho é criar mecanismos computacionais para *modelamento de objetos funcionais*, permitindo além disso, que o projetista possa se comunicar melhor com os objetos envolvidos no projeto, como sugerido em /ALV 87/. O ambiente proposto deverá constituir-se numa ferramenta que permita ao projetista projetar melhor, utilizando-se de um processo que possibilite incluir etapas de "*feedback*", *avaliação e reformulação de táticas* envolvidas no processo de projeto /JOR 84/, mas de tal forma que certas *condições ou restrições* de projeto sejam obedecidas utilizando-se um *mecanismo de controle* /COY 86/ apropriado, e as *consequências funcionais (vitais)* sejam consideradas no decorrer das etapas do processo. Deve-se considerar, ainda, que podem existir duas classes de objetos de projeto: *exatas (onde as restrições são mensuráveis)* e *subjetivas (onde as restrições são aferíveis, e portanto, de difícil mensuração)* /TUR 90/. Então, por exemplo, a *noção de função vital em arquitetura (categoria subjetiva)* pode ser considerada imprecisa, e

portanto, não deverá ser incluída neste trabalho. Quanto ao cálculo estrutural (categoria exata), a noção de função vital pode ser adotada sem maiores riscos, visto que as restrições podem ser mensuráveis.

## 1.1 Utilização de regras de "shape grammars" na análise e criação de projetos

Diversos trabalhos ligados à área de arquitetura fazem referência à utilização de regras de *shape grammars* na análise e criação de projetos. Uma apresentação de *shape grammars* é feita em /FLE 86/:

- *Shape Grammars* podem ser usadas para explicar e descrever uma determinada associação de objetos que possuem características comuns, ou para desenvolver novos objetos e testar as regras em que eles se fundamentam.
- *Shape Grammars* podem conduzir a resultados mais compreensivos, estabelecidos com precisão e rigor, e a um profundo entendimento das questões envolvidas.

Flemming /FLE 86/, referenciado também em /ALV 87/, estabelece a seguinte definição de *Shape Grammars*:

→ Shape Grammars: São conjuntos de regras, que, aplicadas a conjuntos de elementos, produzem efeitos sobre estes elementos e seus relacionamentos espaciais. São sistemas fundamentados em regras que, basicamente, consistem de um estágio inicial, um conjunto finito de elementos de um vocabulário, um conjunto de regras de produção (SE-ENTÃO) que são aplicadas aos elementos, e uma condição terminal.

No trabalho /FLE 86/, *Flemming* sugere a utilização de *shape grammars* no estudo de casos de organizações espaciais de residências. Assim, princípios de planejamento espacial poderiam ser estabelecidos, e esses princípios seriam considerados na concepção de regras de *shape grammars* que seriam utilizadas na geração de plantas baixas, obedecendo somente estes princípios. Então, seria considerado um estágio inicial, composto por um *hall* de entrada, representado por um retângulo rotulado com o identificador H, além de outros rótulos auxiliares (FR = FRente da casa e FU = FUndos da casa) que seriam utilizados como referenciais para a aplicação de regras de composição espacial (Figura 1.1). Note-se que os rótulos podem ser vistos como elementos que são utilizados para controlar a maneira como as relações espaciais são realizadas na geração de um novo elemento de um projeto. Então, algumas partes dos elementos de um projeto podem ser rotuladas de forma que regras possam ser aplicadas somente se uma configuração correta de rótulos estiver presente. Portanto, certas regras podem ser aplicadas para construir configurações de rótulos para controlar a aplicação das regras subsequentes /ROO 87/. Deve-se mencionar, ainda, que geralmente é desejável que o final de um projeto esteja

livre de rótulos ou marcas especiais, visto que sua função é justamente guiar e controlar a geração de elementos. Os rótulos ou marcas são referenciados como dispositivos não terminais dos elementos, visto que eles não aparecem nas produções finais dos projetos válidos. Então, a ausência de rótulos ou marcas produz uma condição para a base de decisão completar a geração de formas usando regras.

A Figura 1.2 mostra algumas regras que podem ser aplicadas ao estágio inicial para a geração de outros estágios do processo de projeto do *layout* da organização espacial. A primeira regra adiciona uma sala (S) nos fundos (FU) do *hall* (H) que se localiza na frente da casa. Essa regra se fundamenta no fato de que em muitas casas o *hall* e a *sala de estar* são adjacentes na frente da casa. Note-se, ainda, que foi criado um rótulo auxiliar (X) que tanto pode ser (FR) como (FU), o que possibilita a aplicação das regras para os dois casos, por meio de uma operação de rotação. Além disso, na Regra 4, foi utilizado o rótulo (C) que representa o centro da planta.

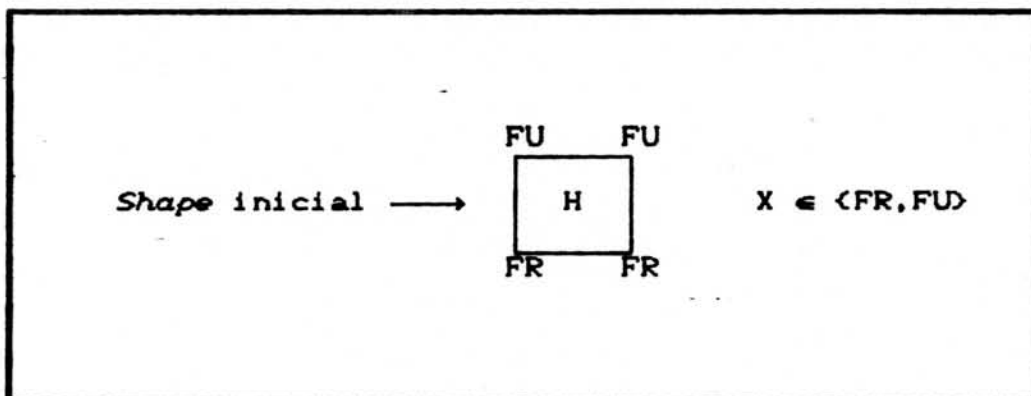
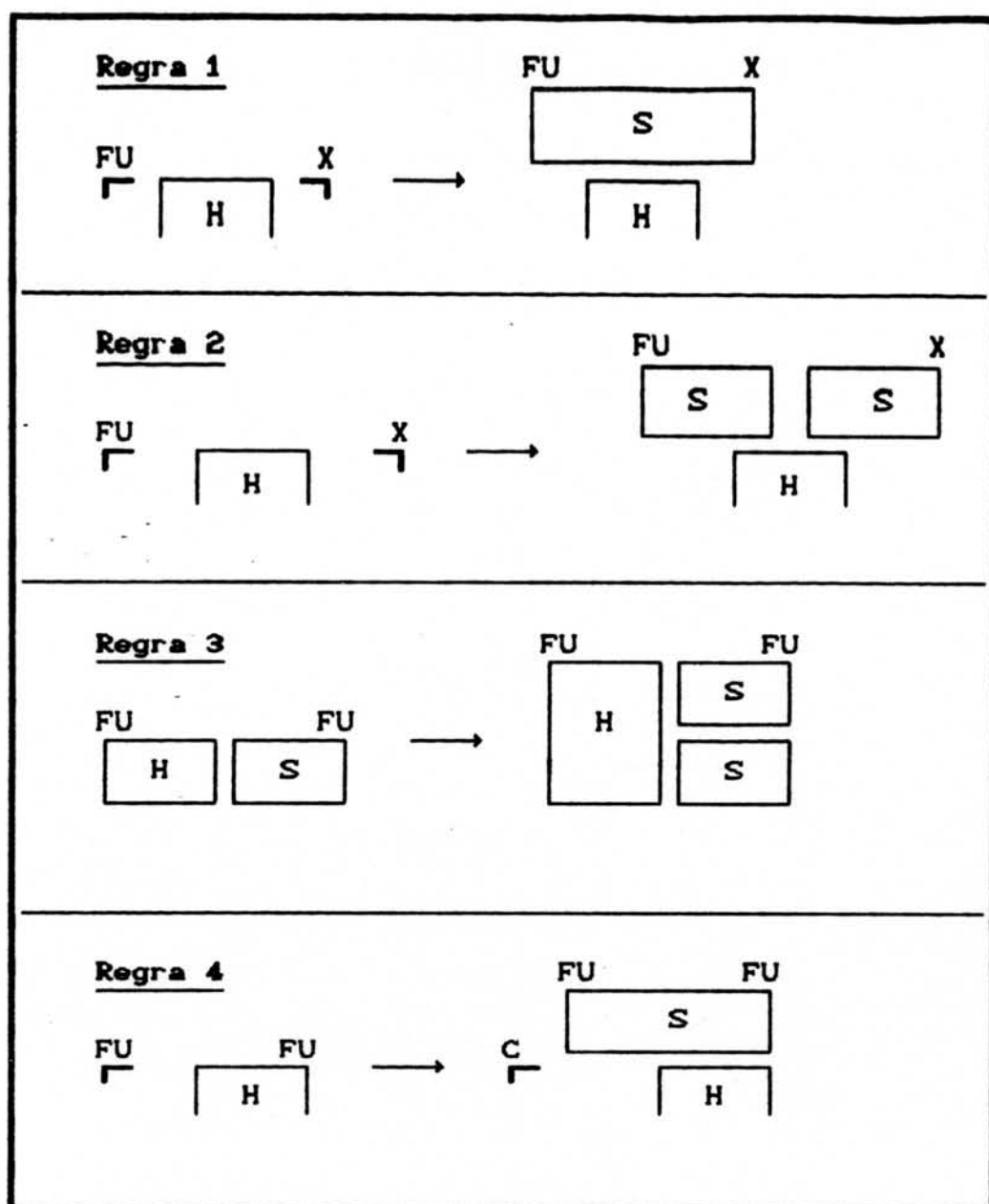


Figura 1.1 - Condição inicial de uma organização espacial



**Figura 1.2** - Regras de uma gramática de layout

Assim, por exemplo, aplicando-se a Regra 1 da Figura 1.2 pode-se obter várias soluções para uma organização espacial, que pode ser representada na forma de uma estrutura do tipo árvore, como mostra a Figura 1.3.

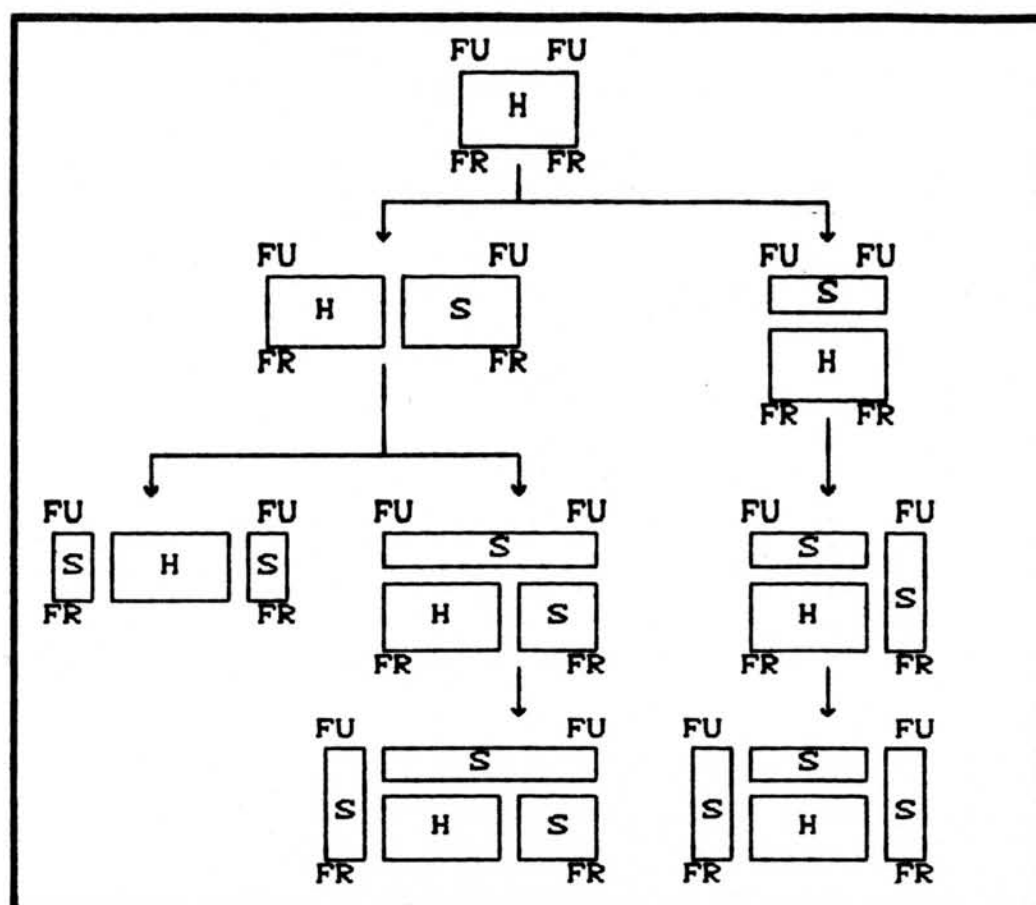


Figura 1.3 - Derivação de layouts pela aplicação da Regra 1 da Figura 1.2

## 1.2 Geração de objetos tridimensionais a partir da sugestão de utilização de regras de shape grammars

Na maioria dos trabalhos que referenciam o uso de *Shape grammars* na concepção e análise de projetos, a abordagem bidimensional tem recebido uma maior atenção por parte dos autores /COY 86/ /FLE 86/ /ROO 87/, ficando a geração tridimensional, geralmente, colocada na forma de sugestões teóricas /ALV 87/ /COY 86/ /FLE 86/ /STI 85/, sem a apresentação de um desenvolvimento prático e formalizado. Assim, por exemplo, Stiny em seu trabalho /STI 85/, afirma que projetos podem ser gerados manipulando-se as relações espaciais existentes entre os volumes de um vocabulário. Então, um projeto pode iniciar com um volume simples e um

novo projeto pode ser produzido a partir do projeto corrente pela adição ou subtração de um novo volume, criando-se relações espaciais que, posteriormente, poderão ser empregadas na geração de outros projetos, constituindo dessa forma, uma linguagem de projeto fundamentada em *shape grammars*.

É de fundamental importância salientar que o trabalho a ser desenvolvido neste documento originou-se na idéia de construção de um "LEGO eletrônico", sugerida pelo arquiteto *Benamy Turkienicz* e publicada em /TUR 90/. Assim, no sistema proposto por *Turkienicz*, objetos tridimensionais seriam modelados por meio de acoplamentos, a partir de uma situação inicial, utilizando-se uma gramática de formas (*shape grammars*). Também, a parte referente à inserção de uma linguagem de ações e dos relacionamentos funcionais foram sugeridas por *Turkienicz* como objetivos a serem atingidos. O trabalho levou em consideração também uma análise de objetos armados com o jogo LEGO. Foi cedido por *Turkienicz*, também, um "LEGO de madeira" /TUR 90/ para manipulação e análise.

Assim, no decorrer deste documento, o que se pretende fazer é explorar a aplicação das idéias de *shape grammars* em uma abordagem tridimensional, como sugerido por *Turkienicz*. A motivação principal para a elaboração dos capítulos constituintes deste trabalho pode ser resumida em uma pergunta:

→ Como gerar e manipular objetos tridimensionais, a partir da idéia de composição de projetos utilizando-se regras de *shape grammars* ?



Além de realizar uma investigação para descobrir uma forma adequada de satisfazer à pergunta colocada anteriormente, uma outra questão deverá ser respondida ao final deste trabalho:

→ Como incorporar os atributos e relacionamentos funcionais (funções vitais) aos elementos tridimensionais de um projeto, de forma que ao ser aplicada uma sequência de ações (regras), as consequências funcionais (vitais) resultantes possam ser obtidas para análise ?

Este trabalho foi dividido nos seguintes capítulos:

- Capítulo 2: Apresenta algumas definições importantes relativas à utilização de regras na geração de volumes, de acordo com os princípios de funcionamento de um "LEGO eletrônico", propostos em /TUR 90/.
- Capítulo 3: Estabelece conceitos mais formalizados para a geração de volumes e apresenta os primeiros mecanismos da linguagem de ações.
- Capítulo 4: Apresenta o mecanismo de repetição gerativa, a ser utilizado para a geração simultânea de volumes, de acordo com uma regra de geração pré-estabelecida.

- > Capítulo 5: Estabelece mecanismos da linguagem de ações para a aplicação de operações de alteração sobre os volumes gerados.
- > Capítulo 6: Define um mecanismo que permite o acoplamento de dois ou mais conjuntos de volumes, gerados em momentos distintos no processo de projeto.
- > Capítulo 7: Incorpora procedimentos e dispositivos para automatizar a geração de volumes, de acordo com determinadas regras de ações de projeto. Além disso, são definidos alguns mecanismos auxiliares da linguagem de ações.
- > Capítulo 8: Acrescenta o mecanismo auxiliar de controle da linguagem de ações.
- > Capítulo 9: Define mecanismos para incorporar os atributos e os relacionamentos funcionais (vitais) dos elementos de um projeto.
- > Capítulo 10: Apresenta algumas possibilidades de aplicação das idéias estabelecidas nos capítulos anteriores em outras áreas de pesquisa (biologia, química e matemática).
- > Capítulo 11: Apresenta algumas conclusões a respeito do trabalho desenvolvido.
- > Anexos: Ilustram alguns resultados obtidos com a utilização do programa SMOF.EXE (Sistema de Modelamento de Objetos Funcionais) desenvolvido para testar as idéias apresentadas no decorrer deste documento.

## 2. UTILIZAÇÃO DE REGRAS NA CRIAÇÃO DE VOLUMES

De acordo com /MAH 83/, uma regra importante na geração dos elementos de um projeto é que um objeto inicial deve servir como ponto de partida para o projeto de outro. Assim, a elaboração de um projeto pode ser realizada em diversas etapas, a partir de uma situação inicial. Assim, deve-se gerar um volume inicial, e então, a partir desse volume inicial, criar outros volumes mediante a utilização de determinadas regras de geração de volumes, como sugerido nas definições de *shape grammars* /FLE 86/.

O volume básico que será utilizado como matéria-prima para a elaboração dos estudos contidos neste trabalho, será o *paralelepípedo*. Assim, todas as operações e procedimentos a serem concebidas posteriormente, serão fundamentadas nessa forma. O *paralelepípedo* foi escolhido porque é uma forma geométrica básica, de fácil definição e manuseio.

Inicialmente, considere-se um volume inicial (um *paralelepípedo*), que terá 8 vértices (a, b, c, d, e, f, g, e h), como ilustrado na Figura 2.1.

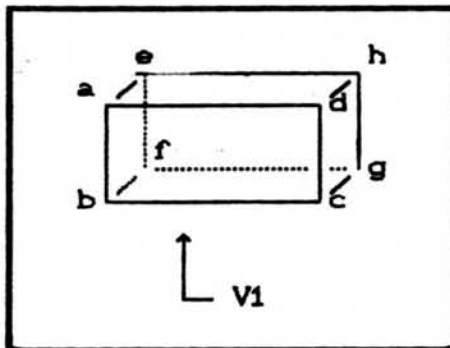


Figura 2.1 - Volume inicial.

A partir de VI, pode-se criar outros volumes, aplicando-se três tipos fundamentais de *acoplamento* de volumes, definidos em /AND 81/. Note-se que o referido trabalho apenas define os acoplamentos, não apresentando uma gramática de formas para volumes. A seguir, serão definidos os tipos de acoplamento (positivos e negativos), que também não estão definidos em /AND 81/. Observe-se que estes tipos de acoplamento foram definidos nesta dissertação devido a necessidade de acoplar nas duas direções distintas, para um mesmo tipo de acoplamento (*geminção, justaposição e superposição*). Vale mencionar, ainda, que as operações de acoplamento estão intimamente relacionadas aos rótulos das faces do volume de referência utilizado.

- |   |   |
|---|---|
| 1 | → <u>Geminção</u> : É um acoplamento na direção lateral, à esquerda ou à direita do paralelepípedo. |
| 2 | → <u>Justaposição</u> : É um acoplamento na parte posterior ou anterior do paralelepípedo.          |
| 3 | → <u>Superposição</u> : É um acoplamento na direção vertical, acima ou abaixo do paralelepípedo.    |

Visto que cada uma das três operações de acoplamento definidas acima podem ser subdivididas em duas operações, dependendo do sentido em que o acoplamento ocorrerá, pode-se redefiní-las da seguinte maneira:

- |   |  |
|---|--|
| 1 | → <u>Geminção positiva</u> : É um acoplamento à direita do paralelepípedo, i. é, no sentido do eixo X. |
|---|--|

- 2 → Geminacão negativa: É um acoplamento à esquerda do paralelepípedo, i. é, no sentido inverso ao eixo X.
- 3 → Justaposição positiva: É um acoplamento na parte posterior do paralelepípedo, i. é, no sentido do eixo Z.
- 4 → Justaposição negativa: É um acoplamento na parte frontal do paralelepípedo, i. é, no sentido oposto ao eixo Z.
- 5 → Superposição positiva: É um acoplamento na parte superior do paralelepípedo, i. é, no sentido do eixo Y.
- 6 → Superposição negativa: É um acoplamento na parte inferior do paralelepípedo, i. é, no sentido inverso ao eixo Y.

As Figuras 2.2 → 2.7 mostram o funcionamento de cada uma das seis operações de acoplamento definidas acima. Note-se, ainda, que os elementos (volumes) vão sendo conectados como se fossem peças de um jogo de armar, de acordo com a proposta do "LEGO eletrônico", apresentada por Turkienicz /TUR 90/.

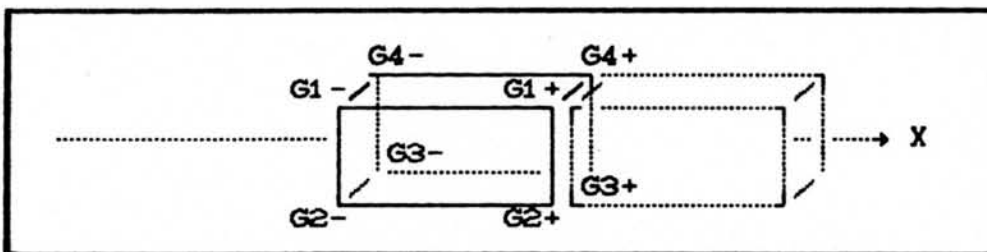


Figura 2.2 - Geminacão positiva.

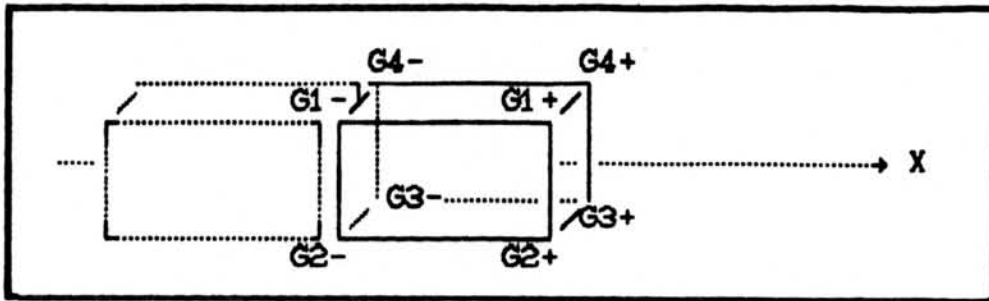


Figura 2.3 - Geminção negativa.

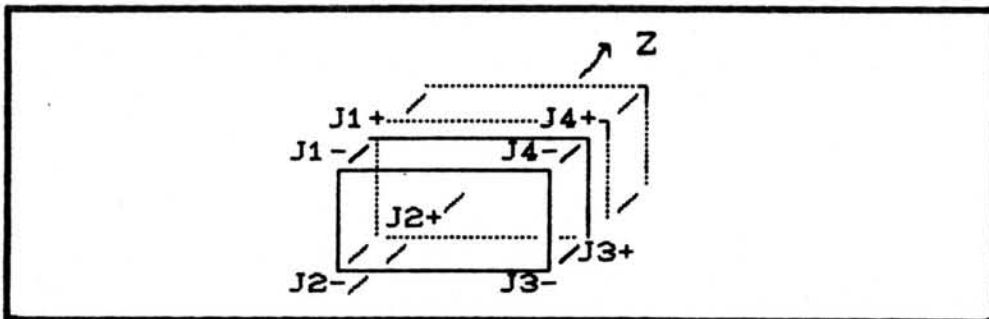


Figura 2.4 - Justaposição positiva.

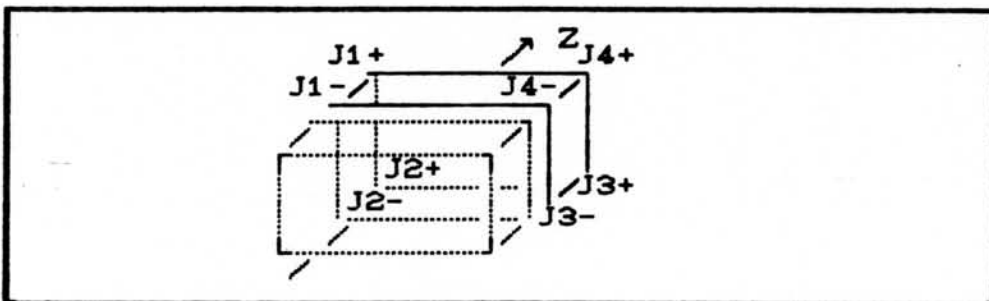


Figura 2.5 - Justaposição negativa.

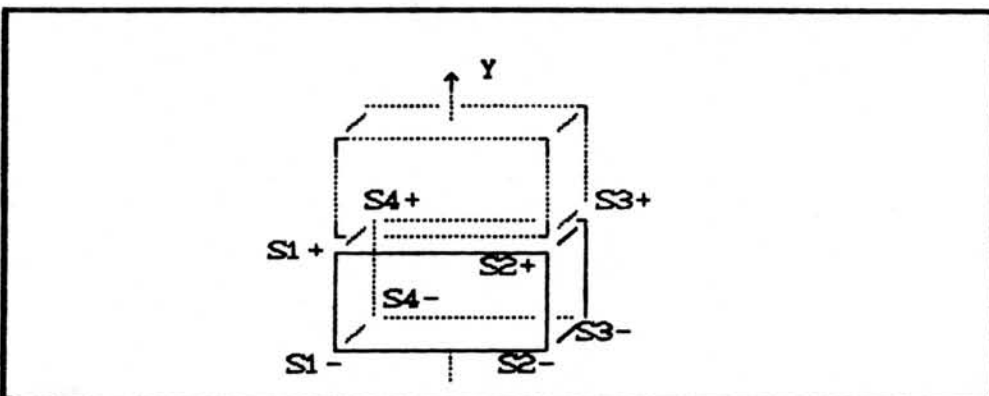


Figura 2.6 - Superposição positiva.

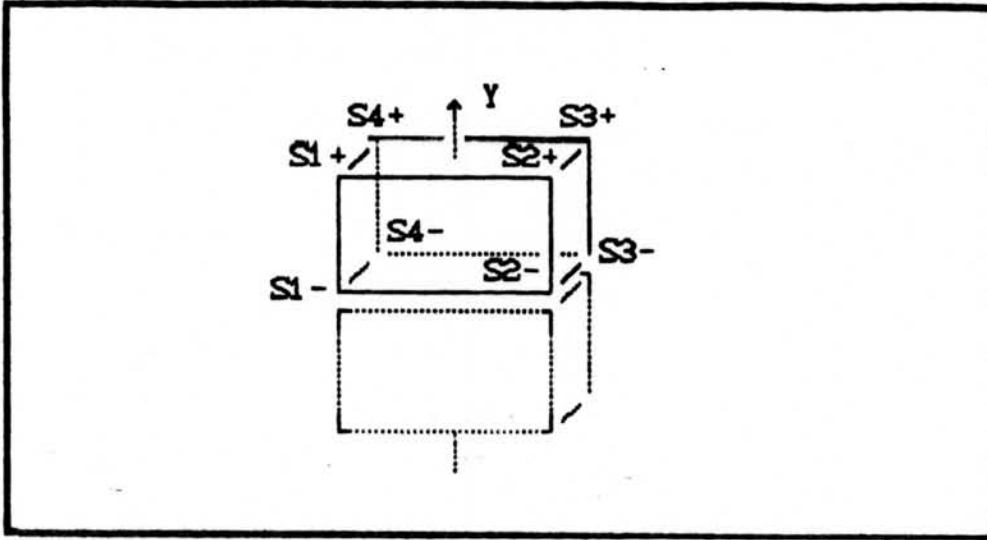


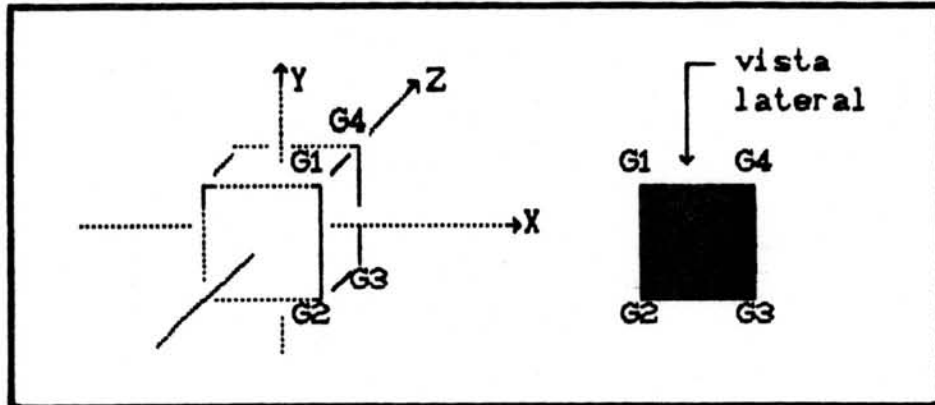
Figura 2.7 - Superposição negativa.

A partir de agora, torna-se necessário definir um conceito fundamental para o perfeito entendimento do que será apresentado a seguir, visto que este conceito está intimamente ligado à geração de novos volumes. Então, a partir de um paralelepípedo já existente no universo de trabalho pode-se definir:

Semente - É o paralelepípedo que é tomado como base para as operações de acoplamentos e alterações.

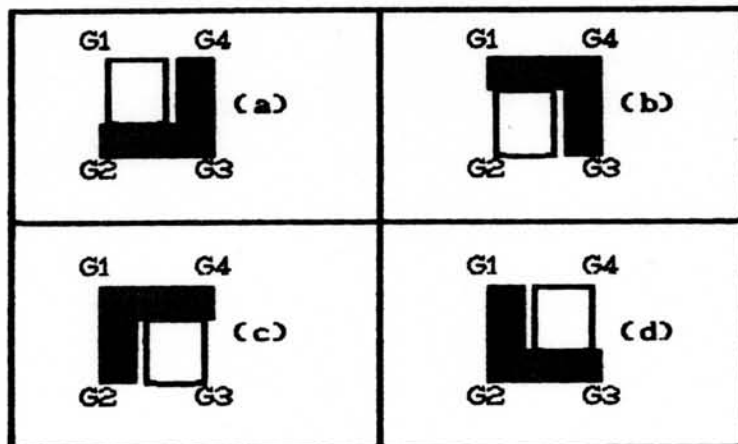
Deve-se considerar, então, o caso em que o novo volume a ser gerado pode ter suas dimensões diferentes daquelas que a semente possui. Além disso, se for considerado o fato de que uma operação de acoplamento, utilizando-se paralelepípedos pode ser realizada relativa a cada um dos quatro vértices da face da semente (a face de referência depende do tipo de acoplamento escolhido), é possível definir mais quatro operações a partir daquelas definidas anteriormente. Para demonstrar isso, considere-se o caso de uma *geminção positiva*.

Em primeiro lugar, torna-se necessário definir os nomes dos vértices, que serão tomados como referência (Figura 2.8):



**Figura 2.8** - Nomenclatura dos vértices utilizados como referência nas operações do tipo *geminção positiva*.

O passo seguinte será a definição de quatro operações distintas para este tipo de acoplamento, relativas a G1, G2, G3 e G4 (Figuras 2.9 e 2.10).



**Figura 2.9** - Quatro tipos de *geminção positiva* (vista lateral).



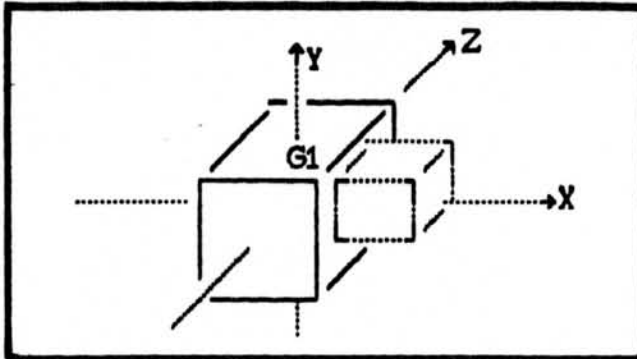


Figura 2.10 - Perspectiva da Figura 2.9(a) / *germinação positiva com referencial em G1.*

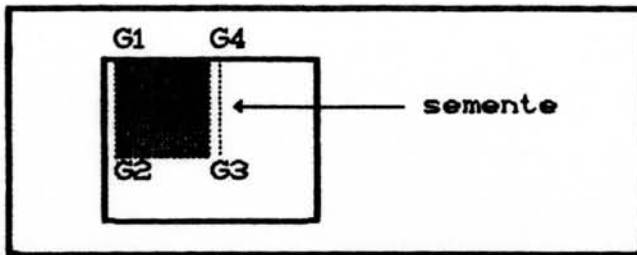


Figura 2.11 - Caso da Figura 2.9(a) quando as dimensões do volume a ser criado forem maiores que as da *semente*.

Se as dimensões do paralelepípedo a ser criado forem maiores que as da *semente*, as operações acima ilustradas podem ser igualmente realizadas (Figuras 2.11 e 2.12).

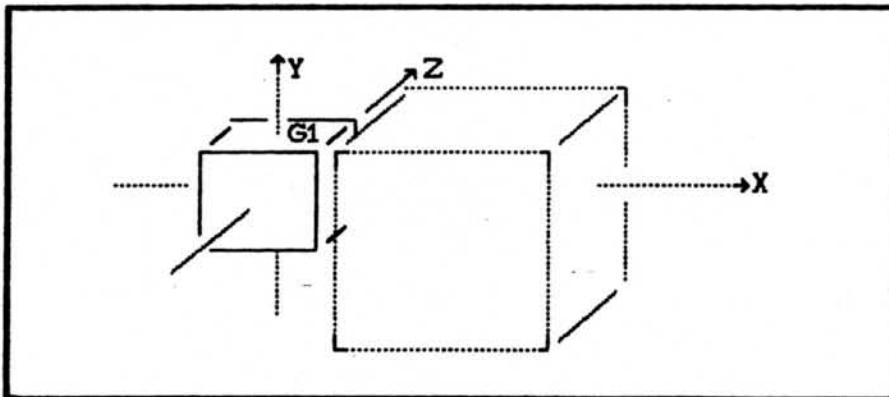


Figura 2.12 - Perspectiva da Figura 2.11.

Outro conceito de fundamental importância é aquele que define as características geométricas de um novo volume a ser gerado:

Volume Padrão - Possui as características geométricas do novo paralelepípedo que será gerado a partir da semente.

Tabela 2.1 - Resumo das operações de acoplamento de volumes.

Geminção	Positiva	$G1^+$	$G2^+$	$G3^+$	$G4^+$
	Negativa	$G1^-$	$G2^-$	$G3^-$	$G4^-$
Justaposição	Positiva	$J1^+$	$J2^+$	$J3^+$	$J4^+$
	Negativa	$J1^-$	$J2^-$	$J3^-$	$J4^-$
Superposição	Positiva	$S1^+$	$S2^+$	$S3^+$	$S4^+$
	Negativa	$S1^-$	$S2^-$	$S3^-$	$S4^-$

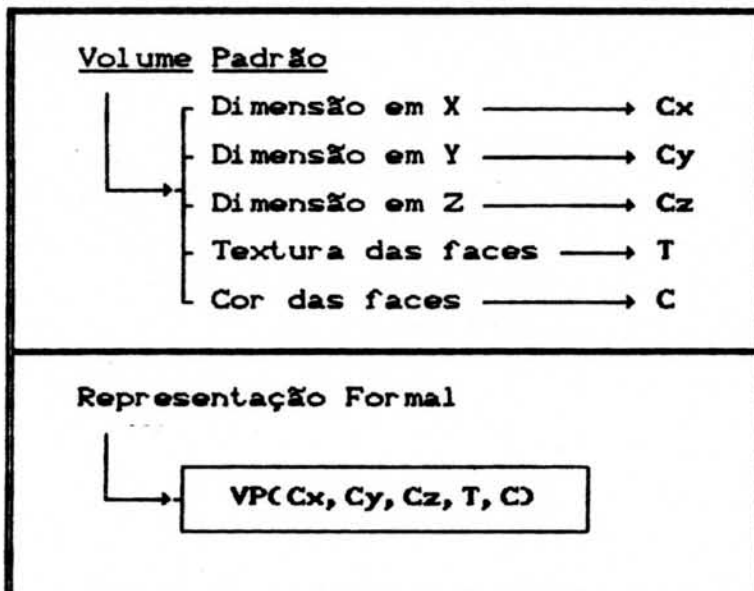
Finalizando este capítulo, a Tabela 2.1 apresenta um resumo das operações de acoplamento necessárias para a manipulação de volumes (são necessárias 24 operações diferentes).

### 3. FORMALIZANDO A GERAÇÃO DE VOLUMES

Na seção anterior, foram definidas as regras básicas de geração de volumes (por meio de acoplamentos), essenciais para a manipulação de objetos em três dimensões. Agora, serão estabelecidos alguns conceitos importantes, cuja finalidade, será proporcionar um formalismo adequado às operações de manipulação de volumes. Vale salientar, ainda, que os mecanismos que serão apresentados a partir de agora, posteriormente, farão parte da *linguagem de ações* que será apresentada no momento oportuno no decorrer deste trabalho.

#### 3.1 Volume padrão

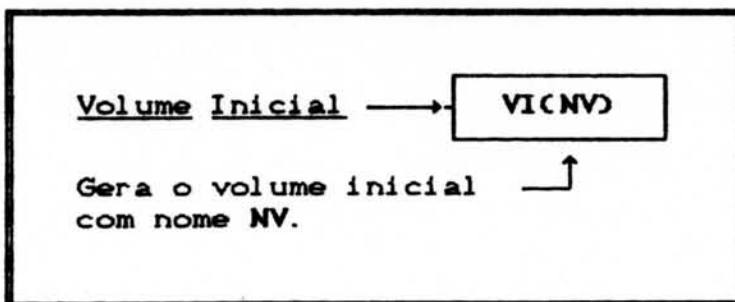
Em primeiro lugar, deve-se estabelecer as dimensões e demais características do *volume padrão*, visto que os volumes a serem gerados seguirão o padrão estabelecido por este mecanismo. Assim, pode-se definir o *volume padrão*:



Como pode ser observado acima, serão necessários cinco parâmetros, na definição do *volume padrão*. Os três primeiros (*dimensão em X, Y e Z*), vão definir as *dimensões* relativas a cada um dos três eixos do sistema de coordenadas. Os parâmetros seguintes, *textura e cor*, serão utilizados para o preenchimento (pintura) das faces, cuja finalidade será proporcionar um aspecto mais elaborado (realístico), dos objetos gerados.

### 3.2 Volume inicial

Visto que a criação de volumes será realizada pela aplicação de *regras de geração*, será necessário gerar uma *forma inicial*, que será o marco inicial do processo gerativo. Então, estabelecido o *volume padrão*, deve-se gerar o *volume inicial*, que será utilizado posteriormente como *semente* para a geração dos demais volumes. Note-se, ainda, que este volume será gerado com as características do *volume padrão*. O comando formal para geração do volume inicial pode ser visto logo abaixo:



### 3.3 Aplicando operações de transformação para gerar os diversos estágios do processo gerativo

A geração de um *volume inicial* pressupõe que existe um *estágio zero*, no qual não existe nenhum objeto e nenhuma operação foi ainda realizada. Esse estágio será

representado por E0.

E0 ← Estágio Zero

Gerado o volume inicial, pode-se aplicar uma operação de transformação sobre este volume, cujo objetivo é a criação de um novo estágio do processo gerativo. Esta transformação poderá ser qualquer uma das 24 operações de acoplamento de volumes definidas anteriormente, e será indicada formalmente como pode ser visto a seguir:

E1 ← VI ← Estágio 1 do processo gerativo  
é igual ao Volume Inicial.

$\tau$  ← V(NV, RG) ← A transformação  $\tau$  gera o  
volume V, com nome NV e  
usa regra de geração RG.

E1 \*  $\tau$  → E2 ← Aplicação de uma transfor-  
mação  $\tau$  sobre o Estágio 1,  
para geração do Estágio 2  
do processo gerativo.

Portanto:

E2 ← Estágio 2 do processo gerativo é igual  
ao volume inicial, mais o novo volume  
gerado pela operação de transformação  
 $\tau$ .

Observe-se, acima, que a transformação  $\tau$  necessita de dois parâmetros:

Nome do Volume [NV] ← Sua função é propor-  
cionar uma individualização do volume, de  
forma a permitir sua manipulação.

**Regra de Geração [RG]** ← Estabelece o tipo de regra a ser utilizada na operação de acoplamento estabelecida pela transformação  $\tau$ . RG pode assumir os valores mostrados na Tabela 3.1.

**Tabela 3.1 - Valores de RG.**

RG	Regra	RG	Regra
1	G1+	13	J1-
2	G2+	14	J2-
3	G3+	15	J3-
4	G4+	16	J4-
5	G1-	17	S1+
6	G2-	18	S2+
7	G3-	19	S3+
8	G4-	20	S4+
9	J1+	21	S1-
10	J2+	22	S2-
11	J3+	23	S3-
12	J4+	24	S4-

A seguir, será apresentado um exemplo que ilustra a utilização dos conceitos definidos até o momento.

**Exemplo 3.1** → Criar um volume inicial, de nome V1, cujas dimensões são todas iguais a 250, com uma textura cujo código é 1 e cor cujo código é 2 e, a seguir, aplicar duas geminações positivas (G1<sup>+</sup>), V2 e V3, e uma superposição negativa (S1<sup>-</sup>), V4, alterando-se a dimensão em Z do volume padrão para 500.

→ Passo 1 - Gerar o Volume Padrão:

VP(250, 250, 250, 1, 2)

→ *Passo 2 - Criar o Volume Inicial:*

$VI('V1') \rightarrow E1$

→ *Passo 3 - Aplicar a primeira geminação:*

$\tau \leftarrow V('V2', 1)$

$E1 * \tau \rightarrow E2$

→ *Passo 4 - Aplicar a segunda geminação:*

$\tau \leftarrow V('V3', 1)$

$E2 * \tau \rightarrow E3$

→ *Passo 5 - Alterar o Volume Padrão:*

$VP(250, 250, 500, 1, 2)$

→ *Passo 6 - Aplicar a superposição:*

$\tau \leftarrow V('V4', 21)$

$E3 * \tau \rightarrow E4$

A Figura 3.1 mostra o resultado do Exemplo 3.1.

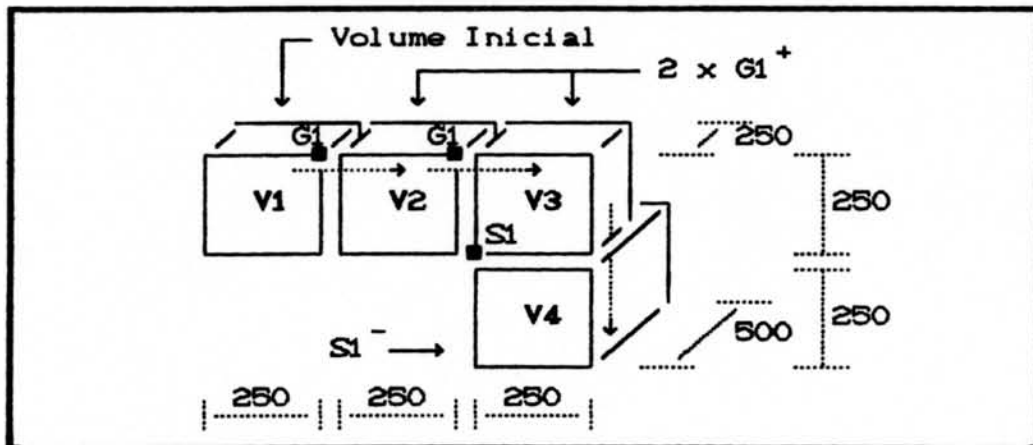


Figura 3.1 - Resultado do Exemplo 3.1.

**Tabela 3.2** - Resumo dos mecanismos e símbolos do capítulo

<b>Comandos para geração de volumes</b>	
VP(Cx, Cy, Cz, T, C)	= volume padrão
VI(NV)	= volume inicial
V(NV, RG)	= gera novo volume
<b>Variáveis</b>	
→ Cx	= dimensão em X do volume
→ Cy	= dimensão em Y do volume
→ Cz	= dimensão em Z do volume
→ T	= código da textura
→ C	= código da cor
→ NV	= nome do volume
→ RG	= código da regra de geração
<b>Outros símbolos</b>	
→ E0, E1, E2, ...	= estágios de projeto
→ $\tau$	= operação de transformação



#### 4. CRIANDO UM MECANISMO DE REPETIÇÃO PARA ACELERAR O PROCESSO DE GERAÇÃO

Como pode ser observado no Exemplo 3.1, apresentado no capítulo anterior, a geração dos volumes V2 e V3, i. é, duas geminações positivas relativas a G1, teve que ser realizada em duas etapas distintas. Outra forma de realizar as mesmas operações, de maneira mais dinâmica (possibilitando maior rapidez na execução do processo gerativo), seria utilizar um mecanismo de repetição automática de operações de transformação  $\sqrt{\text{TUR } 90^\circ}$ . Assim, seria possível gerar vários elementos do projeto, utilizando-se apenas uma operação de transformação de repetição  $\rho$ , o que produziria, obviamente, uma razoável economia de tempo no processo.

A seguir, serão definidas algumas regras importantes para a concepção do mecanismo em questão:

Regra 4.1 - Qualquer uma das operações de acoplamento de volumes poderá ser repetida mediante a utilização da operação de transformação de repetição.

Regra 4.2 - O mecanismo de repetição poderá ser utilizado por um conjunto de regras de geração, para produzir um projeto específico.

Regra 4.3 - O número de repetições, para cada operação de repetição, deverá obedecer a um parâmetro a ser definido na operação de transformação de repetição  $\rho$ .

Portanto, pode-se definir uma operação de

transformação de repetição  $\rho$ , como a seguir:

$\rho \leftarrow \text{VR}(\text{NV}, \text{RG}, \text{NR}) \leftarrow$ Transformação de Repetição
<p>Onde:</p> <p>NV = Nome dos volumes a serem gerados</p> <p>RG = Código da regra de geração a ser utilizada na operação de acoplamento (ver Tabela 3.1) (Regra 4.1)</p> <p>NR = Número de repetições (Regra 4.3)</p>

A seguir, será apresentado um exemplo que ilustra a aplicação da operação de transformação  $\rho$ .

Exemplo 4.1  $\longrightarrow$  Gerar um objeto, a partir do volume inicial  $V_1$ , de mesmas características do Exemplo 3.1, utilizando o seguinte conjunto de regras de geração (Regra 4.2), na sequência solicitada: 3 geminações positivas, 2 justaposições positivas, 3 geminações negativas e 2 justaposições negativas.

$\rightarrow$  Passo 1 - Gerar o volume padrão:

$\text{VP}(250, 250, 250, 1, 2)$

$\rightarrow$  Passo 2 - Criar o volume inicial:

$\text{VI}('V_1') \rightarrow E_1$

$\rightarrow$  Passo 3 - Aplicar as geminações positivas:

$\rho \leftarrow \text{VR}('V_1', 1, 3)$

$E_1 * \rho \rightarrow E_2$

$\rightarrow$  Passo 4 - Aplicar as justaposições positivas:

$\rho \leftarrow \text{VR}('V_1', 9, 2)$

$E_2 * \rho \rightarrow E_3$

→ Passo 5 - Aplicar as geminações negativas:

$\rho \leftarrow \text{VRC}('V1', 5, 3)$

$E3 * \rho \rightarrow E4$

→ Passo 6 - Aplicar as justaposições negativas:

$\rho \leftarrow \text{VRC}('V1', 13, 2)$

$E4 * \rho \rightarrow E5$

O resultado deste exemplo pode ser visto na Figura 4.1.

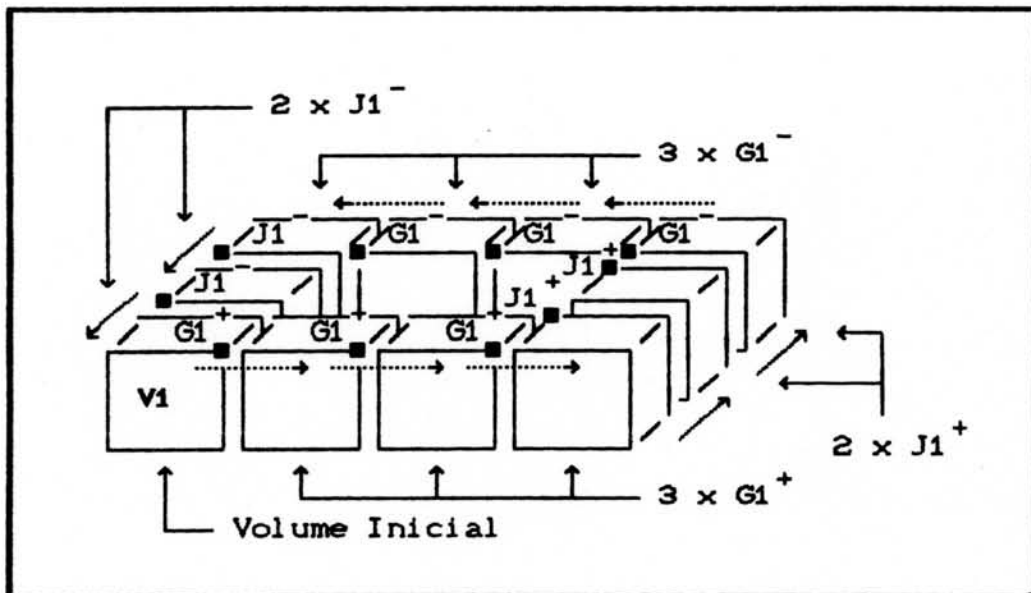


Figura 4.1 - Resultado do Exemplo 4.1.

Tabela 4.1 - Resumo dos mecanismos e símbolos do capítulo

Comandos para geração de volumes	
$\text{VRC}(\text{NV}, \text{RG}, \text{NR})$ = gera volumes com repetição	
Variáveis	
→	NV = nome dos volumes
→	RG = código da regra de geração
→	NR = número de repetições
Outros símbolos	
→	$\rho$ = transformação de geração com repetição

## 5. CRIANDO MECANISMOS ALTERADORES PARA AUMENTAR AS ALTERNATIVAS DE PROJETO

Até aqui, para gerar um objeto, era necessário estabelecer um *volume padrão* e levando em consideração este padrão, gerava-se um *volume inicial* e a partir desta forma, outros volumes eram criados mediante a aplicação de *regras de acoplamento*. Se fosse preciso modificar as dimensões de um dos volumes, isso deveria ser feito antes da sua geração, usando-se o mecanismo de definição do *volume padrão*.

Neste capítulo, serão definidos novos mecanismos que permitirão alterar os volumes após a sua geração. Para tal, será necessário inicialmente, definir algumas regras fundamentais para a concepção dos *mecanismos alteradores*:

Regra 5.1 - Os mecanismos alteradores devem ser utilizados após a aplicação das regras de geração de volumes (acoplamentos), visto que para alterar alguma coisa, é necessário que ela já exista.

Regra 5.2 - Serão necessários diferentes tipos de operações de transformação alteradoras, para permitir maior flexibilidade de projeto.

Regra 5.3 - Os mecanismos alteradores devem permitir sua utilização por meio de um conjunto de regras de geração, possibilitando a sua execução automática a partir das necessidades específicas de projeto.

Regra 5.4 - Ao ser ativado, o mecanismo alterador deverá realizar as operações alteradoras em todos os objetos que possuírem o mesmo parâmetro de individualização, i. é, o mesmo nome.

A seguir, serão definidas as diferentes operações de transformação de alteração  $\alpha$  básicas para o desenvolvimento de projetos, visando atender a exigência estabelecida pela Regra 5.2.

### 5.1 Alterando as dimensões de um volume

Em primeiro lugar, será criada uma operação  $\alpha$ , que permitirá alterar as dimensões dos volumes já criados pelas operações de acoplamento (ver Regra 5.1). Visto que na maioria das vezes, considerando-se as experiências de projeto, será necessário alterar apenas uma ou duas das dimensões dos volumes, os mecanismos em questão serão concebidos de tal maneira que as alterações possam ocorrer de forma independente uma da outra. A seguir, são apresentadas as operações para alteração das dimensões de volumes (paralelepípedos).

$\alpha \leftarrow ACx(NV, NCx)$

↑  
 └─ Transformação Alteradora da  
 Dimensão em X

$\alpha \leftarrow ACy(NV, NCy)$

↑  
 └─ Transformação Alteradora da  
 Dimensão em Y

$$\alpha \leftarrow ACz(NV, NCz)$$

↑ Transformação Alteradora da Dimensão em Z.

Onde:

NV = Nome do volume que sofrerá a alteração

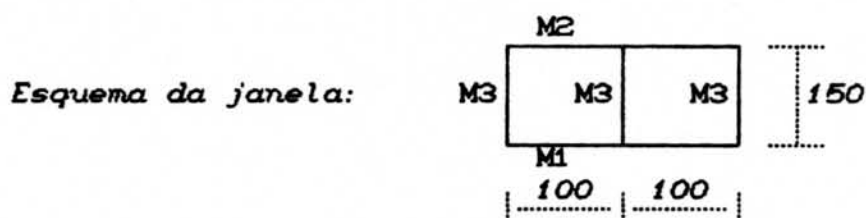
NCx = Nova dimensão em relação ao eixo X

NCy = Nova dimensão em relação ao eixo Y

NCz = Nova dimensão em relação ao eixo Z

A seguir, será apresentado um exemplo ilustrativo das operações  $\alpha$ , definidas até o momento.

Exemplo 5.1 → Gerar os marcos de uma janela (esquemática logo abaixo), usando regras de geração de volumes e mecanismos alteradores (Regra 5.3).



Largura dos marcos = 10

Espessura dos marcos = 5

→ Passo 1 - Gerar o volume padrão:

VP(200,140,5,1,2)

→ Passo 2 - Criar o volume inicial:

VI('M1') → E1

→ Passo 3 - Alterar o volume padrão:

VP(200,10,5,1,2)

→ Passo 4 - Aplicar uma superposição positiva

$S1^+$ :

$\tau \leftarrow VC('M2', 17)$

$E1 * \tau \rightarrow E2$

→ Passo 5 - Alterar a dimensão em Y de M1:

$\alpha \leftarrow ACy('M1', 10)$

$E2 * \alpha \rightarrow E3$

→ Passo 6 - Alterar o volume padrão:

$VP(95, 130, 5, 1, 2)$

→ Passo 7 - Aplicar uma superposição negativa

$S1^-$ :

$\tau \leftarrow VC('M3', 21)$

$E3 * \tau \rightarrow E4$

→ Passo 8 - Aplicar duas geminações positivas

$G1^+$ :

$\rho \leftarrow VRC('M3', 1, 2)$

$E4 * \rho \rightarrow E5$

→ Passo 9 - Alterar as dimensões em X de M3:

$\alpha \leftarrow ACx('M3', 10)$        $\uparrow$  ver Regra 5.4

$E5 * \alpha \rightarrow E6$

Os resultados do Exemplo 5.1 (que é uma forma de geração) podem ser observados nas Figuras 5.1 e 5.2.

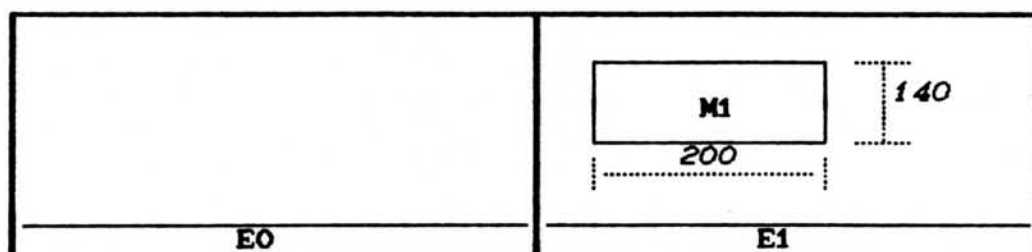


Figura 5.1 - Ilustração esquemática do Exemplo 5.1

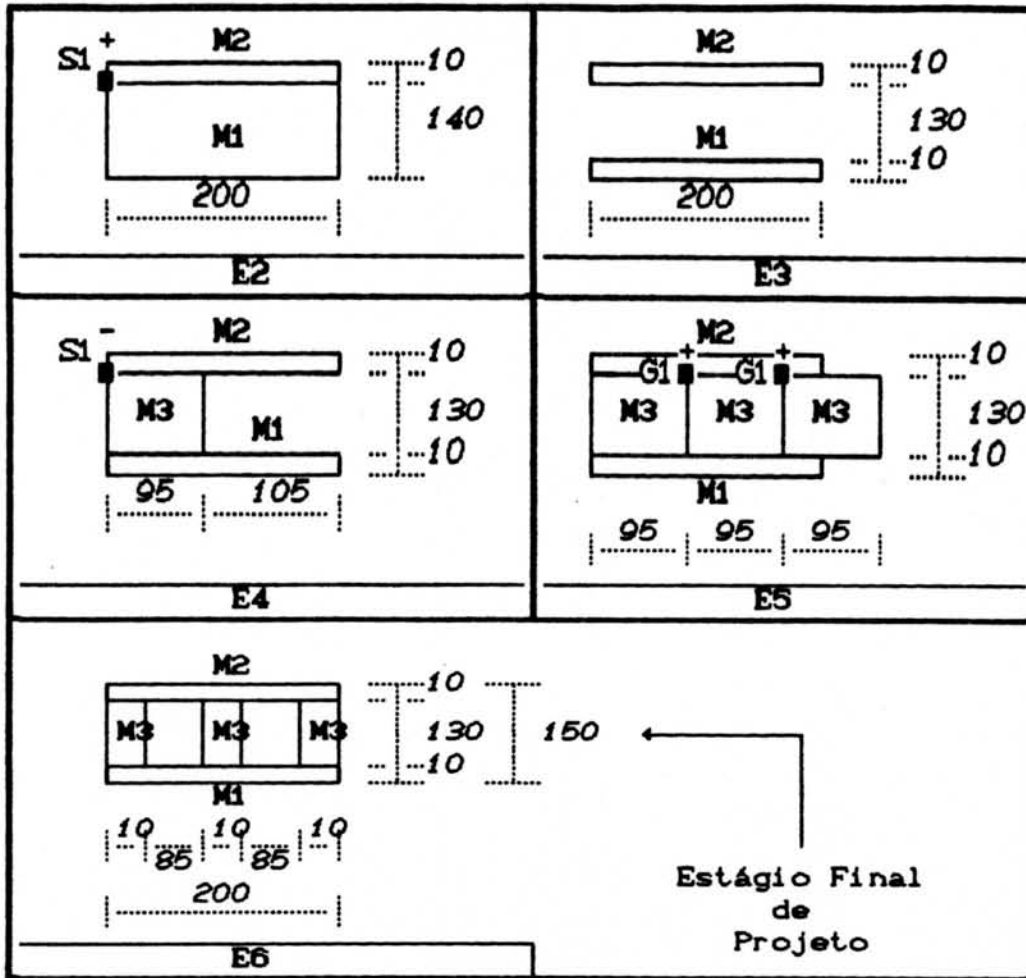


Figura 5.1(cont.) - Ilustração esquemática do Exemplo 5.1

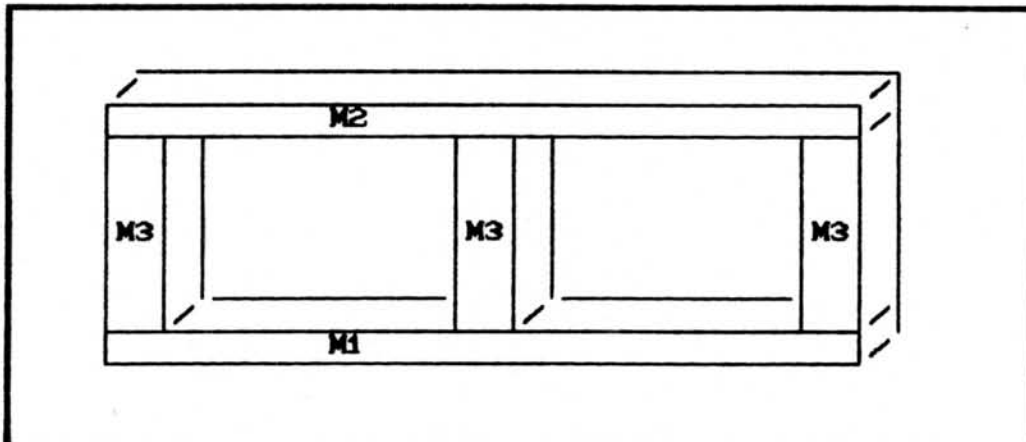


Figura 5.2 - Perspectiva do projeto gerado pela aplicação dos passos definidos no Exemplo 5.1.



## 5.2 Deslocando um volume para outra posição

Em determinados estágios de projeto, haverá a necessidade de *deslocar* um volume em particular, de tal maneira que possa ser atendida uma exigência de projeto. Portanto, deve ser criada uma operação de transformação alteradora de *deslocamento* que possa ser aplicada durante o processo gerativo. Vale salientar, ainda, que as operações de *deslocamento*  $\delta$ , constituem-se num tipo de operação alteradora  $\alpha$ . Assim, será utilizada uma representação diferente, apenas para diferenciar os dois casos das operações de *alteração*.

Visto que um *deslocamento* pode acontecer na direção dos três eixos do sistema de coordenadas, serão necessárias três operações de transformação de *deslocamento*  $\delta$  distintas, para realizar esta tarefa:

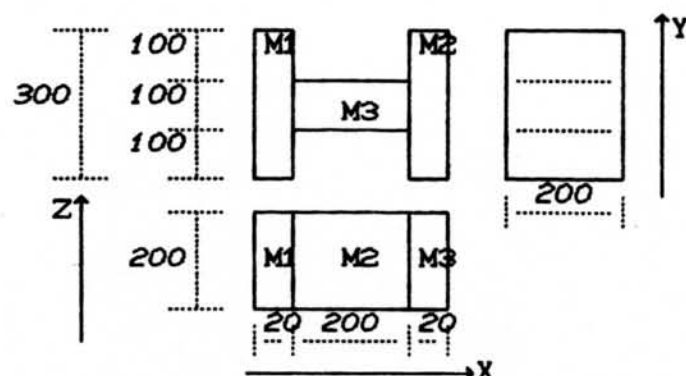
$\delta \leftarrow D_x(NV, D)$ $\uparrow$ Deslocamento na direção do eixo X
$\delta \leftarrow D_y(NV, D)$ $\uparrow$ Deslocamento na direção do eixo Y
$\delta \leftarrow D_z(NV, D)$ $\uparrow$ Deslocamento na direção do eixo Z
Onde: NV = Nome do volume que sofrerá a alteração D = Valor do deslocamento do volume em unidades de comprimento, levando em consideração o seguinte: $\rightarrow$ Deslocamento Positivo (+) $\leftarrow$ No sentido do eixo

→ Deslocamento Negativo (-) ← No sentido oposto ao do eixo

Segue, logo abaixo, um exemplo que mostra o funcionamento das operações  $\delta$ :

Exemplo 5.2 → Gerar o objeto esquematizado a seguir, utilizando as operações de deslocamento  $\delta$ .

Esquema do projeto:



→ Passo 1 - Gerar o volume padrão:

$VP(220, 300, 200, 1, 2)$

→ Passo 2 - Criar o volume inicial:

$VI('M1') \rightarrow E1$

→ Passo 3 - Alterar o volume padrão:

$VP(20, 300, 200, 1, 2)$

→ Passo 4 - Aplicar uma geminação positiva  $G1^+$ :

$\tau \leftarrow VC('M2', 1)$

$E1 * \tau \rightarrow E2$

→ Passo 5 - Alterar a dimensão em X de M1:

$\alpha \leftarrow ACx('M1', 20)$

$E2 * \alpha \rightarrow E3$

→ Passo 6 - Alterar o volume padrão:

$VP(200, 100, 200, 1, 2)$

→ Passo 7 - Aplicar uma geminação negativa  $G2^-$ :

$\tau \leftarrow VC('M3', 6)$

$E3 * \tau \rightarrow E4$

→ Passo 8 - Deslocar M3:

$\delta \leftarrow Dy('M3', 100)$

$E4 * \delta \rightarrow E5$

As Figuras 5.3 e 5.4 mostram os resultados do Exemplo 5.2, cujos passos do processo gerativo foram apresentados logo acima:

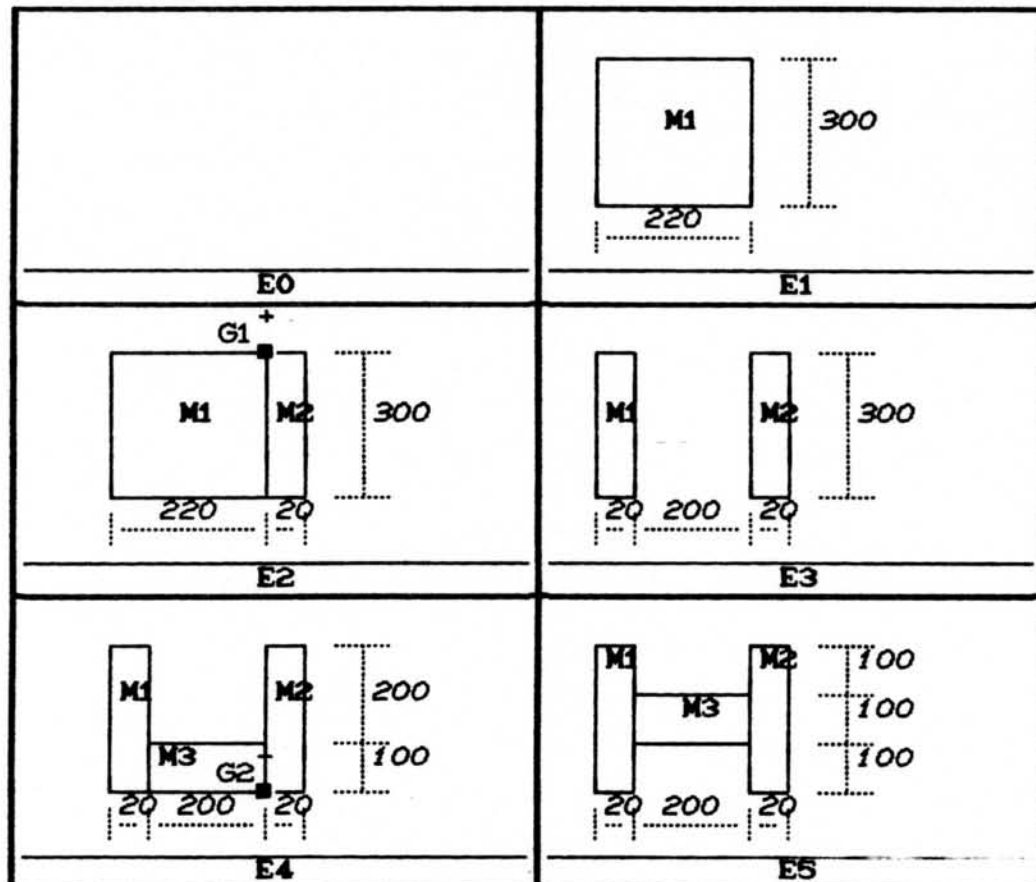


Figura 5.3 - Ilustração esquemática do Exemplo 5.2.

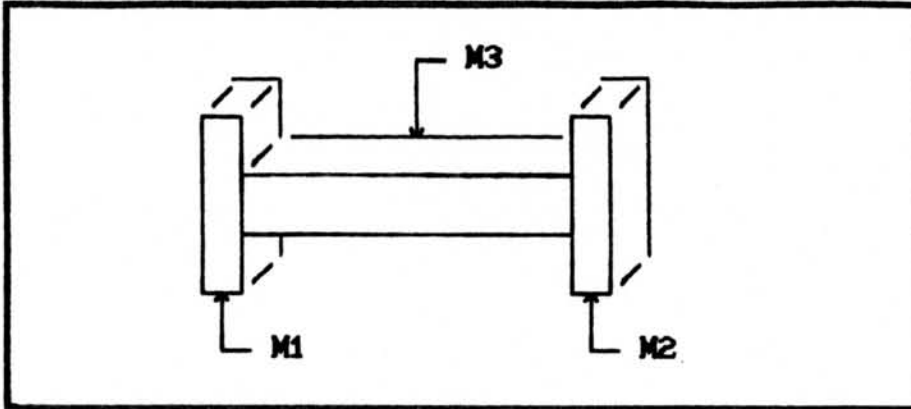
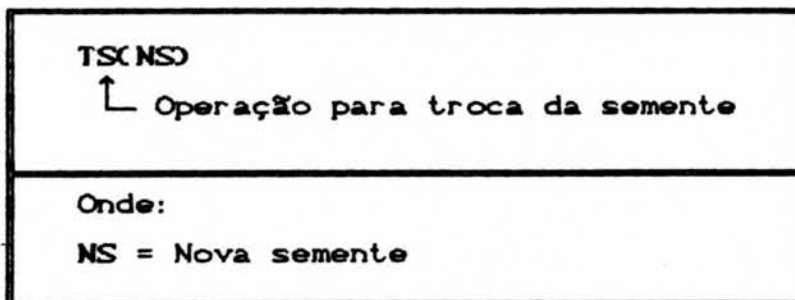


Figura 5.4 - Perspectiva do projeto final do Exemplo 5.2.

### 5.3 Trocando o volume de referência












Algumas vezes, surgirá a necessidade de trocar o volume de referência, para satisfazer a uma necessidade de projeto. Para que isso possa ser feito, deverá existir um mecanismo que realize uma operação de troca da semente, como definido a seguir:



Vale mencionar a importância dessa operação, que proporcionará uma maior flexibilidade durante os estágios do processo projetivo, visto que até aqui, ao ser aplicada uma operação de transformação, esta era realizada sobre o último volume criado. Assim, antes não era possível aplicar uma operação transformadora sobre um volume gerado em outro estágio do projeto.

## 5.4 Alterando a textura e a cor das faces de um volume

**Tabela 5.1** - Códigos de texturas para preenchimento (pintura) das faces dos volumes.

Código	Textura	
1	Cor de fundo	
2	Sólido	
3	Linhas horizontais	
4	Linhas diagonais 1	
5	Linhas diagonais 2	
6	Linhas diagonais 3	
7	Linhas diagonais 4	
8	Rede	
9	Rede diagonal	
10	Pontilhado 1	
11	Pontilhado 2	
12	Pontilhado 3	

**Tabela 5.2** - Códigos de cores.

Código	Cor	Código	Cor
1	Preto	9	Cinza Escuro
2	Azul	10	Azul Claro
3	Verde	11	Verde Claro
4	Violeta	12	Violeta Claro
5	Vermelho	13	Vermelho Claro
6	Rosa	14	Rosa Claro
7	Marron	15	Amarelo
8	Cinza Claro	16	Branco

Quando definiu-se a operação VP (Volume Padrão), criou-se dois parâmetros que estabeleciam uma *textura* e uma *cor* para o preenchimento das faces dos volumes, de forma a permitir uma aparência mais realista de um projeto. Assim, seria conveniente criar um mecanismo para *alterar a textura e a cor* dos volumes. Estes mecanismos serão definidos a seguir:

AT(NV, NT, NC)

↑  
 ↳ Mecanismo que altera a *textura* e a *cor* de um volume já criado

Onde:

NV = Nome do volume que sofrerá a alteração

NT = Código da nova *textura* para preenchimento das faces do volume (ver *Tabela 5.1*)

NC = Código da nova *cor* (ver *Tabela 5.2*)

## 5.5 Eliminação de um volume

Uma operação bastante útil para o processo de projeto, é aquela que permite *eliminar* um volume já criado, para possibilitar a aplicação de uma necessidade de projeto específica, tal como, criar um espaço vazio entre os volumes /TUR 90/. Além disso, esta operação de transformação de *eliminação*, permite *corrigir* uma operação executada de forma indesejável, pois, em muitas ocasiões durante os estágios de projeto, haverá a necessidade de modificar algumas partes, para melhorar o seu aspecto estético, ou mesmo para permitir a inclusão de um novo volume em uma posição específica, para satisfazer a uma determinada *necessidade funcional* do projeto.

Assim, a operação de transformação de *eliminação*  $\epsilon$  pode ser definida:

$\epsilon$  ← EVC(NV)

↑  
 ↳ Mecanismo para *Eliminação* de um volume

Onde:

NV = Nome do volume a ser eliminado

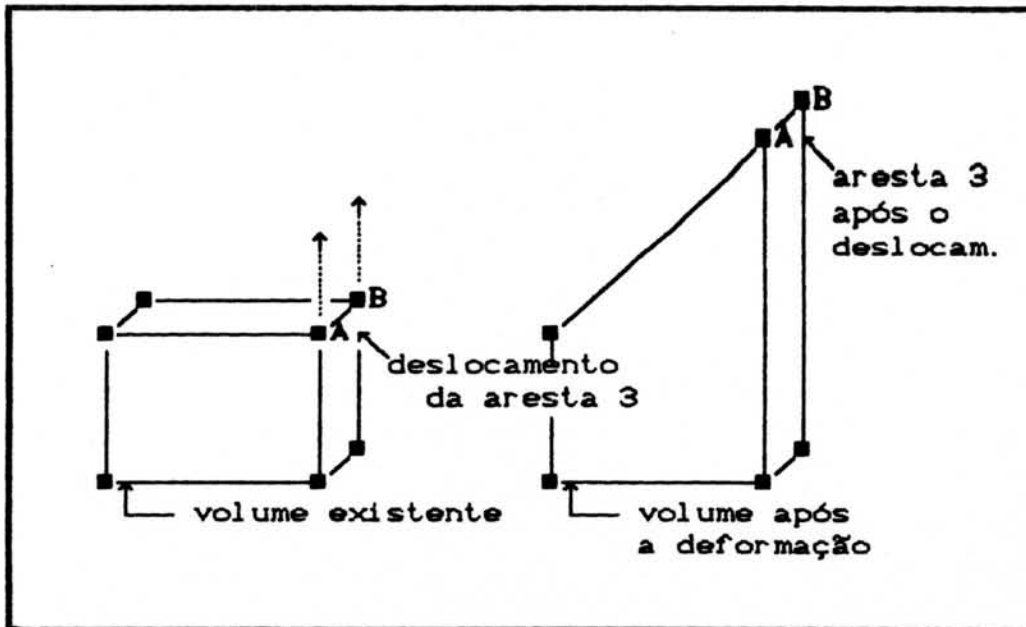


Figura 5.5 - Deformação de um paralelepípedo.

## 5.6 Deformação de um volume existente

Após gerar um volume (lembrar que todo volume é gerado na forma de um *paralelepípedo* que possui as dimensões do *volume padrão*), deve ser possível alterar a sua forma para atender a uma necessidade de projeto. Assim, torna-se possível gerar outros tipos de figuras geométricas tridimensionais, através de uma *deformação* aplicada sobre o *paralelepípedo* existente. Esta operação alteradora funciona pelo *deslocamento* de algumas arestas específicas, escolhidas convenientemente de maneira a proporcionar o modelamento da figura geométrica na forma desejada. A Figura 5.5 mostra um exemplo ilustrativo dos conceitos até aqui estabelecidos. Note-se na Figura 5.5 que, por meio de uma deformação, torna-se possível obter arestas inclinadas

em relação aos eixos do sistema de coordenadas.

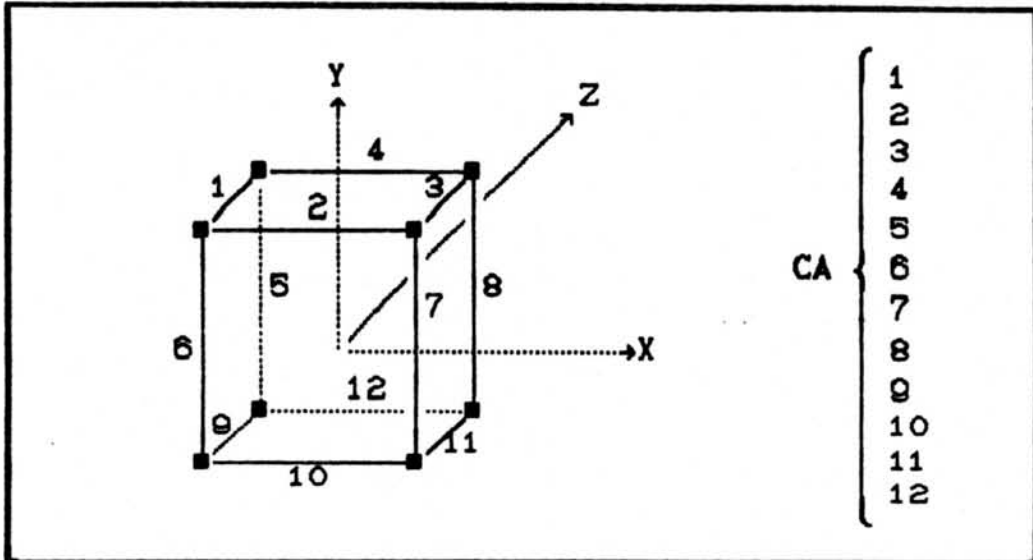


Figura 5.6 - Códigos das arestas em um volume.

A seguir será formalmente definida a operação alteradora de *deformação*  $\Delta$  de um volume já existente no universo de trabalho.

$\Delta \leftarrow$  DVC(NV, CA, DX, DY, DZ)

↑ Mecanismo para *deformar* um volume já existente

Onde:

NV = Nome do volume a ser deformado

CA = Código da aresta a ser deslocada  
(ver Figura 5.6)

DX = Valor do deslocamento no eixo X

DY = Valor do deslocamento no eixo Y

DZ = Valor do deslocamento no eixo Z

Vale salientar, ainda, que esta operação de transformação de alteração permite, entre outras coisas, a criação de elementos de projeto especiais (com arestas não



paralelas aos eixos do sistema de coordenadas), tais como: telhados, rampas e outros que podem ser construídos pela combinação de dois ou mais elementos primários. O Exemplo 5.2 ilustra uma utilização da operação  $\Delta$  na criação de uma árvore.

Exemplo 5.3  $\longrightarrow$  Gerar uma árvore, utilizando um mecanismo de deformação  $\Delta$ . O resultado pode ser observado no ANEXO 1 no final deste trabalho.

- $\rightarrow$  Passo 1 -  $VP(75, 5, 75, 10, 3)$
- $\rightarrow$  Passo 2 -  $VI('A1')$   $\rightarrow$  E1
- $\rightarrow$  Passo 3 -  $\Delta \longleftarrow DVC('A1', 3, 0, 100, 0)$   
E1  $\ast \Delta \rightarrow$  E2
- $\rightarrow$  Passo 4 -  $\Delta \longleftarrow DVC('A1', 4, 0, 100, 0)$   
E2  $\ast \Delta \rightarrow$  E3
- $\rightarrow$  Passo 5 -  $\tau \longleftarrow VC('A2', 2)$   
E3  $\ast \tau \rightarrow$  E4
- $\rightarrow$  Passo 6 -  $\Delta \longleftarrow DVC('A2', 1, 0, 100, 0)$   
E4  $\ast \Delta \rightarrow$  E5
- $\rightarrow$  Passo 7 -  $\Delta \longleftarrow DVC('A2', 4, 0, 100, 0)$   
E5  $\ast \Delta \rightarrow$  E6
- $\rightarrow$  Passo 8 -  $\tau \longleftarrow VC('A3', 10)$   
E6  $\ast \tau \rightarrow$  E7
- $\rightarrow$  Passo 9 -  $\Delta \longleftarrow DVC('A3', 1, 0, 100, 0)$   
E7  $\ast \Delta \rightarrow$  E8
- $\rightarrow$  Passo 10 -  $\Delta \longleftarrow DVC('A3', 2, 0, 100, 0)$   
E8  $\ast \Delta \rightarrow$  E9
- $\rightarrow$  Passo 11 -  $\tau \longleftarrow VC('A4', 6)$   
E9  $\ast \tau \rightarrow$  E10
- $\rightarrow$  Passo 12 -  $\Delta \longleftarrow DVC('A4', 2, 0, 100, 0)$   
E10  $\ast \Delta \rightarrow$  E11
- $\rightarrow$  Passo 13 -  $\Delta \longleftarrow DVC('A4', 3, 0, 100, 0)$   
E11  $\ast \Delta \rightarrow$  E12
- $\rightarrow$  Passo 14 -  $VP(20, 200, 20, 7, 7)$
- $\rightarrow$  Passo 15 -  $\tau \longleftarrow VC('T', 22)$

$E12 * \tau \rightarrow E13$   
 $\rightarrow$  Passo 16 -  $\delta \leftarrow D_x('T', 10)$   
 $E13 * \delta \rightarrow E14$   
 $\rightarrow$  Passo 17 -  $\delta \leftarrow D_z('T', -10)$   
 $E14 * \delta \rightarrow E15$

## 5.7 Rotação de um volume

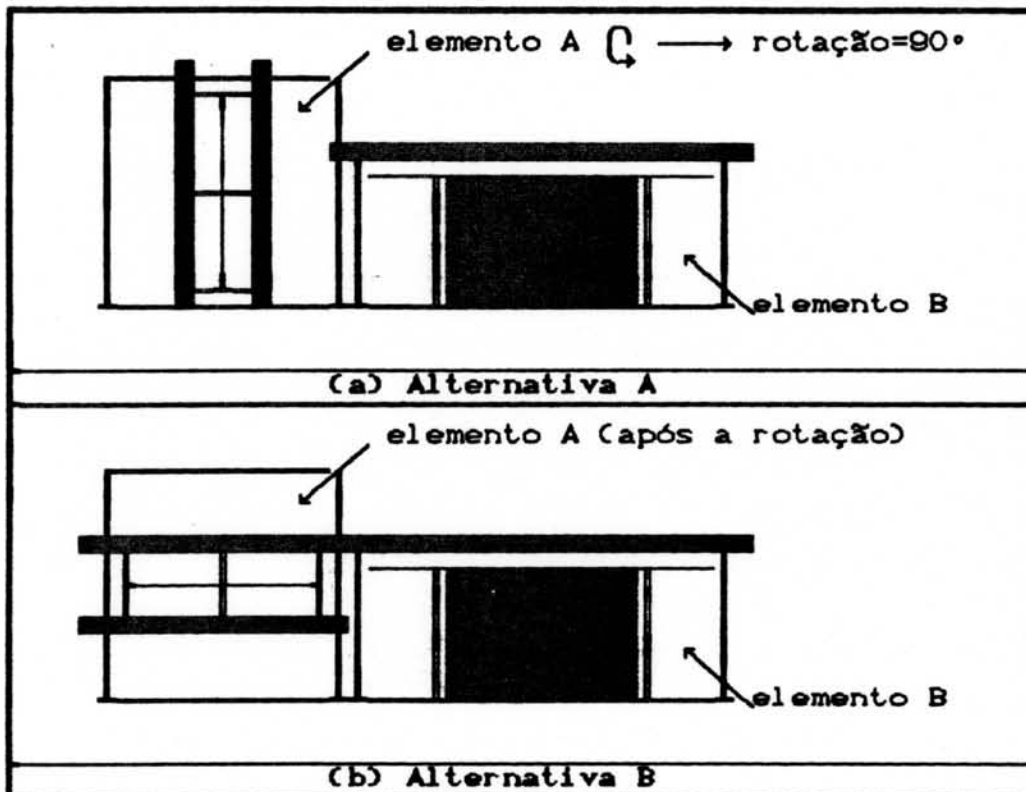
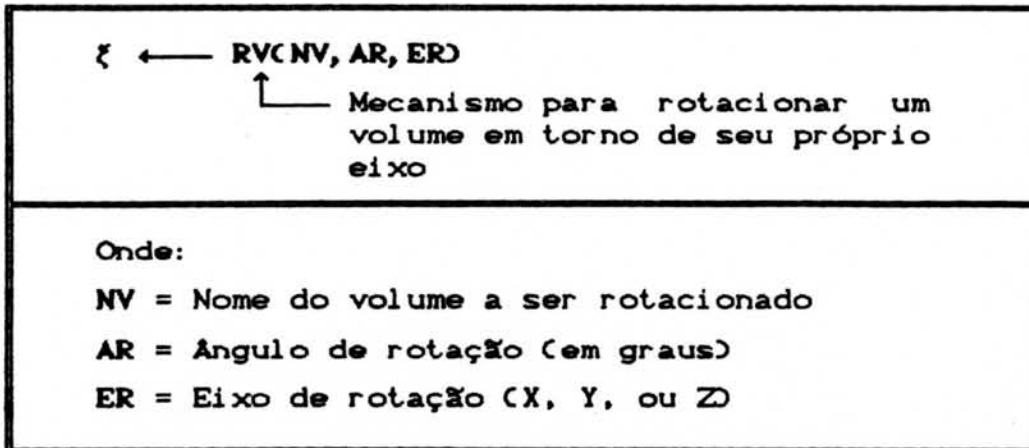


Figura 5.7 - Duas alternativas de um projeto.

Em algumas ocasiões, durante as etapas do processo projetivo, surge a necessidade de alterar a posição de um volume sem modificar as suas dimensões, mantendo-se assim, o seu aspecto geométrico. Isto pode ser feito através de uma operação alteradora que permita rotacionar o volume em torno de seu próprio eixo. Assim, torna-se possível obter diferentes alternativas de visualização para um mesmo projeto, por meio de uma simples

rotação de uma ou mais partes, como acontece na utilização do "LEGO" de madeira de Turkienicz (TUR 90). A Figura 5.7 mostra um exemplo de um projeto arquitetônico, composto por dois elementos (A e B). Note-se que são ilustradas duas alternativas distintas de projeto, sendo que o aspecto visual dessas alternativas se apresenta completamente modificado com a simples rotação do elemento A.

A operação alteradora de rotação  $\xi$  pode ser formalmente representada:



### 5.8 Individualizando volumes de mesmo nome para permitir alterações somente em alguns deles

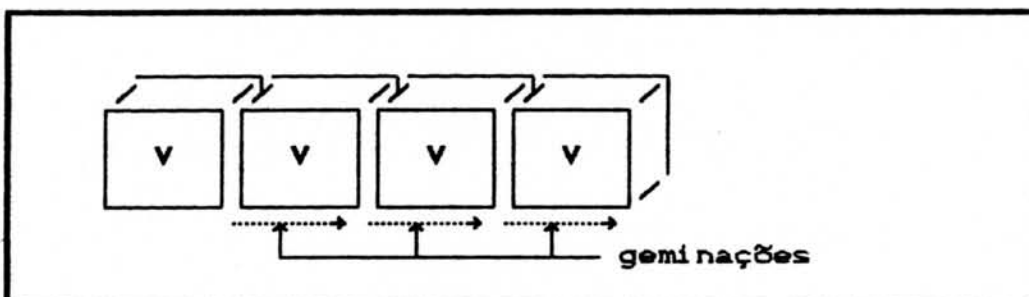


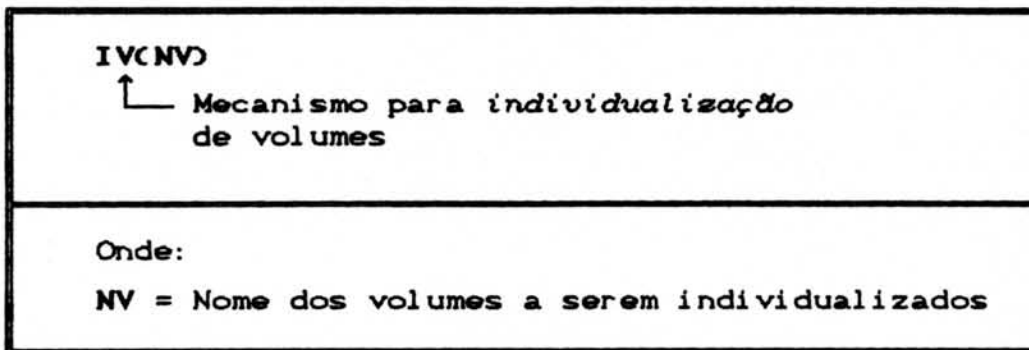
Figura 5.8 - Quatro volumes gerados com o mesmo nome (V).

Ao ser observada a Figura 5.8, pode-se concluir

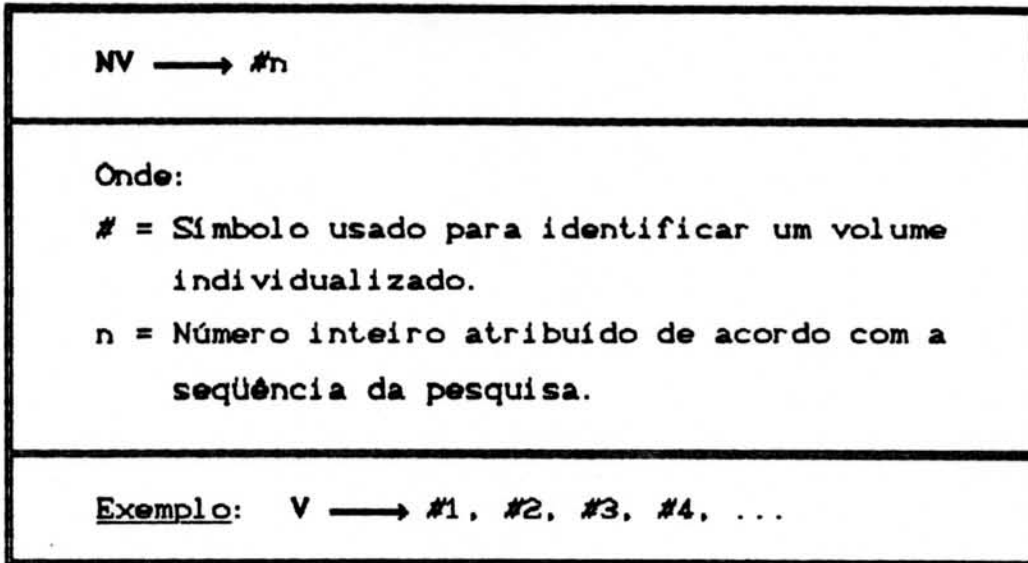
que um volume inicial ('V') foi criado e, a seguir, mais três volumes foram gerados pela aplicação de três geminações positivas (ambos os volumes com o mesmo nome, igual ao nome do volume inicial = 'V').

Pode ocorrer que após gerar os volumes, surja uma necessidade de alteração em apenas um deles. Nesse caso, se não houver um mecanismo que permita *individualizar* os volumes gerados, de mesmo nome, uma operação de alteração aplicada ao volume 'V' provocará a tal alteração em *todos* os quatro volumes gerados.

Assim, pode-se definir o mecanismo que permitirá *individualizar* volumes:



A operação IV, definida acima, provocará uma alteração no nome de cada um dos volumes (cujos nomes sejam iguais a NV), de forma a permitir o manuseio de cada um deles separadamente. Para tal, deverá ocorrer uma pesquisa relativa aos nomes dos volumes gerados até o momento, de forma a verificar quais os volumes que devem ser *individualizados*. Assim, ao encontrar um volume com nome igual a NV, o sistema deverá provocar uma alteração do nome conforme mostrado a seguir:



A Figura 5.9 ilustra como seria executada uma alteração no segundo volume do exemplo da Figura 5.8.

Da mesma forma, após a execução das alterações que se fizerem necessárias, deve haver um mecanismo que permita o retorno do nome dos volumes à condição inicial, se isso for conveniente para o desenvolvimento das etapas seguintes do processo de projeto.

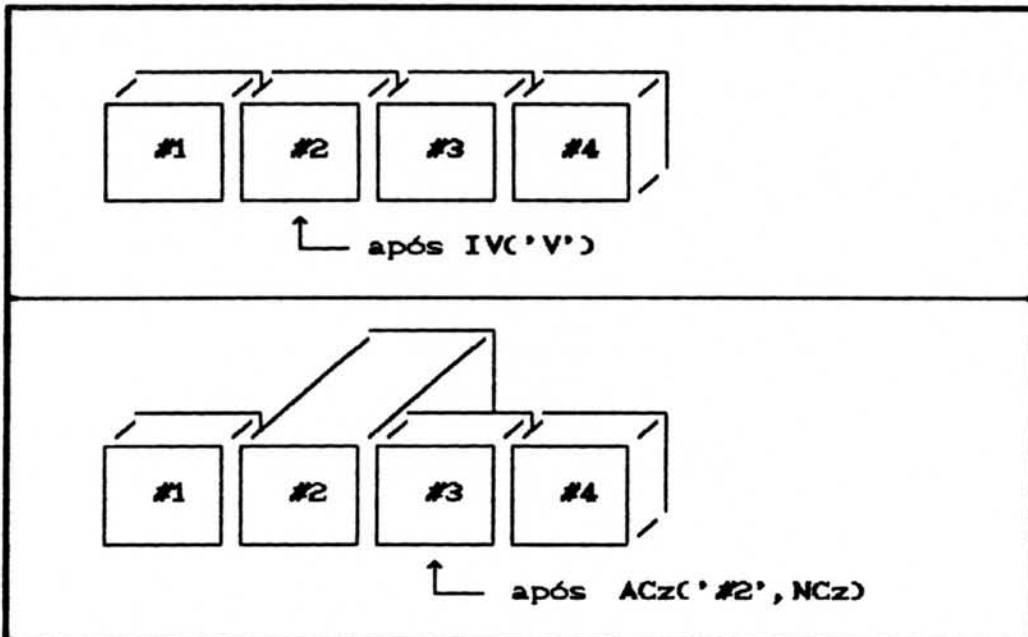


Figura 5.9 - Uma alteração em um dos volumes individualizados.

RN



Mecanismo para recuperar o nome original dos volumes individualizados.

Assim, para o exemplo da Figura 5.8, após a execução da operação RN, todos os volumes voltarão a ter o nome 'V', permitindo dessa forma, a aplicação de um mesmo tratamento operacional a todos os quatro volumes *simultaneamente*.

### 5.9 Alterando o nome de um volume

Um mecanismo simples, mas de grande utilidade prática, deve ser definido para permitir *alterar* o nome de um volume já existente no universo de trabalho. A utilização do mecanismo que será definido formalmente nesta seção, permitirá modificar um nome de volume para outro que seja mais conveniente para a seqüência dos estágios do processo projetivo. Além disso, permitirá *agrupar* objetos criados individualmente, de forma a proporcionar a aplicação de um mesmo tratamento alterador a todos *simultaneamente*, como se fossem apenas um (o processo inverso constitui-se na *individualização*).

ANCNA, NND



Mecanismo para alterar o nome de um volume já existente no universo.

Onde:

NA = Nome atual

NN = Novo nome

Como observação, vale mencionar que, caso exista mais de um volume com nome igual a NA, todos sofrerão a alteração AN.

Tabela 5.3 - Resumo dos mecanismos e símbolos do capítulo

Comandos para alteração de volumes	
ACx(NV, NCx)	= altera dimensão em X
ACy(NV, NCy)	= altera dimensão em Y
ACz(NV, NCz)	= altera dimensão em Z
Dx(NV, D)	= deslocamento em X
Dy(NV, D)	= deslocamento em Y
Dz(NV, D)	= deslocamento em Z
TSC(NS)	= troca semente
AT(NV, NT, NC)	= altera textura
EVC(NV)	= elimina volumes
DVC(NV, CA, DX, DY, DZ)	= deforma volumes
RVC(NV, AR, ER)	= rotaciona volumes
IV(NV)	= individualiza volumes
RN	= recupera volumes
ANCNA, NND	= altera nomes
Variáveis	
→ NV	= nome do volume
→ NCx	= nova dimensão em X
→ NCy	= nova dimensão em Y
→ NCz	= nova dimensão em Z
→ D	= valor do deslocamento
→ NS	= nome da nova semente
→ NT	= código da nova textura
→ NC	= código da nova cor
→ CA	= código da aresta a ser deslocada
→ DX	= deslocamento da aresta em X
→ DY	= deslocamento da aresta em Y
→ DZ	= deslocamento da aresta em Z
→ AR	= ângulo de rotação (em graus)
→ ER	= eixo de rotação (X, Y ou Z)
→ NA	= nome atual
→ NN	= novo nome
Outros símbolos	
→ $\alpha$	= transformação alteradora de dimensões
→ $\delta$	= transformação de deslocamento de volumes
→ $\epsilon$	= transformação de eliminação
→ $\Delta$	= transformação de deformação
→ $\xi$	= transformação de rotação

## 6. ACOPLAMENTO DE UNIVERSOS

A geração de volumes, utilizando-se os mecanismos até aqui definidos, pode se tornar um processo bastante lento, visto que muitos estágios podem ser necessários para realizar as operações necessárias para se alcançar um resultado desejado. Como exemplo disso, pode-se citar o Exemplo 5.3, apresentado no capítulo anterior. Observe-se que nesse exemplo, foram necessários 17 *passos* para se gerar o elemento *árvore*. É lógico que quanto maior o número de passos para se atingir o resultado final de um projeto, mais lento torna-se o processo gerativo, o que pode se constituir numa grande barreira para a concepção de projetos utilizando-se um processo computacional.

Deve-se, portanto, criar uma forma para agilizar o processo, utilizando mecanismos que permitam a geração de volumes de maneira rápida, i. é, reduzindo-se o número de estágios necessários para a concepção de um projeto. Uma forma de obter uma redução significativa, seria o aproveitamento de volumes já criados e que se repetem em outros estágios de projeto, o que pode ser observado mais uma vez no Exemplo 5.3. Observe-se que:

→ para o Exemplo 5.3:  
*passos* 2,3,4  $\cong$  *passos* 5,6,7  $\cong$   
 $\cong$  *passos* 8,9,10  $\cong$  *passos* 11,12,13

Assim, se existisse um mecanismo para o aproveitamento dos volumes gerados pelos *passos* 2, 3 e 4, seria obtida uma economia de 9 *passos*, o que se constitui numa redução significativa, visto que o número de passos



necessários para atingir o final do processo de projeto seria reduzido de 17 para 8 passos, i. é, seria obtida uma redução na ordem de 53% (para o Exemplo 5.3). Note-se, que dependendo do projeto, uma economia ainda maior poderia ser alcançada (caso dos projetos que seguem um estilo pré-definido, com muitos elementos repetidos).

A seguir, serão apresentadas algumas regras de fundamental importância para a definição do mecanismo em questão:

Regra 6.1 - Deve haver um meio de "salvar" um conjunto de volumes em um meio magnético, para posterior utilização.

Regra 6.2 - Deve haver um mecanismo que permita "chamar" um conjunto de volumes previamente armazenados em um meio magnético, posicionando-os adequadamente no universo de trabalho.

Regra 6.3 - O posicionamento dos volumes deve seguir as regras estabelecidas para as operações de acoplamento de volumes.

Para ilustrar a Regra 6.1, considere-se o Exemplo 5.3. Observe-se que é possível gerar uma parte da árvore individualmente e armazená-la em um arquivo de disco, i. é, os passos 1, 2, 3 e 4 podem ser aplicados inicialmente. Feito isso, o volume gerado é "salvo" para posterior utilização. Observe-se que o volume foi armazenado como um universo individualizado.

Utilizando-se um mecanismo que use o Regra 6.2,

pode-se "chamar" a cópia fiel do volume que foi armazenada anteriormente como um universo, e acoplá-la ao volume existente no universo de trabalho, de acordo com as regras de acoplamento de volumes, i. é, *geminando, justapondo e superpondo*, de forma a obter a composição final do projeto. Para o caso do Exemplo 5.3, serão necessárias, na devida ordem: *uma geminação positiva (mais uma operação de rotação de 90° em torno do eixo Y), uma justaposição positiva (seguida de uma rotação de 180° em torno do eixo Y) e uma geminação negativa (seguida de uma operação de rotação de 270° em torno do eixo Y)*, para completar o projeto da parte superior da árvore (Regra 6.3). Os passos 14 a 17 devem ser realizados a seguir. Note-se, que foram necessários mais 3 passos para realizar as operações de rotação. Mesmo assim, obteve-se uma significativa redução no número de passos durante o processo, de 17 para 11, i. é, obteve-se uma redução de 35%.

A seguir, será apresentada a definição do mecanismo para acoplamento de universos  $\theta$ :

$\theta \leftarrow \text{AUCNU, NV, TA}$

↑  
Mecanismo para acoplamento de universos

Onde:

NU = Nome do universo a ser acoplado

NV = Nome dos volumes do universo a ser acoplado, dentro do universo de trabalho atual

TA = Tipo de acoplamento a ser utilizado (ver Tabela 6.1)

Tabela 6.1 - Códigos para o parâmetro TA, a serem utilizados no mecanismo para *acoplamento de universos  $\theta$* .

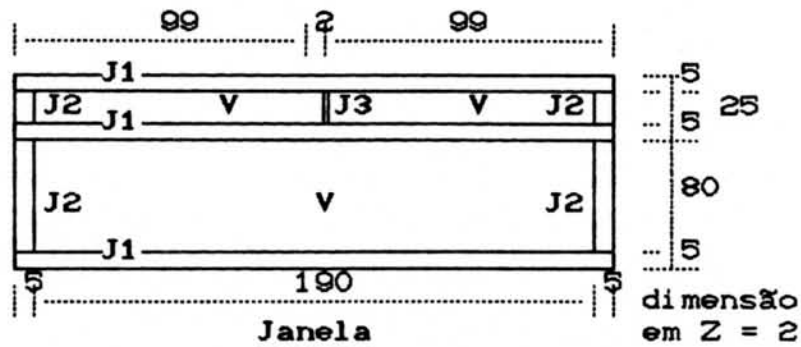
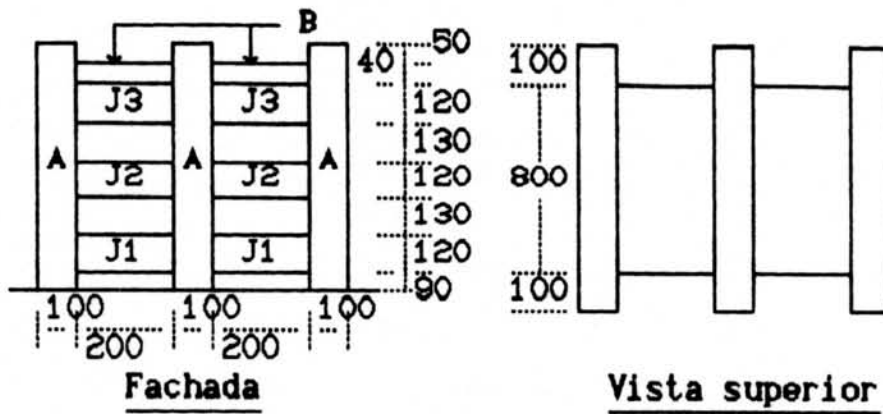
Tipo de acoplamento	TA
Geminação +	1
Geminação -	2
Justaposição +	3
Justaposição -	4
Superposição +	5
Superposição -	6

Note-se, que o mecanismo  $\theta$  poderia ser utilizado para gerar bibliotecas de universos que poderiam ser re-utilizadas em diferentes projetos. Assim, o elemento ilustrativo *drvore*, poderia ser gerado e armazenado em um meio magnético. Dessa forma, toda vez que o projetista necessitasse inserir um elemento *drvore* em um projeto, seria suficiente uma simples chamada ao universo existente nos arquivos magnéticos, aliada ao parâmetro de definição do tipo de acoplamento a ser executado. Além disso, outros elementos poderiam ser gerados e armazenados como parte integrante de uma biblioteca de elementos de projeto.

A seguir, o Exemplo 6.1 ilustra uma utilização prática dos conceitos apresentados neste capítulo.

Exemplo 6.1  $\longrightarrow$  Gerar o edifício e a janela esquematizados a seguir e armazená-los em um meio magnético. A seguir, construir o projeto final do edifício, inserindo 2 elementos janela por andar.

Esquema do projeto:



→ *Geração do Edifício:*

- Passo 1 - VP(300, 800, 1000, 11, 7)
- Passo 2 - VI('A')
- Passo 3 - VR('A', 1, 2)
- Passo 4 - ACx('A', 100)
- Passo 5 - IV('A')
- Passo 6 - VP(200, 750, 800, 2, 5)
- Passo 7 - TSC('#1')
- Passo 8 - VR('B', 2, 1)
- Passo 9 - TSC('#2')
- Passo 10 - VR('B', 2, 1)
- Passo 11 - Dz('B', 100)
- Passo 12 - RN
- Passo 13 - Salvar universo com nome 'EDIF'

→ *Geração da janela:*

- Passo 1 - VP(200, 5, 2, 7, 16)
- Passo 2 - VI('J1')
- Passo 3 - VR('J1', 17, 2)

- Passo 4 - IVC('J1')
- Passo 5 - Dy('#3',105)
- Passo 6 - Dy('#2',80)
- Passo 7 - VP(5,80,2,7,16)
- Passo 8 - TSC('#1')
- Passo 9 - VRC('J2',17,1)
- Passo 10 - VP(190,80,2,1,2)
- Passo 11 - VRC('V',1,1)
- Passo 12 - VP(5,80,2,7,16)
- Passo 13 - VRC('J2',1,1)
- Passo 14 - TSC('#2')
- Passo 15 - VP(5,25,2,7,16)
- Passo 16 - VRC('J2',17,1)
- Passo 17 - VP(94,25,2,1,2)
- Passo 18 - VRC('V',1,1)
- Passo 19 - VP(2,25,2,1,2)
- Passo 20 - VRC('J3',1,1)
- Passo 21 - VP(94,25,2,1,2)
- Passo 22 - VRC('V',1,1)
- Passo 23 - VP(5,25,2,7,16)
- Passo 24 - VRC('J2',1,1)
- Passo 25 - RN
- Passo 26 - Salvar universo com nome 'JANELA'

→ Janelas (1):

- Passo 1 - Chamar universo 'JANELA'
- Passo 2 - AUC('JANELA','J3',5)
- Passo 3 - AUC('JANELA','J1',6)
- Passo 4 - ANC('J1','J2')
- Passo 5 - ANC('V','J2')
- Passo 6 - ANC('J3','J2')
- Passo 7 - Dy('J3',130)
- Passo 8 - Dy('J1',-130)
- Passo 9 - Salvar universo com nome 'JAN'

→ Janelas (2):

- Passo 1 - AUC('JAN', 'JAN', 1)
- Passo 2 - Dx('JAN', 100)
- Passo 3 - ANK('J1', 'JAN')
- Passo 4 - ANK('J2', 'JAN')
- Passo 5 - ANK('J3', 'JAN')
- Passo 6 - Salvar universo com nome 'JAN'

—→ Projeto final:

- Passo 1 - Chamar universo 'EDIF'
- Passo 2 - AUC('JAN', 'JAN', 4)
- Passo 3 - Dx('JAN', 100)
- Passo 4 - Dy('JAN', 90)
- Passo 5 - Dz('JAN', 100)

Os resultados deste exemplo podem ser observados no ANEXO 2 deste trabalho.

Tabela 6.2 - Resumo dos mecanismos e símbolos do capítulo

Comando para manipulação de volumes	
AUCNU, NV, TA) = acoplamento de universos	
Variáveis	
—→	NU = nome do universo a ser acoplado
—→	NV = nome dos volumes do universo a ser acoplado, dentro do universo de trabalho atual
—→	TA = código do tipo de acoplamento
Outros símbolos	
—→	$\theta$ = operação de acoplamento

## 7. UTILIZAÇÃO DE REGRAS DE GERAÇÃO PARA AGILIZAR OS ESTÁGIOS DO PROCESSO PROJETIVO

Um projeto pode apresentar muitas representações ou descrições diferentes na passagem do desenvolvimento conceitual para a realização formal, análise e produção. Assim, no desenvolvimento de representações geométricas e propriedades físicas para uso em análise, o fato dessas representações se referirem a todas as classes de projeto não tem sido bem explorado. Portanto, por exemplo, no desenvolvimento de modelos sólidos, a ênfase tem sido relativa à criação de uma descrição detalhada de um *projeto simples*, mais do que prover ferramentas para a exploração de *classes* de possíveis projetos /ROO 87/. Além disso, um elemento de projeto pode corresponder a uma variedade de possíveis alternativas na sequência de descrição, visto que os diferentes projetos são descritos pelas diferenças de suas características *espaciais, geométricas e funcionais* em suas possíveis partes.

Ao projetar-se alguma coisa, existe um fator de fundamental importância para o processo projetivo. A *idéia de composição* é central para a concepção de novos projetos. Além disso, a geração de volumes em um projeto apresenta duas áreas de concentração /ROO 87/:

<sup>1</sup> → Definição de um *vocabulário de elementos*, i. é, coleções relativamente simples de características formais, como por exemplo: *vigas de edifício, rodas de automóvel ou as poltronas de um avião comercial*.

2 → Condições ou restrições de como esses elementos podem ser colocados juntos ou compostos. As restrições consistem numa tentativa de definir tipos de composições dos elementos que são permissíveis.

Neste capítulo, será apresentada a definição de um vocabulário de elementos /TUR 90/ /MAK 87/, gerado a partir da aplicação das regras de acoplamento de volumes e dos mecanismos alteradores.

## 7.1 Geração de um vocabulário de elementos

Inicialmente, será gerado um novo mecanismo para agilizar o processo de projeto, estabelecendo um conjunto de regras de geração de volumes, e definindo uma sequência de operações cuja execução possa ser simultânea, i. é, estabelecida a sequência, o processo ocorre de forma automática, sem a interferência do operador. Note-se, que esta capacidade de automação permite uma grande rapidez na construção de volumes, visto que com apenas um comando de ação, vários estágios de projeto podem ser executados para alcançar um objetivo pré-definido.

Nos capítulos anteriores, a geração de um projeto era realizada por meio da aplicação de regras de acoplamento de volumes a partir de um volume inicial, cujas dimensões e textura eram definidas pelo volume padrão. Além disso, era possível a execução de alterações de maneira conveniente, de forma a possibilitar a obtenção de uma exigência de projeto. Note-se, que eram necessários vários estágios para o processo de projeto atingir o resultado final.



O objetivo deste capítulo é estabelecer um mecanismo que permita *automatizar* a geração de volumes, de acordo com determinadas *regras de ações* de projeto. Dessa forma, será preciso estabelecer inicialmente uma linguagem de ações /MAK 87/, que permita uma certa interação com o ambiente computacional e que, além disso, proporcione uma forma de tornar o processo visível. Assim, o projetista, de posse de um editor de vocabulário /MAK 87/, poderá estabelecer a *seqüência de execução das ações* de projeto. Essa seqüência, deve então, ser armazenada em um arquivo de ações, para posterior utilização pelo projetista que, de posse de um *cardápio de elementos* /TUR 90/ /MAK 87/, poderá escolher uma das opções disponíveis (essas opções deverão ser apresentadas sob a forma de ícones, cuja finalidade específica será facilitar a tarefa de escolha).

### 7.1.1 A linguagem de ações

A *linguagem de ações* pode ser vista como uma parte integral da experiência de projeto, da representação do processo e da eventual formação e preservação de um projeto. Assim, o projetista, utilizando-se da linguagem de ações pode agir e refletir sobre o processo e até mesmo compartilhar esse processo com outros profissionais. Uma definição apropriada para a linguagem de ações é apresentada a seguir:

Linguagem de Ações - É uma coleção de ações definidas por uma operação de entrada que o editor de vocabulário reconhece como uma intenção de projeto para criar e manipular elementos do vocabulário /MAK 87/.

Note-se que o mecanismo construtor de um estágio de um elemento do vocabulário é um procedimento paramétrico, onde os argumentos representam as *propriedades de projeto*, i. é, *propriedades de dimensionamento e proporcionamento* de um elemento do vocabulário que o projetista precisa manipular. Assim, a *linguagem de ações* deve intervir entre as atribuições desses argumentos de projeto e a construção de um estágio /MAK 87/.

Os comandos da linguagem de ações podem ser definidos em três tipos principais:

- |   |  |
|---|--|
| 1 | → Comandos para geração de volumes (vistos nos capítulos anteriores)                                   |
| 2 | → Comandos auxiliares da linguagem (serão vistos a partir da seção 7.2 deste capítulo e no capítulo 8) |
| 3 | → Comandos para geração de <i>relações funcionais</i> (serão vistos no capítulo 9)                     |

### 7.1.2 O editor de vocabulário

O editor de vocabulário deverá, além de especificar a *seqüência das ações* a serem utilizadas, definir um *ícone* para o elemento do vocabulário a ser gerado pelas ações, i. é, uma *figura esquemática*, que permita ao projetista identificar visualmente o elemento.

A concepção do editor pode ser realizada de duas formas distintas:

- 1 → Utilizando-se um procedimento escrito:  
Constitui-se num meio notacional que registra computacionalmente, na forma de um texto, a seqüência das ações de projeto a serem executadas para gerar um elemento do vocabulário.
- 2 → Utilizando-se um procedimento gráfico-interativo: Constitui-se na utilização de um meio computacional interativo que permite ao projetista visualizar os vários estágios de projeto.

No caso da utilização de um *procedimento escrito*, o usuário deverá digitar a seqüência de ações na forma que foi apresentada nos capítulos anteriores, i. é, utilizando os formalismos notacionais para as operações de *acoplamento* e de *alteração*, como exemplificado a seguir:

```
VP(100,100,100,2,1)
VI('V1')
V('V2',1)
V('V3',5)
...
```

A seguir, um *interpretador* deverá ler a seqüência de comandos e executá-la de forma automática.

### 7.1.2.1 O editor de ícones

Note-se, que a geração do *ícone* deverá ser feita com auxílio de um Editor de ícones, i. é, um editor de

figuras bidimensionais, que após a sua geração, serão incorporadas ao elemento de projeto correspondente e que foi gerado pelo editor de vocabulário. Assim, toda vez que o projetista solicitar o cardápio de elementos do vocabulário /TUR 90/ /MAK 87/, os ícones correspondentes serão apresentados em uma janela especial, de forma a facilitar a escolha do elemento a ser gerado.

O editor de ícones será constituído por uma malha padrão de posicionamento de pixels (pontos gráficos da tela ou de outro dispositivo de saída gráfica), que o projetista do ícone deverá preencher para obter o desenho esquemático do elemento do vocabulário que foi projetado. A Figura 7.1 ilustra a malha padrão (uma matriz bidimensional de 25 x 15 pixels) e um exemplo de ícone para o elemento janela, cuja seqüência de geração foi apresentada no Exemplo 6.1 no capítulo anterior.

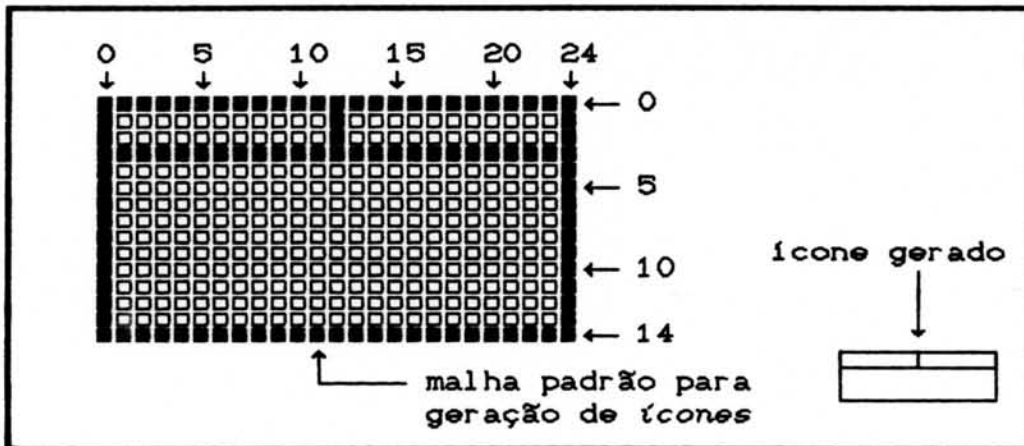


Figura 7.1 - Exemplo da definição de um ícone para representar o elemento janela definido no Exemplo 6.1.

O projetista do elemento do vocabulário, além de criar um ícone para representar o objeto, deverá definir uma tecla de função (F1, F2, F3, ..., shift-F1, shift-F2, shift-F3, ...), que ao ser ativada, provocará a execução

automática do elemento selecionado. Os ícones serão gerados utilizando-se um programa especial (EDICON.EXE), utilitário do sistema principal (SMOF.EXE). O utilitário de edição terá funções específicas para *acender* e *apagar* os pixels da malha padrão. Após a definição do desenho do ícone, o usuário deverá gravar o ícone com o mesmo nome do arquivo de ações que o ativará (surgirá uma pergunta para esta tarefa).

Exemplo: <i>JANELA.ICO</i> ← arquivo do ícone <i>JANELA.REG</i> ← arquivo de ações
---

### 7.1.2.2 Utilização de um procedimento gráfico-interativo para acompanhar o processo de edição

O simples registro da seqüência de ações não se constitui num meio eficiente para geração de elementos de um vocabulário pois, se o projetista não tiver um ambiente interativo adequado com o sistema, os resultados obtidos a partir da edição dos mecanismos para geração de volumes, acoplamentos e alterações somente poderão ser visualizados ao final do processo de concepção. Assim, surge a necessidade de acompanhar cada uma das etapas do processo, de forma a permitir correções ou modificações de acordo com as necessidades do projeto do elemento.

Assim, o projetista deverá dispor de um ambiente gráfico que permita visualizar cada uma das operações requisitadas, i. é, deverão existir *janelas de exibição*, tanto para a observação dos objetos tridimensionais, como para exibição de vistas, em projeções ortogonais, de forma a permitir operações de deslocamento de um ponto de observação dos objetos. O ambiente gráfico em questão,

apropriadamente denominado *Câmera Sintética*, foi desenvolvido durante as disciplinas de Computação Gráfica do CPGCC, e convenientemente aprimorado para incorporar a utilização dos mecanismos desenvolvidos neste documento, constituindo-se assim numa espécie de "laboratório de pesquisas", onde os conceitos e idéias apresentados podem ser testados.

Note-se, ainda, que ao gerar um novo volume ou ao aplicar uma operação de transformação alteradora, a expressão sintática deverá ser registrada automaticamente no arquivo de edição. Da mesma forma, se um volume for eliminado, a correspondente expressão sintática que gerou o volume deverá ser eliminada, levando-se em consideração as *conseqüências* desta operação sobre a geração dos demais elementos.

Convém salientar, que o projetista poderá utilizar as duas opções ao mesmo tempo, o que dará maior flexibilidade ao sistema, i. é, a concepção do elemento do vocabulário poderá ser realizada paralelamente utilizando-se tanto o *editor de vocabulário* como o *ambiente gráfico-interativo* do sistema.

## 7.2 Incorporando um mecanismo para entrada de parâmetros

Em algumas situações, o usuário de um vocabulário de elementos pode necessitar de um mecanismo que permita a entrada de um ou mais parâmetros das operações de *acoplamento* e de *alteração* de volumes, a serem utilizados dentro da seqüência de ações estabelecida no editor de vocabulário. Assim, por exemplo, o projetista pode querer gerar uma matriz bidimensional de volumes, de forma que o sistema solicite a entrada da *ordem* da matriz de volumes, ao ser pressionada a tecla de função que aciona a execução

das ações estabelecidas no arquivo de ações do elemento. Note-se, que este simples mecanismo pode proporcionar uma grande flexibilidade ao sistema, que poderá gerar resultados diferentes a partir de uma mesma seqüência de ações, visto que somente alguns parâmetros serão modificados. No caso da matriz bidimensional de volumes, o parâmetro a ser fornecido pelo projetista, será relativo ao mecanismo de repetição, i. é, o parâmetro NR das operações VR.

A entrada de um parâmetro será especificada pelo seguinte comando:

<p>?M, IP          ↑          mecanismo para entrada de um parâmetro a ser utilizado por uma seqüência de ações, na geração de um elemento do vocabulário</p>
<p>Onde:</p> <p>? = Símbolo utilizado para identificar a entrada de um parâmetro</p> <p>M = Mensagem a ser exibida na janela de mensagens do sistema</p> <p>IP = Identificação do parâmetro</p>

Assim, no caso da entrada de um parâmetro para identificar a ordem de uma matriz bidimensional de volumes o comando seria:

? 'Qual a ordem da matriz de volumes', NR

Assim, para o exemplo dado, toda vez que o *interpretador* encontrar um comando que possua o parâmetro NR, este assumirá o valor fornecido pelo projetista. A seguir, será apresentado um exemplo completo de uma seqüência de ações para geração de uma matriz bidimensional

de volumes, de ordem  $(N \times 3)$  (Figura 7.2):

```

?'Nome dos volumes:', NV
?'Ordem X da matriz de volumes:', NR
VI(NV)
VR(NV, 1, NR)
V(NV, 17)
VR(NV, 5, NR)
V(NV, 17)
VR(NV, 1, NR)
  
```

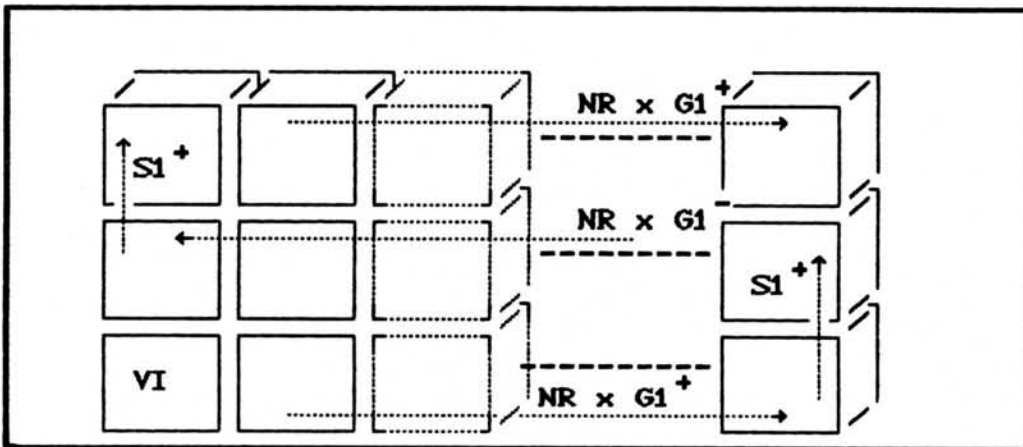
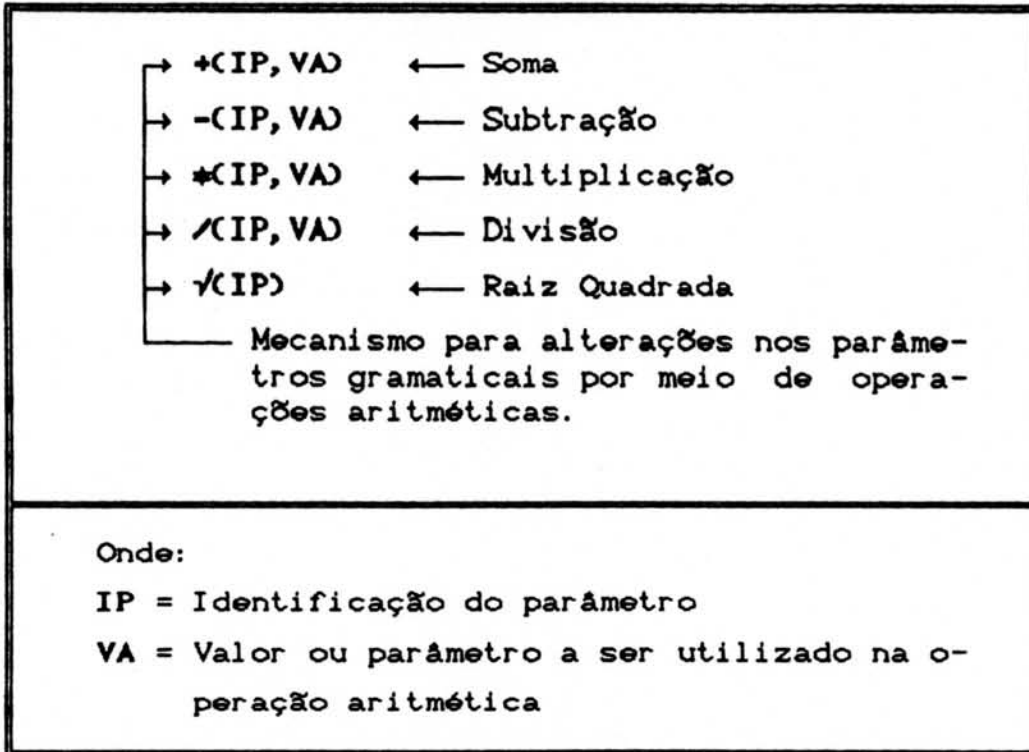


Figura 7.2 - Matriz bidimensional de volumes, gerada a partir de uma seqüência de ações, com utilização do mecanismo de entrada de parâmetros.

### 7.3 Um mecanismo para operações aritméticas

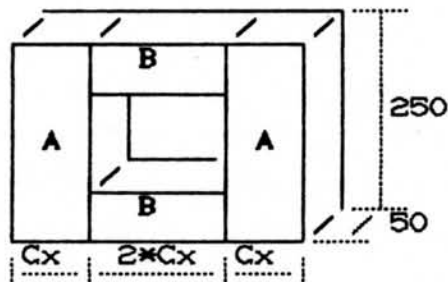
O mecanismo que será apresentado nesta seção, terá como função estabelecer uma forma para alteração do valor de um determinado parâmetro aplicando-se sobre este parâmetro uma das quatro operações aritméticas definidas a seguir.





Este mecanismo é útil quando, por exemplo, ao ser fornecido um parâmetro ao sistema, utilizando-se o mecanismo para entrada de um parâmetro (apresentado na seção anterior), se quer alterar o seu valor numérico para geração de um volume com características relativas ao valor anterior. O exemplo apresentado a seguir, ilustra o funcionamento do mecanismo em discussão.

Exemplo 7.1 → Gerar o módulo ilustrado abaixo de forma que a dimensão em X dos elementos identificados pela letra B seja o dobro da dimensão em X dos elementos identificados pela letra A.



Seqüência de ações:

```

?Largura das laterais do modulo (A) = ', Cx
VP(Cx, 250, 50, 2, 1)
VI('A')
*(Cx, 2)
VP(Cx, 50, 50, 2, 1)
V('B', 2)
TSC('A')
V('B', 1)
/(Cx, 2)
VP(Cx, 250, 50, 2, 1)
V('A', 1)

```

Note-se, que o identificador IP deverá ser um dos parâmetros numéricos estabelecidos nos mecanismos que foram definidos até aqui, i. é:

NR	←	Número de repetições
Cx, Cy, Cz	←	Dimensão em X, Y e Z
Dx, Dy, Dz	←	Deformação em X, Y e Z
NCx, NCy, NCz	←	Nova dimensão em X, Y e Z
D	←	Deslocamento
AR	←	Ângulo de rotação

## 7.4 Atribuição

O mecanismo de atribuição é de fundamental importância para o perfeito funcionamento da linguagem de ações, visto que permite a transmissão do valor de um parâmetro para outro, de forma a satisfazer uma exigência do projeto de uma seqüência de ações. Este mecanismo será apresentado a seguir.

$$:=(IP1, IP2)$$

↑ Mecanismo de atribuição

Onde:

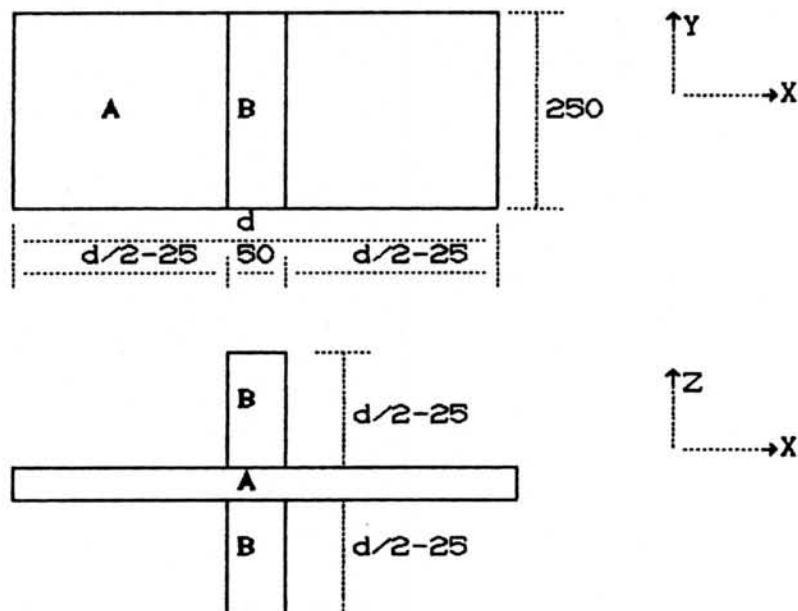
IP1 = Identificação do parâmetro que receberá o valor envolvido na atribuição

IP2 = Identificação do parâmetro fonte do valor de atribuição

Note-se, que será imprescindível considerar uma adequada compatibilidade entre os tipos dos parâmetros IP1 e IP2 para que a operação seja corretamente executada. Assim, se o tipo do parâmetro IP1 for *inteiro*, caso o tipo do parâmetro IP2 for *real*, este último deverá ser obrigatoriamente convertido para *real*.

O Exemplo 7.2, apresentado a seguir, ilustra o funcionamento do mecanismo de atribuição.

Exemplo 7.2 → Gerar o elemento modular esquematizado abaixo, utilizando o mecanismo de atribuição.



Seqüência de ações:

```

?d = ', Cx
VP(Cx, 250, 50, 2, 1)
VI('A')
/(Cx, 2)
-(Cx, 25)
:=(Cz, Cx)
VP(50, 250, Cz, 2, 1)
V('B', 9)
TSC('A')
V('B', 13)
:=(D, Cx)
Dx('B', D)

```

## 7.5 Exibição de mensagens

Ao ser executada uma seqüência de ações, pode ser interessante exibir ao usuário final do sistema uma determinada *mensagem*, de forma a possibilitar a transmissão de informações relevantes para a continuidade do processo de projeto. Além disso, essas informações devem ser exibidas em uma *janela de exibição* específica para essa finalidade, e podem consistir de:

- 1 → Resultados numéricos de operações executadas.  
Exemplo:  
'Area total = 120'
- 2 → Instruções para tomada de decisões a partir dos resultados obtidos após a execução de uma seqüência de ações.

Exemplo:

'A seguir, não será permitido GEMINAR!'

3 → Informações de operações executadas.

Exemplo:

'Erro ---> Dimensão inferior a 500'

Assim, pode-se definir o mecanismo para exibição de mensagens, como a seguir:

MENSAGEM(M, IP)

└─ Mecanismo para exibição de mensagens

Onde:

MENSAGEM = Identificador do mecanismo

M = Mensagem a ser exibida na janela de mensagens do sistema (entre apóstrofes)

IP = Identificação do parâmetro de exibição (opcional)

Como observação, vale salientar, que no caso da não existência do parâmetro IP, este deverá ser substituído por dois apóstrofes.

Exemplos:

→ Mensagem com parâmetro de exibição:

MENSAGEM('DIMENSÃO EM X = ', Cx)

→ Mensagem sem parâmetro de exibição:

MENSAGEM('OPERAÇÃO INVÁLIDA', '')

## 7.6 Variáveis auxiliares

Como pode ser observado nas definições dos mecanismos anteriores, o uso de parâmetros dentro de um comando de ação deve obedecer às seguintes regras sintáticas:

REGRA 1 → *Somente deverão ser utilizados os parâmetros específicos, definidos para o mecanismo em uso.*

Exemplos:

VP(Cx, Cy, Cz, T, C) ← *comando válido*

VP(Cx, D, Cy, Textura, C) ← *comando inválido*

erro 3: parâmetro inexistente  
 erro 2: fora de ordem  
 erro 1: parâmetro pertencente à outro mecanismo

REGRA 2 → *Valores numéricos ou strings serão aceitos, desde que, seja respeitada a devida compatibilidade em relação ao tipo do parâmetro esperado.*

Exemplos:

VP(250, Cy, 50.5, 5, C) ← *comando válido*

VP(250-100, Cy, 50.5, 100, 2.5) ← *comando inválido*

erro 3: tipo real  
 erro 2: muito grande  
 erro 1: operações aritméticas dentro do comando não são permitidas (devem ser realizadas antes e o resultado atribuído ao parâmetro Cx)

Como em toda linguagem de programação, a linguagem de ações deve oferecer ao usuário programador uma forma temporária para o armazenamento de valores inteiros, reais ou strings. Assim, com esse recurso computacional, o usuário programador pode realizar operações sobre o valor de um determinado parâmetro, sem a necessidade de sua perda ou alteração. Por exemplo, se o usuário final (que não precisa conhecer o programa) do sistema fornecer o valor da dimensão  $X$  do volume padrão, i. é,  $Cx$ , o mesmo valor pode ser atribuído a uma variável temporária que poderia ser multiplicada por 2 e dividida pelo valor de outro parâmetro, para ser posteriormente atribuído ao valor do parâmetro de deslocamento  $D$ . Note-se, que nesse caso não houve a perda do valor de  $Cx$ , visto que ele foi transportado para a variável temporária e, portanto, continua existindo e pode ser naturalmente utilizado. Observe-se, ainda, que uma variável temporária pode ser utilizada para armazenar o valor de qualquer parâmetro, desde que seja respeitada a compatibilidade de tipos.

Note-se que serão necessários 3 tipos de variáveis temporárias, ou auxiliares, cuja definição será apresentada a seguir.

Variáveis Auxiliares:

I : array[1..n] do tipo inteiro

R : array[1..n] do tipo real

S : array[1..n] do tipo string

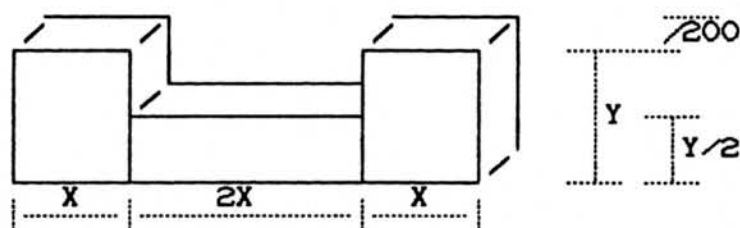
Onde:

n = número máximo de variáveis

Vale salientar, ainda, que a partir de agora, as variáveis auxiliares do tipo inteiro e real, poderão ser utilizadas nas operações aritméticas, de atribuição, entrada de parâmetros e exibição de mensagens.

O exemplo a seguir ilustra a utilização de variáveis auxiliares:

Exemplo 7.3 → Gerar os blocos esquematizados.



Sequência de ações:

```
?'Dimensao X dos blocos laterais = ',R[1]
?'Dimensao Y do bloco central = ',R[2]
:=(Cx, R[1])
:=(Cy, R[2])
*(Cx, 3)
*(Cy, 2)
VP(Cx, Cy, 200, 2, 3)
VI('B_LATERAL')
VC('B_LATERAL', 1)
:=(NCx, R[1])
ACx('B_LATERAL', NCx)
:=(Cx, R[1])
:=(Cy, R[2])
*(Cx, 2)
VP(Cx, Cy, 200, 2, 3)
VC('B_CENTRAL', 6)
```



## 7.7 Observações e comentários

Determinadas porções de uma seqüência de ações devem merecer uma atenção especial por parte do usuário programador e devem, portanto, receber algum tipo de especificação sob a forma de um seqüência textual de comentário. Assim, torna-se possível especificar o uso de um determinado mecanismo, informando a sua finalidade específica dentro da seqüência de ações. Note-se, que com a utilização desse recurso é possível obter uma seqüência de ações mais clara sob o ponto de vista do programador, visto que a função de cada segmento da seqüência fica adequadamente especificada.

O símbolo @ será utilizado para identificar uma *linha de comentário*, como especificado a seguir:

<p>● C            ↑            └─ linha de comentário</p>
<p>Onde:            ● = Identificador da <i>linha de comentário</i>            C = Texto do comentário</p>
<p>→ Exemplo:            ● -----            ● Geração do módulo A            ● -----</p>

Tabela 7.1 - Resumo dos mecanismos e símbolos do capítulo

Comandos auxiliares da linguagem de ações	
?M, IP	= entrada de parâmetros
+(IP, VA)	= operação de soma
-(IP, VA)	= operação de subtração
* (IP, VA)	= operação de multiplicação
/ (IP, VA)	= operação de divisão
√(IP, VA)	= raiz quadrada de um parâmetro
:= (IP1, IP2)	= comando de atribuição
MENSAGEM(M, IP)	= exibição de mensagens
@ C	= comentários
Identificadores de parâmetros e variáveis	
→ M	= mensagem (uma string)
→ IP	= identificação de um parâmetro
→ IP1	= identificação do parâmetro 1
→ IP2	= identificação do parâmetro 2
→ VA	= valor do parâmetro
→ I[n]	= variável auxiliar do tipo <i>inteiro</i>
→ R[n]	= variável auxiliar do tipo <i>real</i>
→ S[n]	= variável auxiliar do tipo <i>string</i>
→ n	= identificação da variável ( <i>inteiro</i> )
→ C	= texto do comentário
→ 'ss'	= uma <i>string</i>
→ ss	= valor da <i>string</i>

## 8. MECANISMO DE CONTROLE

A *linguagem de ações*, cujas definições de seus mecanismos aparecem nos capítulos anteriores, permite a geração de objetos em três dimensões de acordo com uma seqüência pré-estabelecida de ações. Assim, a criação de volumes dá-se de forma natural, sem *restrições* ou *condições* especiais de projeto, i. é, não ocorrem desvios condicionais, pois ainda não foi descrito um mecanismo para controlar a seqüência de ações.

A definição de *regras de geração* /TUR 90/ e transformação de objetos tridimensionais pode ser realizada de forma a estabelecer diferentes seqüências de ações, cuja utilização pode ou não ser ativada, mediante a utilização de *obstáculos controladores*, ou *restrições* /TUR 90/, como sugerido nas definições de *shape grammars* /FLE 86/. Esses mecanismos de controle devem ser aplicados de forma a permitir ou evitar o acesso a um ou mais segmentos para atingir um determinado objetivo de projeto. Assim, torna-se possível controlar estruturas de construção embutidas na *linguagem de ações*, permitindo a invocação automática de regras. Note-se que nesse caso, o processo de geração pode ser visto como uma pesquisa em uma estrutura do tipo *árvore*, cujos nodos constituem soluções parciais de projeto, que podem ou não serem acessadas /GER 85/.

A aplicação de *restrições* /TUR 90/ à geração de volumes constitui outra abordagem para o mecanismo controlador. Nesse caso, por exemplo, somente determinados valores de parâmetros serão aceitos, de forma a atingir objetivos esperados na seqüência de projeto. Assim, *rotações* de determinados objetos poderiam ser previamente definidas em ângulos de 0°, 90°, 180° e 270°. O controle

seria então aplicado de forma a evitar que o usuário tentasse rotacionar um objeto utilizando um ângulo diferente daqueles que foram anteriormente especificados.

A seguir, serão estabelecidas algumas regras importantes para formalizar a definição do mecanismo de controle:

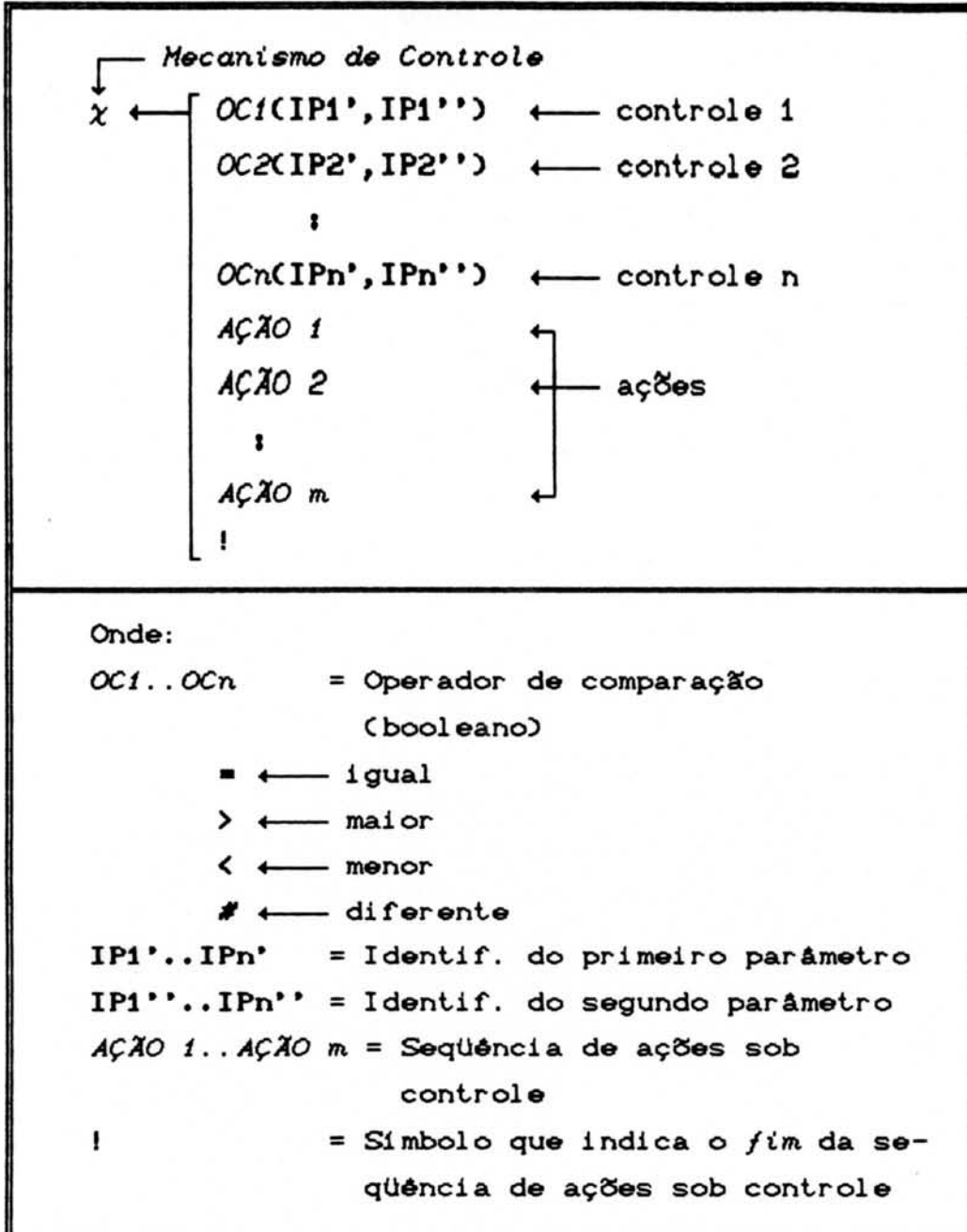
Regra 8.1 - O mecanismo de controle deve exercer uma função de comparação entre dois parâmetros, sendo que, se o resultado da comparação for verdade, a sequência de ações que segue a instrução de controle deve ser executada.

Regra 8.2 - Deve haver um símbolo especial para indicar o fim de uma sequência de ações sob controle.

Regra 8.3 - Deve ser permitida a utilização de mais de uma comparação, sendo que, nesse caso, se o resultado de uma das comparações for falso, a sequência de ações que segue esta última instrução não deve ser executada.

Regra 8.4 - Deverá existir um operador específico para indicar cada uma das operações booleanas essenciais (igual, maior, menor e diferente).

A partir dos conceitos apresentados acima, será definido formalmente o mecanismo de controle  $\chi$ .

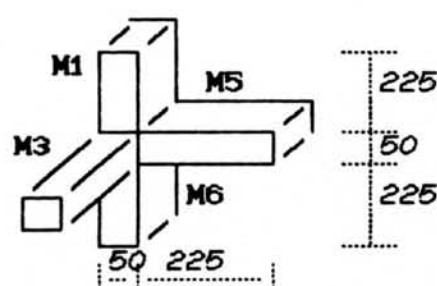


Note-se que o *interpretador* da linguagem de ações, ao atingir o primeiro operador de comparação deverá verificar se o resultado da comparação é *verdadeiro* ou *falso*. Se for *verdadeiro*, então a próxima linha deverá ser executada. Se for *falso*, todas as ações que estiverem entre o controle que *falhou* e a *marca de finalização* da sequência de ações sob controle, deverão ser ignoradas pelo interpretador.

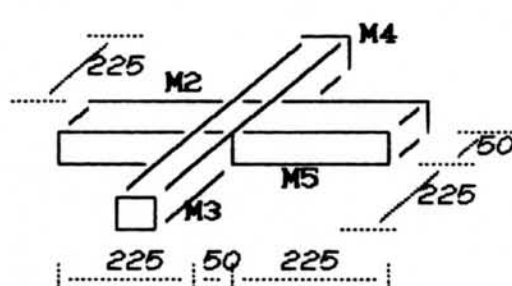
O Exemplo 8.1, apresentado a seguir, e convenientemente discutido, demonstra uma aplicação prática para o mecanismo  $\chi$ .

Exemplo 8.1  $\longrightarrow$  Desenvolver uma seqüência de ações para gerar os módulos de conexão A e B esquematizados abaixo, de forma que as conexões entre os módulos sejam realizadas adequadamente de acordo com os três tipos de acoplamento permissíveis (geminção, justaposição e superposição).

Módulo A



Módulo B



```

@ ==> Geracao do Modulo A
@ ==> Define CENTRO
VP(50,50,50,2,5)
@ ==> Define tipo de acoplamento
>(I[0],0)
?'[1]G+ [2]G- [3]J+ [4]J- [5]S+ [6]S- ==>',I[1]
!
>(I[0],0)
=(I[1],1)
:=(RG,1)
!
>(I[0],0)
=(I[1],2)
:=(RG,5)
!
>(I[0],0)
=(I[1],3)
:=(RG,9)
!
>(I[0],0)
=(I[1],4)

```

```

:= (RG,13)
!
>(I[0],0)
=(I[1],5)
:= (RG,17)
!
>(I[0],0)
=(I[1],6)
:= (RG,21)
!
@ ==> Gera Centro
>(I[0],0)
V<'CENTRO',RG)
!
=(I[0],0)
VI<'CENTRO')
!
@ ==> Deslocamento em X+
@ -----
>(I[0],0)
=(I[1],1)
=(I[9],1)
Dx<'CENTRO',225)
!
>(I[0],0)
=(I[1],1)
=(I[9],2)
Dx<'CENTRO',225)
!
@ ==> Deslocamento em X-
@ -----
>(I[0],0)
=(I[1],2)
=(I[9],1)
Dx<'CENTRO',-225)
!
>(I[0],0)
=(I[1],2)
=(I[9],2)
Dx<'CENTRO',-450)
!
@ ==> Deslocamento em Z+
@ -----
>(I[0],0)
=(I[1],3)
=(I[9],1)
Dz<'CENTRO',225)
!
>(I[0],0)
=(I[1],3)
=(I[9],2)
Dz<'CENTRO',450)

```

```

!
@ ==> Deslocamento em Z-
@ -----
>(I[0],0)
=(I[1],4)
=(I[9],1)
Dz('CENTRO',-225)
!
>(I[0],0)
=(I[1],4)
=(I[9],2)
Dz('CENTRO',-225)
!
@ ==> Deslocamento em Y+
@ -----
>(I[0],0)
=(I[1],5)
=(I[9],1)
Dy('CENTRO',450)
!
>(I[0],0)
=(I[1],5)
=(I[9],2)
Dy('CENTRO',225)
!
@ ==> Deslocamento em Y-
@ -----
>(I[0],0)
=(I[1],6)
=(I[9],1)
Dy('CENTRO',-450)
!
>(I[0],0)
=(I[1],6)
=(I[9],2)
Dy('CENTRO',-225)
!
@ ==> Gera M5
VP(225,50,50,2,5)
TSC('CENTRO')
VC('M',1)
@ ==> Gera M3
VP(50,50,225,2,5)
TSC('CENTRO')
VC('M',13)
@ ==> Gera M1
VP(50,225,50,2,5)
TSC('CENTRO')
VC('M',17)
@ ==> Gera M6
TSC('CENTRO')
VC('M',21)

```



```

@ ==> Semente no CENTRO
ANC 'CENTRO', 'C')
TSC 'C')
ANC 'M', 'M1')
:= (I[9], 1)
+ (I[0], 1)

```

```

@ ==> Geracao do Modulo B
@ ==> Define CENTRO
VP(50,50,50,2,5)
@ ==> Define tipo de acoplamento
>(I[0],0)
?'[1]G+ [2]G- [3]J+ [4]J- [5]S+ [6]S- ==>', I[1]
!
>(I[0],0)
=(I[1],1)
:=(RG,1)
!
>(I[0],0)
=(I[1],2)
:=(RG,5)
!
>(I[0],0)
=(I[1],3)
:=(RG,9)
!
>(I[0],0)
=(I[1],4)
:=(RG,13)
!
>(I[0],0)
=(I[1],5)
:=(RG,17)
!
>(I[0],0)
=(I[1],6)
:=(RG,21)
!
@ ==> Gera Centro
>(I[0],0)
VC 'CENTRO', RG)
!
=(I[0],0)
VIC 'CENTRO')
!
@ ==> Deslocamento em X+
@ -----
>(I[0],0)
=(I[1],1)
=(I[9],1)

```

```

DxC 'CENTRO', 450)
!
>(I[0],0)
=(I[1],1)
=(I[9],2)
DxC 'CENTRO', 450)
!
@ ==> Deslocamento em X-
@ -----
>(I[0],0)
=(I[1],2)
=(I[9],1)
DxC 'CENTRO', -225)
!
>(I[0],0)
=(I[1],2)
=(I[9],2)
DxC 'CENTRO', -450)
!
@ ==> Deslocamento em Z+
@ -----
>(I[0],0)
=(I[1],3)
=(I[9],1)
DzC 'CENTRO', 225)
!
>(I[0],0)
=(I[1],3)
=(I[9],2)
DzC 'CENTRO', 450)
!
@ ==> Deslocamento em Z-
@ -----
>(I[0],0)
=(I[1],4)
=(I[9],1)
DzC 'CENTRO', -450)
!
>(I[0],0)
=(I[1],4)
=(I[9],2)
DzC 'CENTRO', -450)
!
@ ==> Deslocamento em Y+
@ -----
>(I[0],0)
=(I[1],5)
=(I[9],1)
DyC 'CENTRO', 225)
!
@ ==> Deslocamento em Y-
@ -----

```

```

>(I[0],0)
=(I[1],6)
=(I[9],1)
Dy('CENTRO',-225)
!
@ ==> Gera M5
VP(225,50,50,2,5)
TSC 'CENTRO')
V('M',1)
@ ==> Gera M3
VP(50,50,225,2,5)
TSC 'CENTRO')
V('M',13)
@ ==> Gera M2
VP(225,50,50,2,5)
TSC 'CENTRO')
V('M',5)
@ ==> Gera M4
VP(50,50,225,2,5)
TSC 'CENTRO')
V('M',9)
@ ==> Semente no CENTRO
ANC 'CENTRO', 'C')
TSC 'C')
ANC 'M', 'M2')
:=(I[9],2)
+(I[0],1)

```

Observe-se, que a variável auxiliar I[0] foi utilizada como um parâmetro especial para indicar se o próximo objeto a ser criado (e sua seqüência de ações correspondente) é o *objeto inicial*. Assim, se I[0] > 0, então o próximo objeto a ser criado não é o *objeto inicial* e determinadas seqüências de ações devem ser executadas. Outra variável auxiliar, I[9], também foi utilizada para indicar o tipo do último objeto que foi criado. Assim se I[9] = 1, então, um objeto do tipo Módulo A foi gerado, e se I[9] = 2, um objeto do tipo Módulo B foi o *objeto anterior*. Portanto, o valor atribuído a I[9] será utilizado como um *fator de decisão* que terá como função decidir qual seqüência de ações deverá ser executada. Note-se, também, a importância do volume CENTRO, na geração dos módulos do exemplo, visto que ele é utilizado como um *volume de*

referência para o perfeito funcionamento das operações de acoplamento. Assim, a primeira ação de geração volumétrica a ser executada é a criação e o devido *deslocamento*, de acordo com o tipo de acoplamento a ser executado (estabelecido pelo valor da variável auxiliar I[1]) do CENTRO do módulo e, a seguir, utilizando-se este volume como *semente*, os demais elementos do módulo são criados. Um exemplo ilustrativo pode ser visto no ANEXO 9 deste trabalho.

Tabela 8.1 - Resumo dos mecanismos e símbolos do capítulo

Comando auxiliar da linguagem de ações	
OC1(IP1',IP1'')	← controle 1
OC2(IP2',IP2'')	← controle 2
:	
OCn(IPn',IPn'')	← controle n
AÇÃO 1	← ações
AÇÃO 2	
:	
AÇÃO m	
!	
} = mecanismo de controle	
Identificadores de parâmetros e operadores	
→ OC1...OCn	= operador de comparação (booleano)
■	← igual
>	← maior
<	← menor
#	← diferente
→ IP1'...IPn'	= identif. do parâmetro 1
→ IP1''...IPn''	= identif. do parâmetro 2
→ AÇÃO 1..AÇÃO m	= seqüência de ações sob controle
Símbolos	
→ !	= marca fim da seqüência de ações sob controle
→ χ	= mecanismo de controle

## 9. RELAÇÕES FUNCIONAIS

A elaboração de um projeto pode ser vista como uma tarefa extremamente complexa, onde diversos aspectos devem ser considerados, dependendo da aplicação. Assim, por exemplo, não é suficiente considerar somente os aspectos gráficos ou geométricos dos elementos projetados. Outros fatores devem ser cuidadosamente considerados e analisados, incorporando assim, processos e estratégias, principalmente para permitir a tomada de certas decisões fundamentais para o projeto. Da mesma forma, devem existir mecanismos eficientes para permitir o armazenamento das características e dos requisitos do produto a ser desenvolvido. Assim, um sistema computacional de projeto, deve ser capaz de assimilar as exigências necessárias para se produzir o efeito desejado, relativas ao produto que se quer obter. Então, de posse dessas informações, reguladas pela incorporação de testes de compatibilidade, pode-se chegar ao resultado final com maior ou menor nível de sintonia com os efeitos desejados.

### 9.1 Como definir a estrutura funcional de um projeto

Para que se possa construir corretamente a estrutura funcional de um projeto, deve-se, inicialmente, ativar uma seqüência de procedimentos metódicos para o perfeito entendimento do processo projetivo.

Então, como pode ser visto em /GRA 89/, um processo de projeto pode ser definido de acordo com as seguintes fases:

- 1 → Elaborar os requisitos do projeto: Consiste em estabelecer a totalidade das funções do projeto, assim como, as restrições e condições iniciais que devem ser implementadas.
- 2 → Definir a estrutura funcional: Consiste na decomposição das funções estabelecidas na fase anterior, em uma rede de sub-funções e funções primitivas.
- 3 → Pesquisar soluções: Implica na identificação apropriada de princípios (leis da física) para o cumprimento das funções.
- 4 → Inserir os detalhes do projeto: Usa os diversos níveis funcionais para obter resultados mais ou menos detalhados.

Assim, um processo de engenharia de projeto pode ser utilizado para identificar as características funcionais para a especificação de um produto e sua manufatura. Então, torna-se possível a identificação de um conjunto de tipos básicos de conhecimento que podem ser gerados por atividades normalmente utilizadas em todos os tipos de engenharia de projeto /SMI 89/:

- 1 → Descrição dos requisitos: Os requisitos são descritos ou modelados de modo formal ou informal e modificados de acordo com a elaboração das possibilidades de proje-

to, identificação de falhas e descoberta de inconsistências.

- 2 → Decomposição: Os requisitos e suas respectivas ações são subdivididos em problemas de mais fácil gerenciamento.
- 3 → Planejamento estratégico: As regras de objetivos intermediários e subproblemas são identificados, levando-se em consideração as características e dificuldades de projeto.
- 4 → Detalhamento e parametrização: As partes de um projeto são trabalhadas em seus detalhes para estabelecer valores, parâmetros de projeto, custos etc.
- 5 → Síntese: Esquemas e soluções de projeto são gerados a partir de soluções parciais e subproblemas.
- 6 → Simulação: Consiste na verificação da eficiência dos esquemas e mecanismos propostos.
- 7 → Análise: Detalhes de projeto propostos, valores de parâmetros e outros aspectos são testados contra os requisitos e outros critérios.
- 8 → Otimização: As interações e relacionamentos entre as subpartes de um projeto são otimizadas em relação a critérios globais.
- 9 → Documentação: Esclarecimentos sobre as decisões de projeto e decisões adotadas são preparadas e registradas.

Assim, ao final do processo de projeto, é possível obter uma especificação completa do produto,

incluindo custo, manufatura, teste, documentação, instalação, manutenção etc. Note-se, ainda, que o processo deve ocorrer de forma a permitir uma contínua exploração, avaliando-se as várias soluções possíveis, até que, finalmente, uma solução otimizada seja encontrada e, posteriormente, validada /SMI 89/.

## 9.2 Criando classes de objetos de projeto

Inicialmente, considerem-se as seguintes regras:

Regra 9.1 - Os elementos de um projeto serão denominados objetos.

Regra 9.2 - Os objetos de um projeto serão agrupados em classes, i. é, ocorrências de objetos que possuem os mesmos atributos dentro do universo de projeto.

Regra 9.3 - Um objeto será constituído geometricamente por um ou mais volumes (acoplados de acordo com as regras e mecanismos definidos nos capítulos anteriores).

Para satisfazer à Regra 9.2, será estabelecida uma forma para gerar e armazenar as propriedades, ou atributos, de uma classe de objetos de projeto. Essa forma de representação será utilizada pelo sistema principal de modelamento de objetos funcionais, que deverá possuir mecanismos para a criação de estruturas de dados. Aqueles mecanismos que foram definidos em capítulos anteriores deste trabalho serão incorporados posteriormente.



### 9.2.1 Utilizando uma estrutura de dados relacional para representar os objetos de um projeto

Um sistema relacional é um sistema em que os dados são vistos pelo usuário como *tabelas* e os operadores disponíveis para os usuários são operadores que permitem a geração de novas tabelas, a partir de outras já existentes /DAT 86/. Além disso, ao ser definida uma estrutura de dados relacional, algumas terminologias formais devem ser definidas:

- 1 → Domínio: É um conjunto de valores do qual são extraídos os valores reais de um determinado atributo.
- 2 → Relações: Uma relação consiste de um conjunto fixo de atributos, sendo que cada atributo está associado a um determinado domínio (cabecalho da relação), e de um conjunto de tuplas, i. é, conjuntos de pares valor-atributo. Assim, para cada atributo existe um valor pertencente a um determinado domínio.
- 3 → Tuplas: Correspondem às linhas das relações.
- 4 → Atributos: Correspondem às colunas das relações.
- 5 → Chave Primária: É um identificador especial, i. é, um atributo (ou uma combinação de atributos), que é utilizado para individualizar as tuplas de uma relação.

A Figura 9.1 mostra um exemplo de uma relação de materiais em projeto de instalação sanitária, onde pode ser observada a existência de três atributos (Nome, Unidade e

Quantidade), além de uma chave primária (Material) que serve de parâmetro de individualização para acesso às informações contidas na relação. Observe-se também, os domínios de cada atributo que, neste exemplo, podem ser de dois tipos: string (sequência de caracteres alfa-numéricos) e inteiro (valor numérico sem ponto decimal).

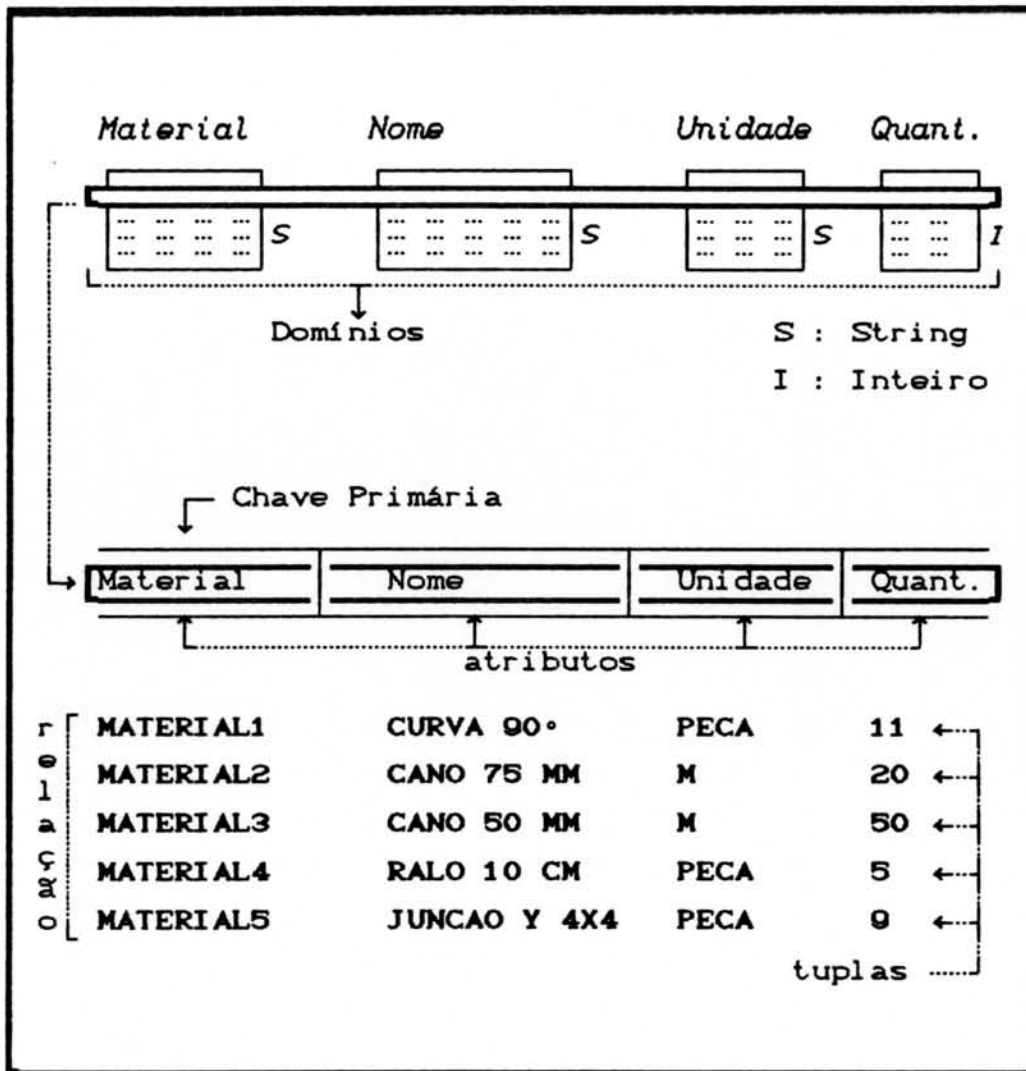


Figura 9.1 - Uma relação de materiais de uma instalação sanitária.

Para complementar esta seção, vamos citar as principais propriedades das relações /DAT 86/, que se constituem em algumas das razões pelas quais este tipo de estrutura de dados foi escolhido para representar objetos

neste trabalho.

- 1 → Não há tuplas duplicadas: Esta propriedade se fundamenta no fato de que o corpo de uma relação pode ser visto matematicamente como um conjunto de tuplas e, por definição, conjuntos não possuem elementos duplicados.
- 2 → Tuplas não são ordenadas: Aqui também, considera-se que o corpo de uma relação é um conjunto matemático, e conjuntos não são ordenados. Portanto, a ordem em que as tuplas aparecem na relação não é importante.
- 3 → Atributos não são ordenados: Mais uma vez, como relações são conjuntos de atributos, a ordem em que os atributos aparecem não é relevante. Assim, os atributos devem ser referenciados pelo nome e não pela posição que ocupam dentro da relação.
- 4 → Os valores dos atributos são atômicos: Para cada posição linha/coluna de uma relação existe somente um valor, nunca um conjunto de valores, i. é, relações não apresentam grupos repetidos, sendo portanto normalizadas.

Note-se, ainda, que todos os entes (objetos) que existem na natureza são únicos, não existindo outro elemento exatamente igual (até mesmo pelo fato de que "dois corpos não podem ocupar o mesmo lugar no tempo e no espaço"). Portanto, visto que as relações não apresentam elementos duplicados, este tipo de representação é bastante adequado para representar os objetos de um projeto.

Vale mencionar, ainda, que é comum a utilização de estruturas de dados relacionais em sistemas gráficos interativos /PAT 82/ /SPO 84/. Assim, os conceitos que serão estabelecidos a seguir serão fundamentados em estruturas de dados relacionais, e deverão obedecer às definições estabelecidas nesta seção.

### 9.2.2 Geração de classes de objetos utilizando relações

A partir de agora, será utilizado um exemplo que servirá para ilustrar os conceitos e definições que serão desenvolvidos a seguir. O referido exemplo, com aplicação em cálculo estrutural de concreto armado (área de engenharia civil) foi escolhido devido ao fato de que apresenta uma série de propriedades e situações que se ajustam aos propósitos deste trabalho.

Inicialmente, considere-se a elaboração de um projeto de uma estrutura de concreto armado para construção de um edifício, onde aparecem os seguintes elementos básicos (objetos) /ROC 84/:

- 1 → Lajes: São placas de concreto armado, cuja função é resistir às cargas verticais transmitidas a um plano horizontal ( piso dos edifícios).
- 2 → Vigas: São peças de concreto armado, de altura superior a das lajes, que são dispostas em duas direções perpendiculares, servindo de apoio às lajes.

3 → Pilares: São estruturas verticais que servem para transmitir as cargas de cada pavimento (cuja estrutura é formada por lajes repousando em um conjunto de vigas) às fundações. Note-se, ainda, que os pilares recebem as cargas dos pavimentos pelas vigas que se apoiam nestes elementos verticais.

4 → Fundações: São estruturas destinadas a transmitir ao solo as cargas provenientes dos diversos pavimentos, servindo, portanto, de apoio aos pilares.

A seguir, a Figura 9.2 mostra um desenho esquemático dos elementos que foram definidos.

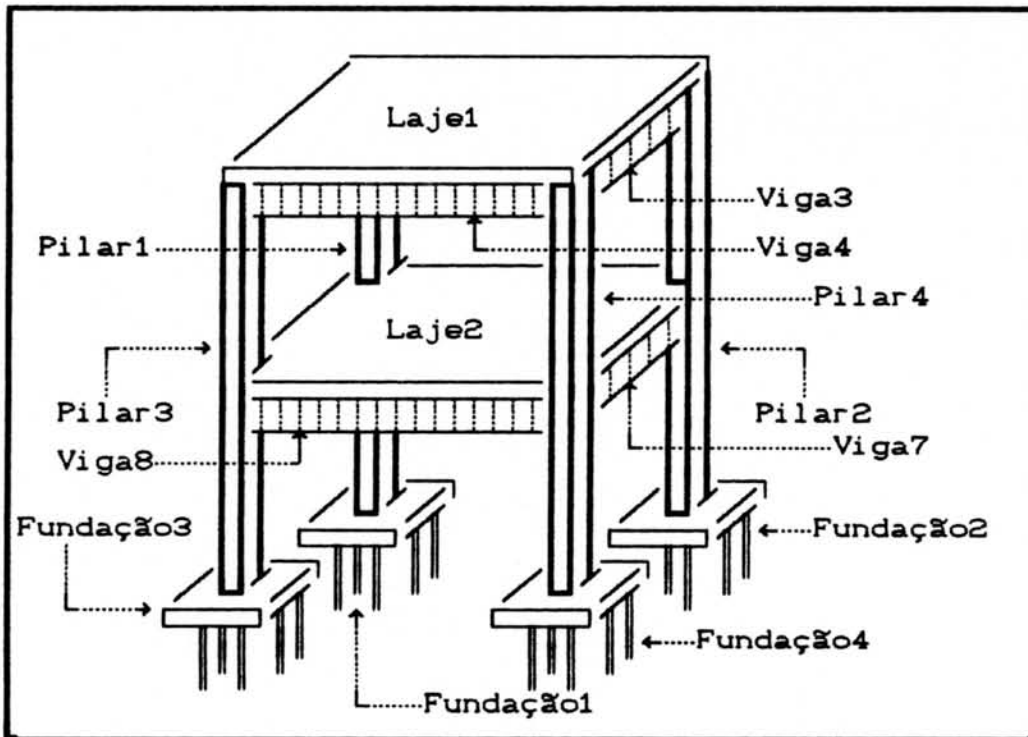


Figura 9.2 - Elementos de uma estrutura de edifício.

Os elementos estruturais definidos acima, podem ser vistos como *classes* de objetos, i. é, *relações* que descrevem a estrutura e comportamento de suas *ocorrências*. Assim, o comportamento de uma classe (*Viga*, por exemplo) é determinado pelos *métodos funcionais* (*seqüências de ações para geração de volumes e relações funcionais*) desta classe. Além disso, cada classe de objetos, apresenta uma série individual de características (*atributos*), que devem ser considerados e devidamente armazenados em *relações*. Posteriormente, os procedimentos a serem executados deverão ser incluídos e associados às ocorrências de cada classe, por meio de seqüências e sub-seqüências de ações, para a criação, manipulação, eliminação e ativação de outros procedimentos a serem considerados devido às conseqüências produzidas pela inserção ou eliminação de objetos no todo de projeto. Esta última consideração, será detalhadamente definida em outra seção neste capítulo, e vai tratar dos *relacionamentos* que ocorrem e que podem ser produzidos entre os objetos envolvidos em um processo de projeto.

Como exemplo, serão criadas as seguintes *classes* de elementos estruturais para uma estrutura de edifício:

<sup>1</sup> → Classe LAJE: Arquivo LAJE. ATR.	
Domínio	Atributo
I	XMIN
I	ZMIN
I	XMAX
I	ZMAX
I	ESPESSURA
I	CARGA_TOTAL
S	UTILIZACAO

2 → Classe VIGA: Arquivo VIGA.ATR.

Domínio	Atributos
I	COORD_INIC_X
I	COORD_INIC_Z
I	COORD_FIM_X
I	COORD_FIM_Z
I	BASE
I	ALTURA
I	CARGA_TOTAL

3 → Classe PILAR: Arquivo PILAR.ATR.

Domínio	Atributo
I	COORD_CG_X
I	COORD_CG_Z
I	DIMENSAO_X
I	DIMENSAO_Z
I	ALTURA
I	CARGA_TOTAL

4 → Classe FUNDAÇÃO: Arquivo FUNDACAO.ATR.

Domínio	Atributo
I	COORD_X
I	COORD_Z
I	CARGA_TOTAL
S	TIPO_DE_BLOCO
I	DIAM_DAS_ESTACAS

### 9.2.3 O mecanismo para criação de uma classe

A partir desta seção, serão definidos formalmente os mecanismos suplementares da *linguagem de ações*, para criação, gerenciamento e manipulação de *relações funcionais*. Uma observação importante a ser considerada, reside no fato de que estes mecanismos terão o seu funcionamento aplicado diretamente em meios de armazenamento externos ao sistema principal, i. é, por exemplo, em discos rígidos (*winchester de computadores pessoais*).

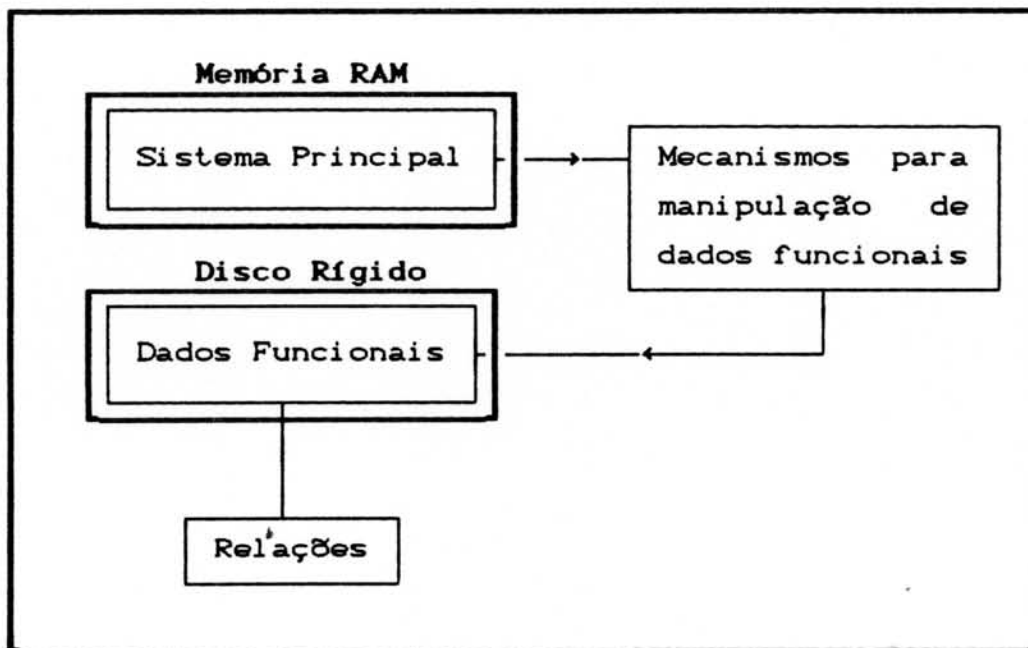


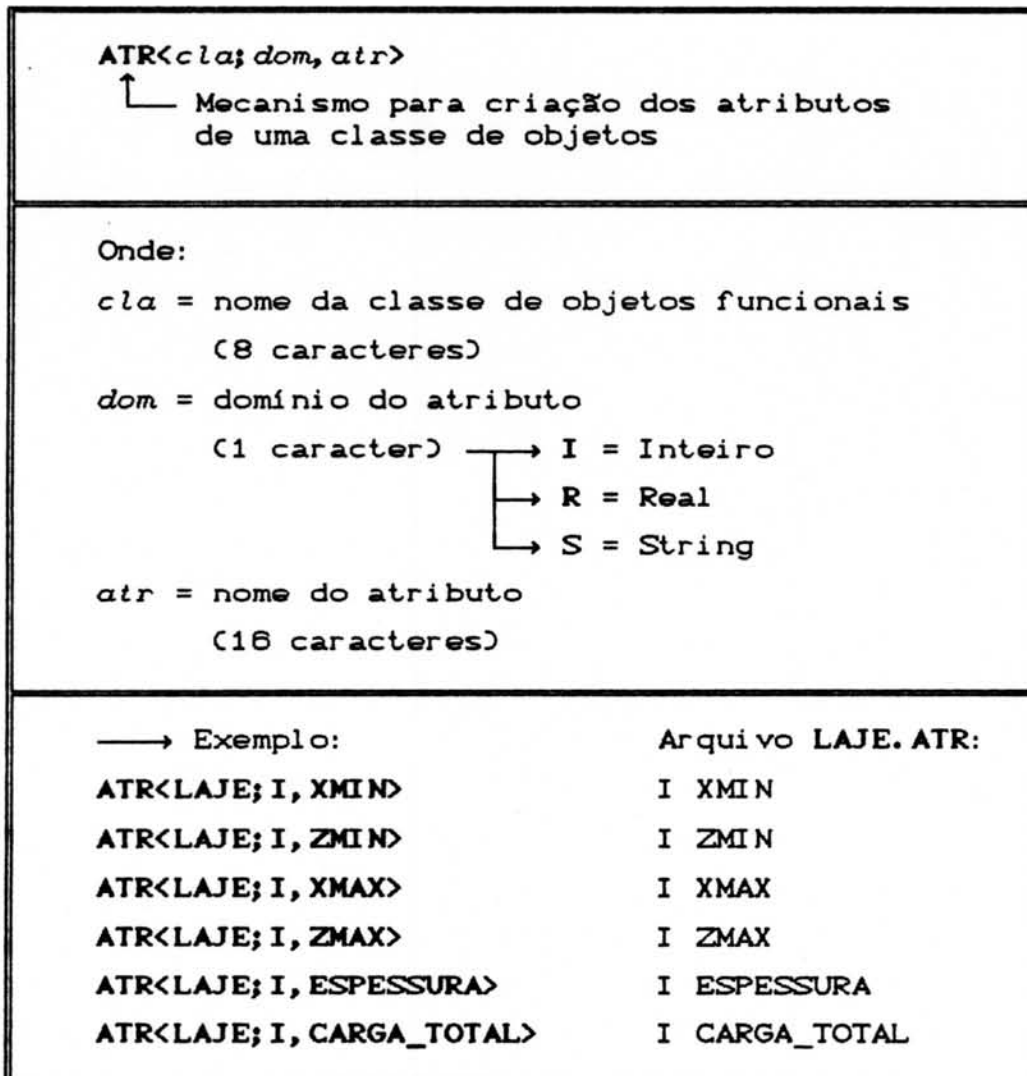
Figura 9.3 - Independência entre o sistema principal e os dados funcionais.

A principal vantagem na utilização desse tipo de armazenamento de informações é permitir o armazenamento de grandes quantidades de dados, independentemente do limite de memória RAM dos computadores. Assim, visto que os dados residem em um meio externo ao sistema, este último pode crescer independentemente da quantidade de informações armazenadas. Também, o acesso é principalmente a



*portabilidade* dessas informações torna-se mais eficiente, permitindo que outros sistemas possam fazer uso dos resultados obtidos, permitindo, por exemplo, a geração de relatórios, orçamentos e outros documentos necessários para a execução de um produto. Como desvantagem, deve-se salientar que o acesso às informações será mais lento, visto que uma consulta em um meio de armazenamento externo não é realizada diretamente na memória do computador. A Figura 9.3 mostra um esquema onde podem ser observados o sistema principal e a base de dados.

A seguir, será definido o mecanismo para criação de classes de objetos funcionais.



Como observação, deve-se mencionar que, a partir do momento em que o comando ATR é executado, a referida classe é criada com seu *primeiro* atributo. Assim, outros atributos podem ser gerados para a mesma classe, incorporando-se aos demais anteriormente criados, bastando para isso, executar outro comando ATR para a mesma classe.

### 9.3 Representação das ocorrências de uma classe

Após serem criados os atributos de uma classe de objetos e seus respectivos domínios, torna-se possível a representação das ocorrências ou entidades dessa classe, i. é, os objetos com suas características próprias que os individualizam no mundo real. Dessa forma, as entidades representarão as *lajes, vigas, pilares e fundações* do exemplo apresentado na seção anterior, além de outras relações que serão necessárias para a aplicação das ações funcionais de projeto, que serão avaliadas no momento oportuno neste trabalho.

O comando da linguagem de ações ENT, cuja definição formal é apresentada a seguir, será utilizado para gerar uma ocorrência de uma classe.

ENT<cla;nun,at1,at2,at3,...,atn>

↑ Mecanismo para criação de uma entidade

Onde:

cla = nome da classe a que pertence a entidade  
(8 caracteres)

nun = número da entidade (funciona como uma  
chave primária) (inteiro)

*at1, at2, at3, ..., atn* = valores das ocorrências dos atributos da classe (devem ser respeitados os domínios estabelecidos na relação de definição da classe correspondente)

—→ Exemplo:

:= (I(2), 400)

:= (I(3), 500)

:= (I(5), 1400)

ENT<LAJE; 1, 0, 0, I(2), I(3), 8, I(5)>

+ (I(2), 500)

ENT<LAJE; 2, 400, 0, I(2), I(3), 7, 1540>

Arquivo LAJE.ENT:

LAJE1	0	0	400	500	8	1400
LAJE2	400	0	900	500	7	1540

Note-se que as ocorrências das entidades serão individualizadas pelo identificador especial localizado na primeira coluna da relação de entidade, e que será formado pela concatenação entre o nome da classe *cla* e o número da entidade *nun* (LAJE1, LAJE2 no exemplo acima). Assim, esta *chave primária* vai servir como parâmetro de referência para acesso às informações relativas aos valores dos atributos de uma dada ocorrência.

Note-se, ainda, que a ordem em que foram estabelecidos os atributos de uma classe (gerada com o comando ATR), deve ser rigorosamente respeitada, visto que ao ser executado o comando ENT, os atributos existentes no arquivo da relação de atributos da classe, serão lidos na ordem em que foram criados, e simultaneamente a essa

operação de leitura, os respectivos valores das ocorrências serão armazenados na relação de entidade.

Como observação complementar, deve-se considerar o fato de que os valores inicialmente atribuídos às ocorrências de uma classe de objetos podem não ser os parâmetros definitivos destas ocorrências, podendo sofrer alterações durante os estágios de desenvolvimento de um processo de projeto, para satisfazer necessidades específicas do projeto ou como resultado de um sub-procedimento de cálculo matemático para dimensionamento, por exemplo.

#### 9.4 Incorporando os relacionamentos funcionais

Como foi visto, os *objetos* existentes e que podem ser identificados no mundo real, podem ser representados em uma relação do tipo *entidade*. Além disso, as entidades que possuem propriedades comuns, i. é, *mesmos atributos*, podem ser agrupadas em uma mesma relação, constituindo dessa forma, as *classes* de objetos.

A partir desta seção, será considerado o fato de que algumas ocorrências existentes nas relações de entidade dependem ou produzem efeitos sobre outras ocorrências de entidades. Este vínculo entre entidades pode ser criado ou eliminado durante as etapas de um projeto, podendo, dessa forma, existir em um dado momento e decorrido algum tempo, após a aplicação de algumas seqüências de ações, deixar de existir, ou até mesmo multiplicar-se, gerando novas vinculações existenciais com outros objetos anteriormente criados.

Há dois tipos principais de relacionamentos /BJO 89/ que podem ocorrer entre os objetos de um projeto, que são:

- 1 → Um objeto pode pertencer a um objeto maior, sendo portanto, parte de outro (Figura 9.4(a), onde os ferros FN1, FN2, FP1 e FP2 fazem parte da viga V1).
- 2 → Um objeto pode estar conectado a outro objeto, estando portanto, acoplado a este outro objeto (Figura 9.4(b), onde os módulos MOD1, MOD2, MOD3 e MOD4, estão conectados uns aos outros).

Observe-se, que nas Figuras 9.4(a) e 9.4(b), os relacionamentos "parte de" e "por acoplamento" são apresentados sob a forma de dois tipos particulares de diagramas. No segundo caso, foi utilizado o diagrama entidade-relacionamento /MAR 85/ para representar como os relacionamentos ocorrem. Nesse caso, pode ser observada a utilização do símbolo  $\vdash$  para representar um acoplamento simples, i. é, com apenas um objeto. Da mesma forma, o símbolo  $\setminus/$  ("pé-de-galinha"), foi empregado para representar a existência de um acoplamento múltiplo, i. é, com mais de um objeto. Por exemplo, o módulo MOD4 possui um acoplamento simples do tipo S+ com o módulo de nome MOD3, que entretanto, apresenta um acoplamento múltiplo do tipo S- com o módulo MOD4, visto que existem dois módulos MOD3 e um módulo MOD4. Quanto ao tipo de relacionamento "parte de", deve ser considerado que um objeto pode ser constituído por vários outros, e dependendo do nível de detalhamento que se queira, podem ser obtidas visões /WIE 86/ diferentes do mesmo objeto, aplicando-se refinamentos funcionais /JAL 89/ que permitam visualizar ou

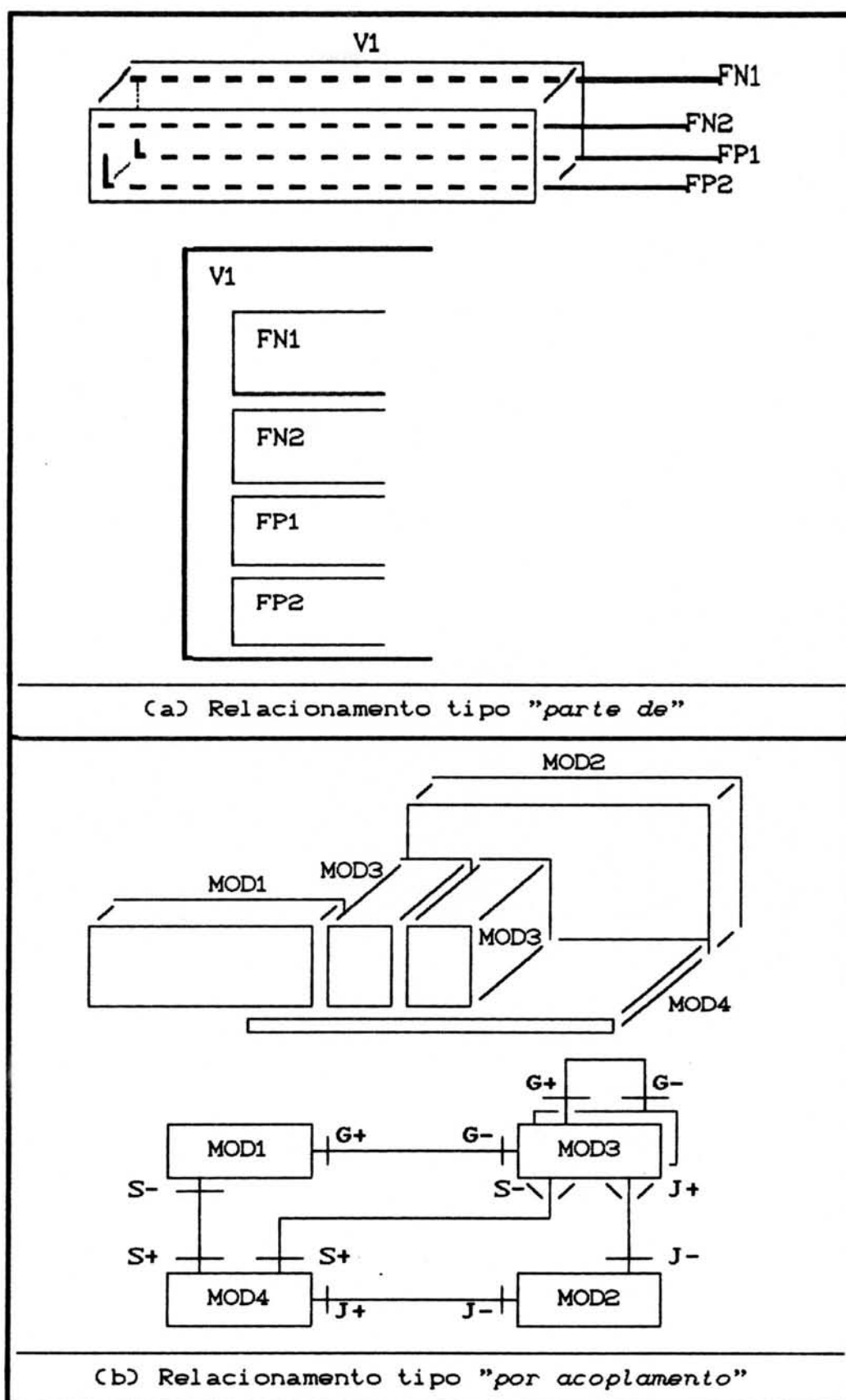
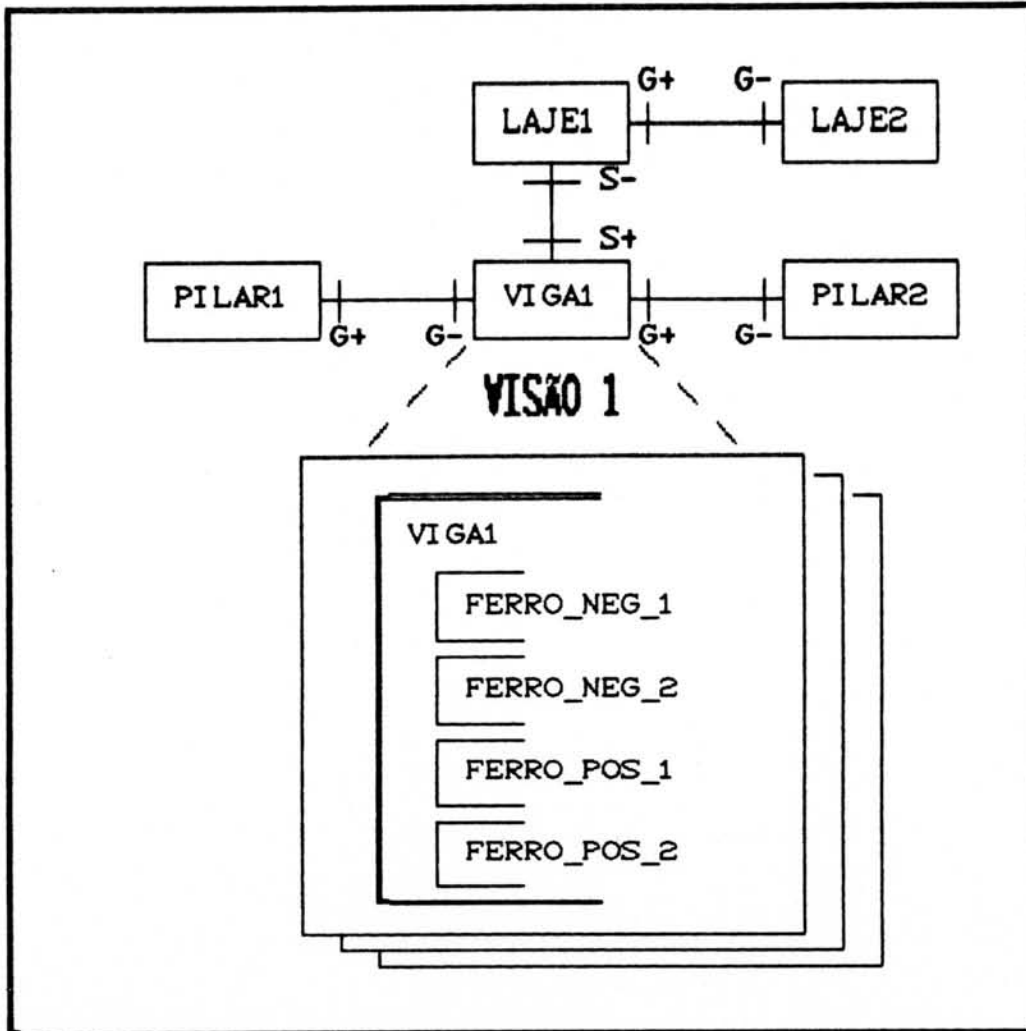


Figura 9.4 - Tipos de relacionamento.

emitir outros tipos de documentos, tais como relatórios ou relações de materiais, de partes específicas do produto que está sendo projetado /BJO 89/. Para representar este tipo de relacionamento foi adotado o *diagrama de ações* /MAR 85/ que pode ser visto na Figura 9.4(a). Neste exemplo, pode ser observada a existência de um volume principal V1 (viga de um edifício) e quatro outros volumes FN1, FN2, FP1 e FP2 (ferros negativos e positivos) que fazem parte de uma ocorrência da classe VIGA. Assim, em um nível de refinamento menor, os elementos FN1, FN2, FP1 e FP2 podem ficar *ocultos*, mas se for necessário elevar este nível, para permitir um melhor entendimento dessa parte do projeto, pode ser criada uma nova *visão* do objeto VIGA, onde esses elementos agora aparecem e fazem parte integrante da ocorrência. A Figura 9.5 mostra um esquema diagramático para representação dos elementos estruturais de um edifício com um refinamento funcional aplicado à ocorrência VIGA1 para exibição de uma *visão* das ferragens do referido elemento estrutural. Da mesma forma, o diagrama de ações pode sofrer refinamentos, permitindo assim, por exemplo, a representação das ações necessárias à geração dos *volumes* que constituem os objetos, assim como, das ações empregadas para gerar as relações de *classe, entidade* e outras que serão definidas a seguir.

Ainda sobre a questão dos refinamentos funcionais, este tipo de concepção de novos elementos e operações de projeto, pode ser vista como uma *abstração hierárquica* /BJO 89/, i. é, uma ferramenta que permite subdividir um objeto complexo em partes mais simples, estabelecendo também, os relacionamentos internos existentes entre estes elementos.



**Figura 9.5** - Esquema diagramático para representação de relacionamentos com uma visão de refinamento funcional.

### 9.4.1 Inerências

Por definição, *inerência* é qualidade de inerente, i. é, qualidade de algo "que por natureza está inseparavelmente ligado a alguma coisa" /FER 78/. Visto que objetos pertencem a classes e as classes de objetos podem apresentar ligações funcionais entre si, pode-se dizer que, algumas classes são *inerentes* (subentendem) a outras classes.



Para exemplificar o conceito de inerência entre classes, considere-se o exemplo de trabalho, onde podem ser estabelecidos os seguintes conceitos:

- 1 → As ocorrências da classe LAJE se apoiam em ocorrências da classe VIGA.
- 2 → As ocorrências da classe LAJE podem se acoplar lateralmente a outras ocorrências da classe LAJE.
- 3 → As ocorrências da classe VIGA podem se apoiar em ocorrências da classe PILAR.
- 4 → As ocorrências da classe VIGA podem se apoiar em outras ocorrências da classe VIGA.
- 5 → As ocorrências da classe PILAR se apoiam em ocorrências da classe FUNDAÇÃO.
- 6 → As ocorrências da classe PILAR podem se acoplar verticalmente à outras ocorrências da classe PILAR.

A Figura 9.6 mostra um diagrama das possíveis conexões funcionais entre os elementos de uma estrutura de edifício, fundamentada nos conceitos estabelecidos acima.

A seguir, será apresentado o comando da linguagem de ações responsável pela geração de inerências entre classes de objetos que tenham sido criadas anteriormente com o mecanismo ATR.

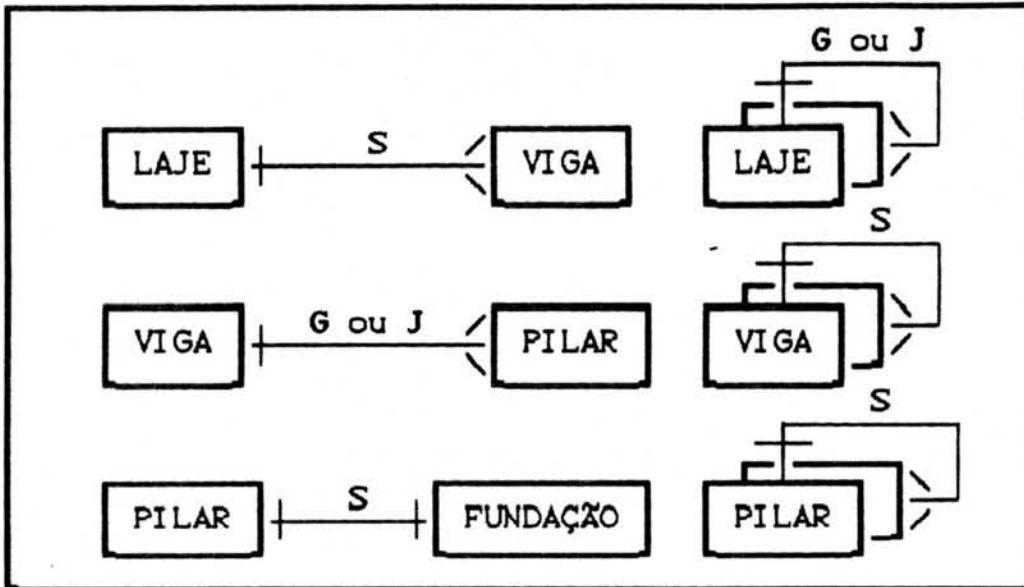


Figura 9.6 - Inerências entre as classes funcionais de uma estrutura de edifício.

INE<cl1,cl2,arq>

↑ Mecanismo para geração de inerências

Onde:

cl1 = nome da primeira classe  
(8 caracteres)

cl2 = nome da segunda classe  
(8 caracteres)

arq = nome do arquivo que vai conter as ocorrências das inerências (relacionamentos)  
(8 caracteres)

→ Exemplo:

INE<LAJE, LAJE, L&L>

INE<LAJE, VIGA, L&V>

INE<VIGA, VIGA, V&V>

INE<VIGA, PILAR, V&P>

INE<PILAR, PILAR, P&P>

INE<PILAR, FUNDAÇÃO, P&F>

Arquivo INERENC. CLA:

LAJE LAJE L&L

LAJE VIGA L&V

VIGA VIGA V&V

VIGA PILAR V&P

PILAR PILAR P&P

PILAR FUNDAÇÃO P&F

## 9.4.2 Geração dos relacionamentos

A geração das ocorrências de relacionamentos funcionais entre as entidades pertencentes às classes de objetos, cuja inerência foi estabelecida no arquivo INERENC.CLA, pode ser realizada mediante a utilização do mecanismo para criação de relacionamentos REL, que será definido a seguir.

REL<arq; en1, en2, aco>	
└─ Mecanismo para geração de relacionamentos	
Onde:	
arq = nome do arquivo de relacionamentos (8 caracteres)	
en1 = nome da primeira entidade (16 caracteres)	
en2 = nome da segunda entidade (16 caracteres)	
aco = tipo de acoplamento (2 caracteres)	
→ Exemplo:	Arquivo L&V.REL:
TNM<S(1); LAJE1>	LAJE1 VIGA1 S-
TNM<S(2); VIGA1>	LAJE1 VIGA2 S-
REL<L&V; S(1), S(2), S->	LAJE1 VIGA3 S-
TNM<S(2); VIGA2>	LAJE1 VIGA4 S-
REL<L&V; S(1), S(2), S->	
REL<L&V; S(1), VIGA3	, S->
REL<L&V; S(1), VIGA4	, S->

Observe-se no exemplo apresentado acima, as duas formas permissíveis de referenciar uma entidade dentro do comando REL. A primeira delas usa o mecanismo TNM, que

transforma um nome de entidade fornecido pelo usuário em um nome cujo formato é padronizado para satisfazer ao seu domínio dentro da relação de relacionamentos. A segunda

TNM<vax; nen>

↑  
└ Aplica o formato padrão ao nome da ent.

Onde:

vax = variável auxiliar que receberá o nome de entidade padronizado em 16 caracteres

nen = nome da entidade

(máximo de 16 caracteres)

forma de se referenciar uma entidade (modo direto), se for utilizada, deve levar em consideração o fato de que o nome da entidade deve ter 16 caracteres. Portanto, se o nome tiver um número inferior ao exigido por este domínio, deverão ser utilizados tantos espaços quantos forem necessários para completar este número de caracteres.

Outro mecanismo auxiliar que pode ser utilizado é o comando NEN, servindo para referenciar um nome de entidade, sendo diferente do comando TNM na forma de indicação da entidade, visto que permite indicar o nome da classe (8 caracteres no máximo) e o número da entidade. Assim, o comando concatena *cla* com *num* para construir um nome no formato padrão para nomes de entidade, com 16 caracteres.

NEN<vax; cla, num>

↑  
└ Cria um nome de entidade no formato padrão com 16 caracteres, transferindo este nome para uma variável auxiliar do tipo string.

Onde:

*vax* = variável auxiliar que receberá o nome da entidade padronizado.

*cla* = classe a que pertence a entidade (8 caracteres)

*num* = número da entidade (inteiro)

→ Exemplo:

NEN<S(1); VIGA, 5>

Efeito:

S(1) = 'VIGAS  
 └──────────┘  
 ↓  
 16 caracteres

## 9.5 Mecanismos para leitura de atributos

Após a geração das relações de entidades, os atributos destas poderão ser lidos e transferidos para variáveis auxiliares, respeitando-se os domínios. Esta operação de leitura é de fundamental importância para a aplicação de regras na tomada de decisões durante as etapas de elaboração de um projeto. Dessa forma, será possível o acesso às informações das características dos objetos envolvidos em um projeto, para um determinado instante do processo de geração. Note-se, ainda, que isto não significa que os valores acessados sejam valores definitivos, pois estes valores podem sofrer alterações durante a execução das diversas decomposições de ações funcionais.

A seguir, serão definidos os comandos para leitura de atributos para os domínios *inteiro*, *real* e *string*.

LAI<vai; cla, nen, ati>

└─ Lê atributo c/ domínio inteiro

LAR<var; cla, nen, atr>

└─ Lê atributo real c/ domínio real

LAS<vas; cla, nen, ats>

└─ Lê atributo string c/ domínio string

Onde:

vai = variável auxiliar (inteiro)

var = variável auxiliar (real)

vas = variável auxiliar (string)

cla = nome da classe

(8 caracteres)

nen = nome da entidade

(16 caracteres)

ati = atributo a ser lido (inteiro)

atr = atributo a ser lido (real)

ats = atributo a ser lido (string)

—→ Exemplos:

Efeitos(ex.):

NEN<S[1]; LAJE, 1>

LAI<I[10]; LAJE, S[1], XMIN>

I[10] = 500

TNM<S[2]; FERROS>

LAR<R[5]; FERRO, S[2], BITOLA>

R[5] = 4.2

NEN<S[3]; LAJE, 3>

LAS<S[7]; LAJE, S[3], UTILIZACAO>

S[7] = 'SALA'

## 9.6 Realizando alterações nas relações de entidade

Como foi mencionado anteriormente, os valores dos atributos não permanecem estáticos durante a execução das diversas etapas de um projeto. Assim, para satisfazer algumas necessidades específicas, tais como um

re-dimensionamento geométrico ou deslocamento de um elemento, pode ser preciso alterar o conteúdo de um conjunto de atributos. Além disso, a ação de alterar algumas características funcionais de um objeto individual pode produzir consequências importantes sobre outros elementos do projeto, sendo necessário, portanto, em alguns casos, alterar os valores de alguns atributos desses objetos diretamente atingidos.

Os comandos que serão definidos a seguir, constituem-se num instrumento versátil para se indicar o atributo da ocorrência de uma classe a ser modificado.

```
AAI<cla;nen,ati,vai>
  ↑
  └ Altera atributo inteiro
AAR<cla;nen,atr,var>
  ↑
  └ Altera atributo real
AAS<cla;nen,ats,vas>
  ↑
  └ Altera atributo string
```

Onde:

```
cla = nome da classe
      (8 caracteres)
nen = nome da entidade
      (16 caracteres)
ati = atributo a ser alterado (inteiro)
atr = atributo a ser alterado (real)
ats = atributo a ser alterado (string)
vai = valor do tipo inteiro
var = valor do tipo real
vas = valor do tipo string
```

→ Exemplo:	Efeitos:
$I=(I(1), 1)$	
$NEN<SI(1); VIGA, I(1)>$	
$AAI<VIGA; SI(1), ALTURA, 50>$	$\frac{\text{Altura}}{50}$
$NEN<SI(2); FERRO, 2>$	
$AAR<FERRO; SI(2), BITOLA, 5.0>$	$\frac{\text{Bitola}}{5.0}$
$TNM<SI(3); LAJE2>$	
$AAS<LAJE; SI(3), UTILIZACAO, BANHO>$	$\frac{\text{Utilização}}{\text{BANHO}}$

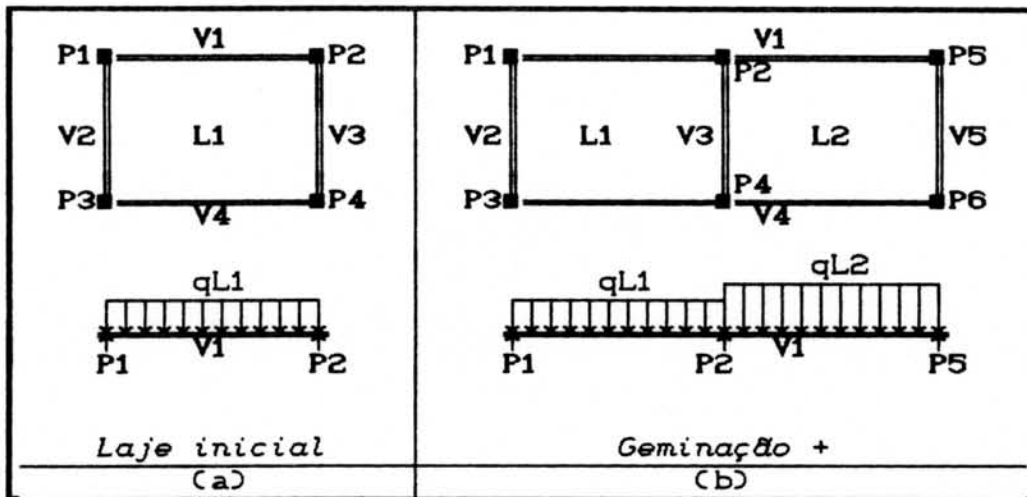


Figura 9.7 - Aplicação de uma geminação positiva a partir de uma laje inicial.

Para ilustrar a utilização dos mecanismos de alteração, considere-se as seguintes situações do exemplo de projeto de uma estrutura de edifício:

- 1 → Gera-se, numa primeira etapa, uma laje inicial, que se apoia em quatro vigas, que por sua vez, se apoiam nos quatro pilares posicionados nos vértices da laje (Figura 9.7(a), onde se colocam também os esquemas de cargas).



- <sup>2</sup> → *Aplica-se uma geminação positiva, considerando-se como semente o objeto laje inicial (Figura 9.7(b)).*
- <sup>3</sup> → Uma Conseqüência (automática): *As vigas V1 e V4 devem ser re-dimensionadas geometricamente (alteração no comprimento e na altura).*

## 9.7 Eliminação de uma entidade

Em determinadas situações de projeto, pode surgir a necessidade de *eliminar* uma entidade. Esta operação de remoção de um elemento pode ser motivada por *razões econômicas* ou *detalhes construtivos fundamentados na experiência*. Nesse caso, também devem ser consideradas as *conseqüências funcionais* que podem surgir a partir da remoção de um determinado elemento. Assim, por exemplo, não se pode simplesmente remover um pilar de uma estrutura. Outras providências (ações) devem ser executadas para impedir a queda da estrutura. Da mesma forma, não se pode retirar uma parede sem antes verificar se essa parede tem *função estrutural* ou se é apenas um simples *enchimento divisório*, sob pena de ocorrer um desabamento de toda a estrutura.

O comando ELE será empregado para referenciar a ocorrência de uma classe de objetos a ser eliminada da estrutura de dados do projeto.

ELE<cla;nen>  
 ↑  
 Elimina uma entidade.

Onde:

*cla* = nome da classe a que pertence a entidade  
(8 caracteres)

*nen* = nome da entidade a ser eliminada

→ Exemplo (Figura 9.8):

NEN<SI(1);PILAR,2>

NEN<SI(2);PILAR,4>

ELE<PILAR;SI(1)>

ELE<PILAR;SI(2)>

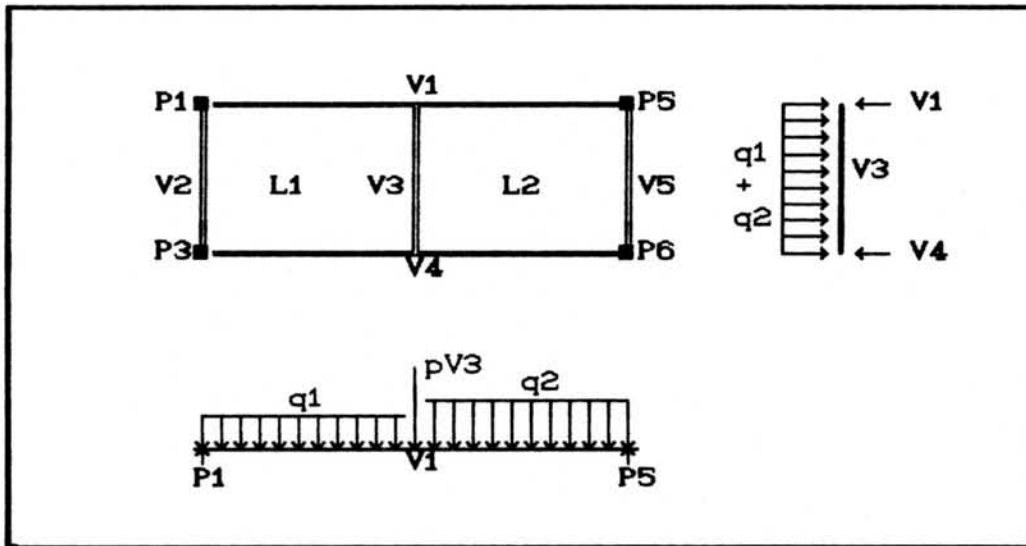


Figura 9.8 - Idem Figura 9.7(b) (geminação +), com eliminação dos objetos P2 e P4.

## 9.8 Eliminação de relacionamentos

Como foi mencionado, a eliminação de um conjunto de elementos pode provocar a propagação de consequências sobre um conjunto de partes de um projeto. Assim, não é suficiente remover um elemento. Deve-se, além disso, eliminar todos os relacionamentos funcionais que este

elemento possui com outras partes do todo projetado e ativar todas as ações necessárias para desencadear a totalidade dos efeitos produzidos pela remoção do elemento, ativando-se também todas as seqüências de ações para remediar estes efeitos, evitando-se conseqüências indesejáveis.

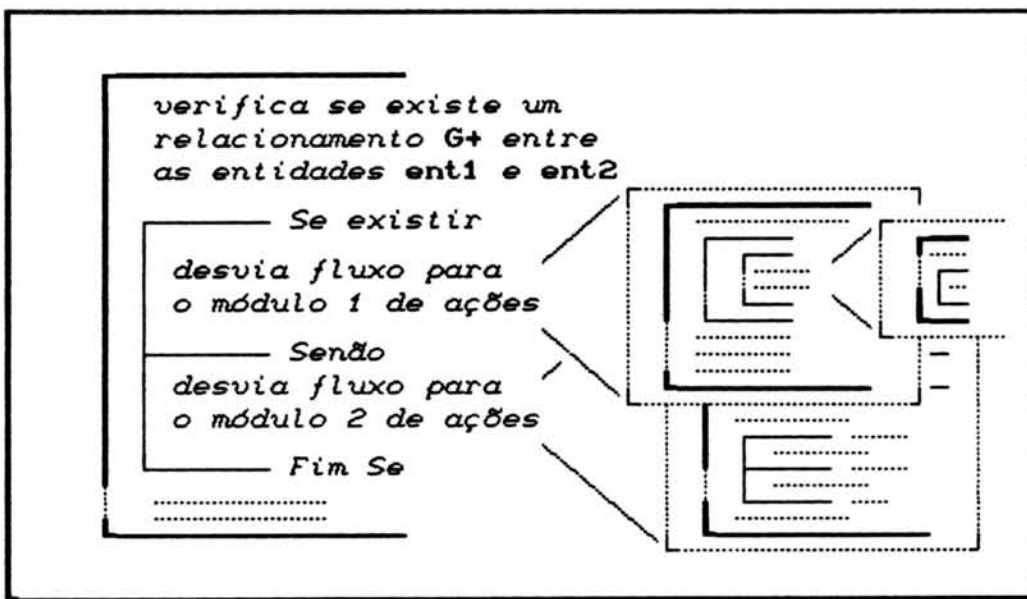
A seguir, será apresentado o mecanismo que permite remover todos os relacionamentos funcionais de uma entidade para uma determinada inerência.

<p>ELR&lt;arq;nen&gt;          ↑ Elimina os relacionamentos de uma entidade para uma determinada inerência</p>
<p>Onde:          arq = nome do arquivo de relacionamentos (8 caracteres)          nen = nome da entidade a ter seus relacionamentos eliminados</p>
<p>→ Exemplo:          NEN&lt;S(1);PILAR,2&gt;          NEN&lt;S(2);PILAR,4&gt;          ELR&lt;P&amp;V;S(1)&gt;          ELR&lt;P&amp;V;S(2)&gt;</p>

## 9.9 Leitura dos relacionamentos de uma entidade

Em algumas ocasiões durante um projeto, será necessário determinar quais os relacionamentos que uma determinada entidade possui com outras entidades. Assim, será possível descobrir se uma entidade apresenta, em um

dados instantâneos de projeto, um relacionamento do tipo *geminção positiva (G+)* com alguma outra entidade. Se existir tal relacionamento, dependendo do objetivo que se queira atingir, o fluxo de execução deverá ser desviado para uma seqüência de ações que considere essa instância. Se não existir, outras providências deverão ser adotadas, por exemplo, para remediar essa situação. Portanto, utilizando-se o mecanismo de controle, definido anteriormente neste trabalho, será possível criar novas decomposições de ações funcionais, estabelecendo-se a totalidade das ocorrências de casos possíveis (Figura 9.9).



**Figura 9.9** - Diagramas de ações para as diversas decomposições de ações funcionais em um projeto, com desvio do fluxo estabelecido pelo mecanismo de controle da linguagem de ações.

O mecanismo, definido a seguir, será empregado para fornecer os relacionamentos funcionais de uma entidade.

ENI<vax; arq, cla, nen, aco>

↑  
Fornece o relacionamento de uma entidade para um determinado acoplamento

Onde:

vax = variável auxiliar do tipo string que receberá o nome da entidade

arq = nome do arquivo de inerências  
(8 caracteres)

cla = classe da entidade a ser pesquisada  
(8 caracteres)

nen = nome da entidade a ser pesquisada  
(16 caracteres)

aco = tipo de acoplamento  
(2 caracteres)

→ Exemplo (Figura 9.8):

NEN<S[1]; LAJE, 1>

ENI<S[2]; L&L, LAJE, S[1], G+>

Efeito:

S[2] = 'LAJE2'

### 9.10 Mecanismo que fornece o nome da entidade que possui um determinado conjunto de atributos

O comando da linguagem de ações que será apresentado a seguir, permite especificar um conjunto de valores e a obtenção da entidade que possui essa ocorrência de atributos em um determinado instante da execução da sequência de ações de um projeto.

NEA<vax; cla, at1, at2, ..., atn>

↑  
Fornece o nome da entidade que possui o conjunto de atributos listados

Onde:

*vax* = variável auxiliar do tipo string que receberá o nome da entidade

*cla* = classe a que pertence a entidade a ser obtida (8 caracteres)

*at1, at2, ..., atn* = conjunto de atributos da entidade

(devem ser respeitados os domínios estabelecidos na relação de definição da classe correspondente)

\*\*\*OBS: O símbolo \* indica que o valor do atributo pode ser qualquer um.

—→ Exemplo:

NEA<SI1>; LAJE, \*, \*, 500, 0, \*, \*>

:=(I[1], 500)

:=(I[2], 400)

NEA<SI2>; VIGA, I[1], I[2], \*, \*, 12, \*>

## 9.11 Transferência do fluxo de ações

A modularização (atividade de projeto lógico, que tem por finalidade a definição de uma organização modular, executável em algum ambiente de programação não especificado /STA 83/) das ações funcionais de um projeto é uma metodologia de programação recomendada. Seus benefícios mais evidentes são /STA 83/:

- 1 → Criação de programas complexos em equipe, aumentando o grau de paralelismo e, conseqüentemente, reduzindo o tempo de desenvolvimento.
- 2 → Redução do tempo de compreensão, programação e teste, bem como redução da probabilidade de existência de falhas de programação.
- 3 → Permitir a aplicação de testes quase completos em módulos pequenos.
- 4 → Redução no esforço de manutenção.
- 5 → Reutilização de um módulo como componente de outra seqüência de ações.

Por exemplo, a geração de um objeto laje inicial, utilizando-se a linguagem de ações, pode ser feita estabelecendo-se a seguinte organização modular (Figura 9.10):

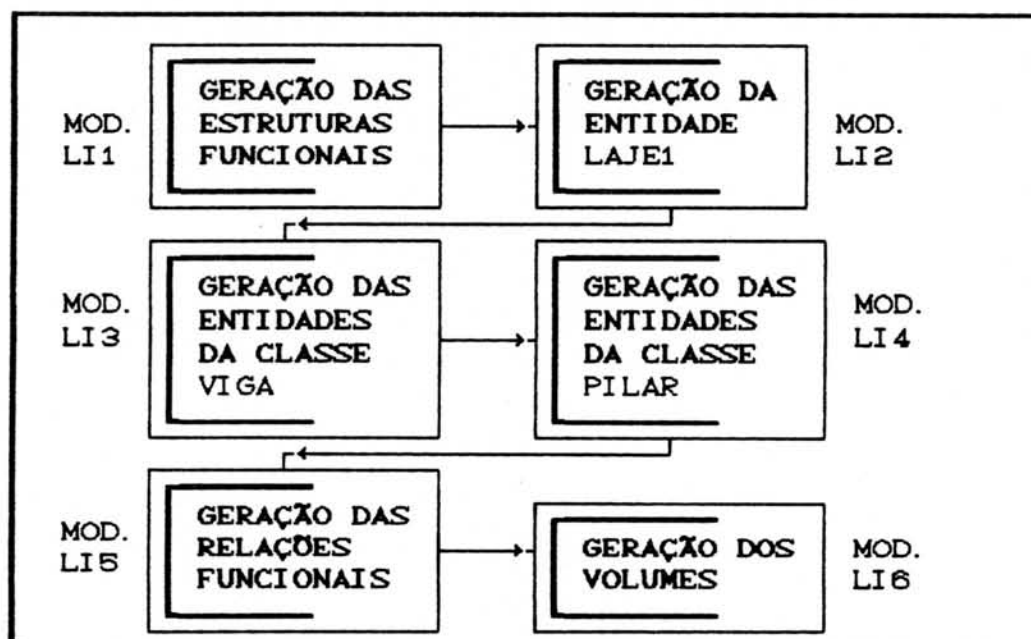



Figura 9.10 - Organização modular para geração da laje inicial.

Assim, o módulo LI1 cria as relações de atributos das classes LAJE\_SEMENTE, VERTICE\_POSICAO, VERTICE\_VIGAS, LAJE, VIGA e PILAR, além da relação de *inerências*. O módulo LI2 será o responsável pela entrada de dados da laje inicial (dimensões, utilização etc) e pela manipulação desses dados para definição dos atributos dessa entidade. O módulo LI3 vai manipular e gerar as entidades da classe VIGA e o módulo LI4 fará o mesmo com as entidades da classe PILAR. Posteriormente, o módulo LI4 vai estabelecer os relacionamentos produzidos entre as entidades geradas. Finalmente, o módulo LI6 fará a leitura das características geométricas das entidades que foram criadas, gerando os *volumes* dessas entidades, utilizando os mecanismos para geração de volumes apresentados em capítulos anteriores deste trabalho. Esta operação de geração de volumes (utilizando-se uma representação do tipo "wireframe") vai originar a produção de outras relações (na memória do computador), para representação dos *vértices, arestas, faces etc.*

A interligação entre os módulos será feita por meio de um mecanismo que permite *transferir o controle de execução das ações* de um módulo para outro. Este comando (TCA) será definido a seguir.

<p>TCA&lt;arq&gt;   Transfere o controle de execução das ações</p>
<p>Onde:  arq = Nome do arquivo de ações (.REG)  (8 caracteres)</p>
<p>→ Exemplo:  TCA&lt;LI2&gt;</p>



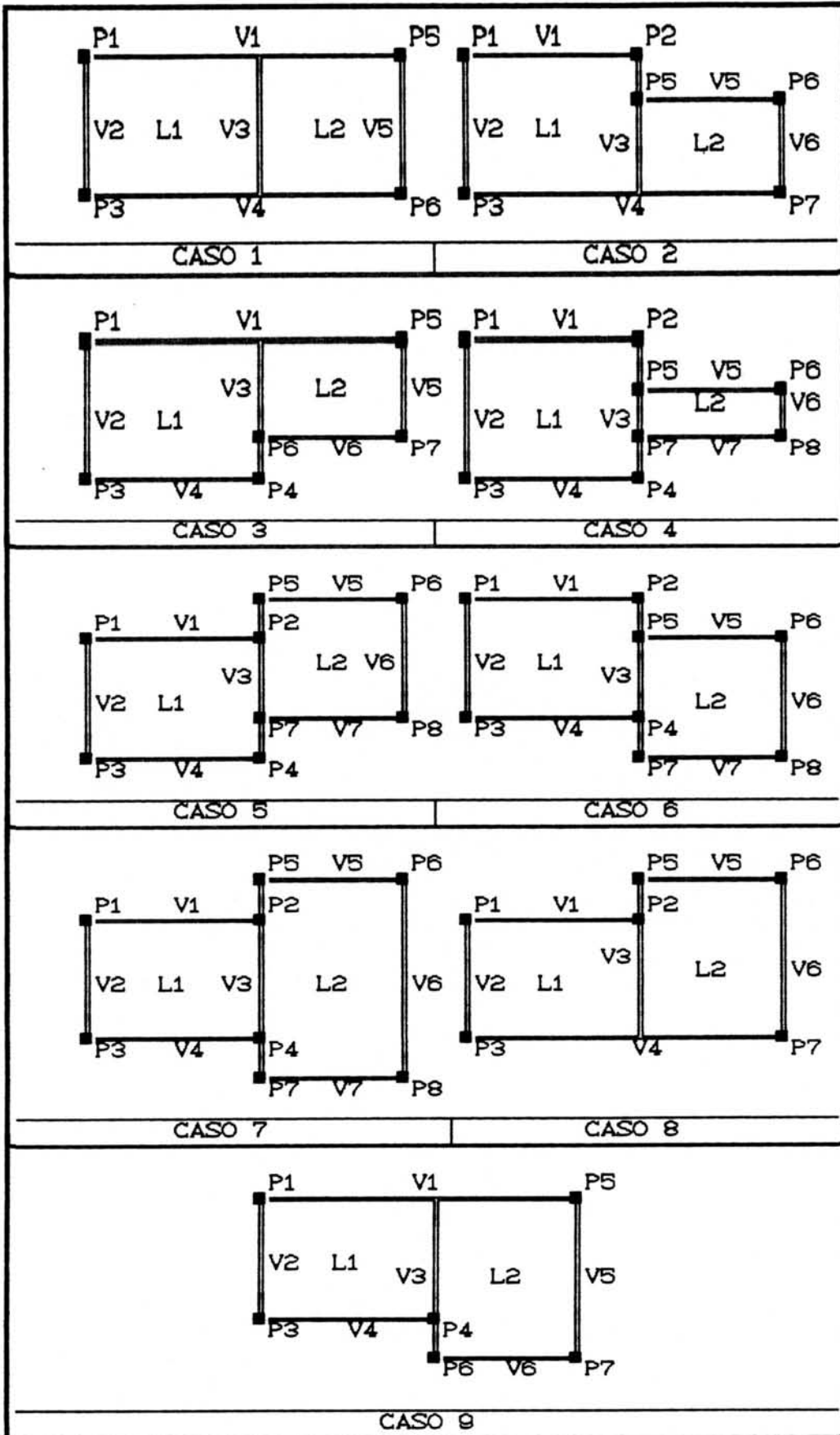


Figura 9.11 - Casos de uma geminação positiva.

Note-se que o mecanismo de controle, visto no Capítulo 8, pode ser utilizado como um mecanismo de decisão relativo ao fluxo da seqüência de ações a ser seguido em um projeto. Assim, por exemplo, vários casos podem ser considerados ao ser aplicada uma *geminção positiva* a partir de uma semente laje inicial. A Figura 9.11 mostra alguns desses casos e a Figura 9.12 apresenta a respectiva organização modular.

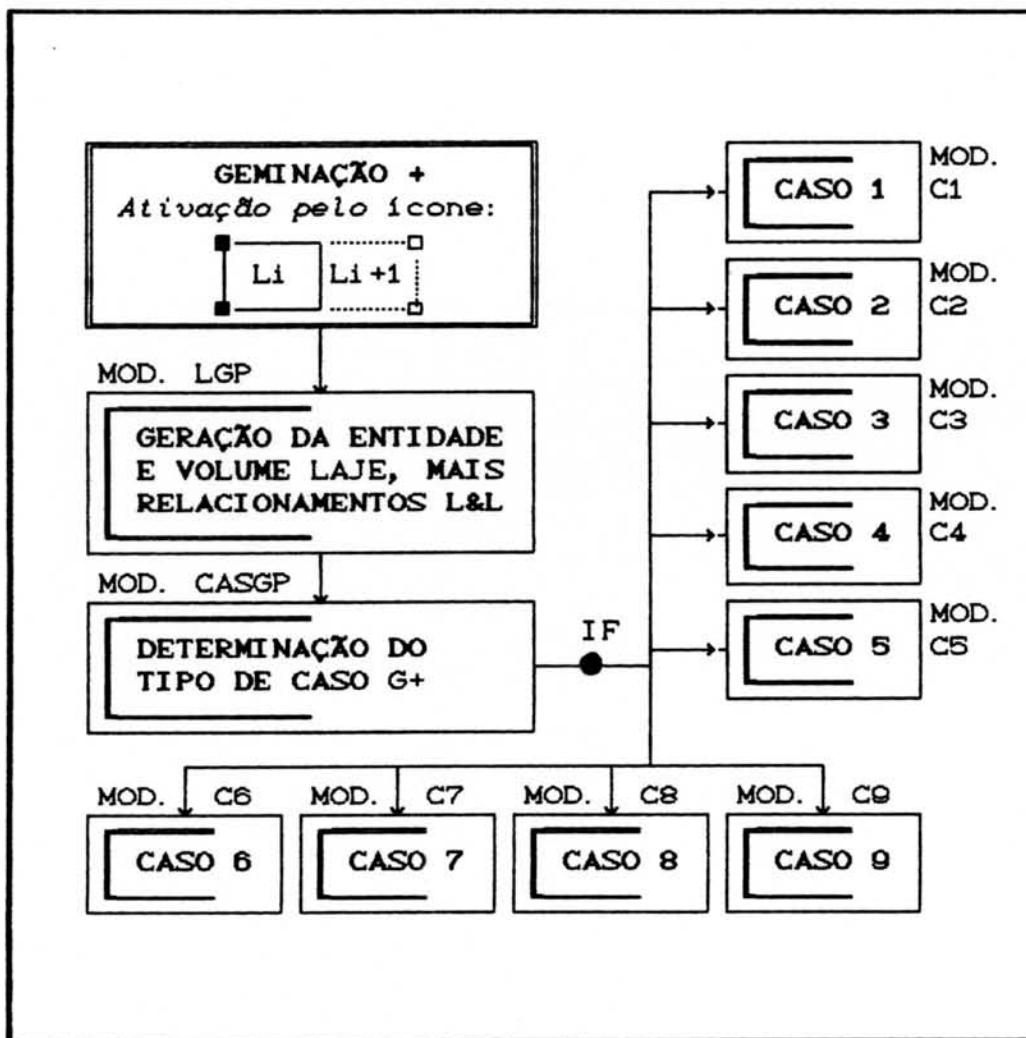


Figura 9.12 - Organização modular para uma *geminção positiva*, a partir da semente laje inicial.

Tabela 9.1 - Resumo dos mecanismos e símbolos do capítulo

---

<b>Comandos para geração de relações funcionais</b>	
<hr/>	
ATR<cla;dom,atr>	= cria atributos de uma classe de objetos
ENT<cla;nun,at1,at2,...,atn>	= cria uma entidade
INE<cl1,cl2,arq>	= gera inerências
REL<arq;en1,en2,aco>	= gera relacionamentos
TNM<vax;nen>	= aplica o formato padrão ao nome de entidade
NEN<vax;cla,num>	= cria um nome de entidade no formato padrão com 16 caracteres, transferindo este nome para uma variável auxiliar do tipo <i>string</i>
LAI<vai;cla,nen,ati>	= lê atributo inteiro
LAR<var;cla,nen,atr>	= lê atributo real
LAS<vas;cla,nen,ats>	= lê atributo <i>string</i>
AAI<cla;nen,ati,vai>	= altera atributo inteiro
AAR<cla;nen,atr,var>	= altera atributo real
AAS<cla;nen,ats,vas>	= altera atributo <i>string</i>
ELE<cla;nen>	= elimina uma entidade
ELR<arq;nen>	= elimina relacionamentos de uma entidade para uma determinada inerência
ENI<vax;arq,cla,nen,aco>	= fornece o relacionamento de uma entidade para um determinado acoplamento
NEA<vax;cla,at1,at2,...,atn>	= fornece o nome da entidade que possui o conjunto de atributos listados
TCA<arq>	= transfere o controle de execução das ações
<hr/>	
<b>Identificadores</b>	
<hr/>	
→ cla	= nome da classe (8 caracteres)
→ dom	= domínio do atributo (1 caracter)
	I ← inteiro
	R ← real
	S ← <i>string</i>
→ atr	= nome do atributo (16 caracteres)
→ nun	= número da entidade (inteiro)
→ at1,at2,...,atn	= valores das ocorrências dos atributos da classe (devem ser respeitados os domínios estabelecidos na relação de definição da classe correspondente)

---

**Tabela 9.1(cont.) - Resumo dos mecanismos e símbolos do capítulo**

<b>Identificadores</b>	
→	cl1 = nome da classe 1 (8 caracteres)
→	cl2 = nome da classe 2 (8 caracteres)
→	arq = nome do arquivo (8 caracteres)
→	en1 = nome da entidade 1 (16 caracteres)
→	en2 = nome da entidade 2 (16 caracteres)
→	aco = tipo de acoplamento (2 caracteres)
→	vax = variável auxiliar
→	nen = nome da entidade (16 caracteres)
→	vai = variável auxiliar (inteiro)
→	var = variável auxiliar (real)
→	vas = variável auxiliar (string)
→	ati = atributo (inteiro)
→	atr = atributo (real)
→	ats = atributo (string)
<b>Arquivos principais</b>	
	arq.REG = arquivo de ações
	arq.ICO = arquivo de definição de um ícone
<b>Arquivos gerados pelos comandos do capítulo</b>	
	cla.ATR = atributos de uma classe
	nen.ENT = valores das ocorrências dos atributos de uma classe (entidades)
	INERENC.CLA = inerências entre classes
	arq.REL = relacionamentos entre entidades
<b>Símbolo</b>	
→	* = qualquer valor

## 10. APLICAÇÕES

A geração de estruturas de concreto armado, vista no capítulo anterior deste documento, constitui-se num exemplo simplificado de aplicação dos conceitos e mecanismos apresentados. Assim, um número ilimitado de outras aplicações, mais elaboradas, em diversas áreas da ciência, podem ser produzidas utilizando-se um *modelamento funcional de objetos*, como será visto neste capítulo, abstraído-se maiores detalhes que fogem ao escopo deste trabalho.

### 10.1 Biologia

As leis de Mendel (1822-1884), a respeito da hereditariedade, estabelecem regras que mostram como se realiza a transmissão dos caracteres nos cruzamentos entre os seres vivos (LAR 62). De acordo com Mendel, os fatores hereditários são transmitidos de pai para filho através dos gametas (*células sexuais*), permanecendo segregados uns dos outros, sem se misturarem. Note-se, ainda, que os efeitos produzidos por esta ação hereditária podem ser mascarados pela dominância (*caracter dominante = aquele que prevalece*) de um sobre o outro (*caracter recessivo = aquele que fica encoberto ou mascarado*), embora isso nem sempre aconteça. A partir desse fenômeno, Mendel estabeleceu a seguinte lei:

—> Primeira Lei de Mendel: "Os filhos de dois indivíduos puros, para certo caráter, serão todos iguais. Mas seus netos estarão

na proporção de 3:1 se houver dominância de um dos fatores, e de 1:2:1 se não houver".

Em uma de suas pesquisas, Mendel fez cruzamentos entre ervilhas verdes e ervilhas amarelas, obtendo resultados que posteriormente o ajudaram a formular sua primeira lei (i. é, uma regra de hereditariedade).

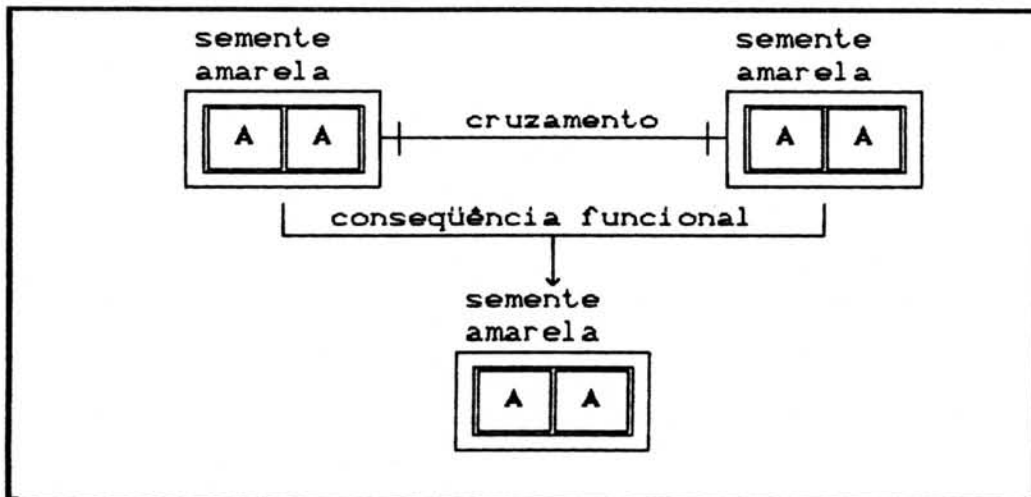


Figura 10.1 - Relacionamento entre duas entidades da classe *semente amarela*.

Observe-se que as sementes podem ser vistas como objetos, que possuem atributos especiais (fatores genéticos) e podem apresentar relacionamentos entre si (cruzamentos). A Figura 10.1 mostra um esquema diagramático para representar o relacionamento (cruzamento) entre duas entidades da mesma classe (*semente amarela*), e a Figura 10.2 faz o mesmo para um cruzamento entre *sementes verdes*. Nestas figuras, note-se que A é o fator das sementes amarelas, v o das sementes verdes. Além disso, cada célula sexual dá um fator (gen) à semente e cada semente tem, portanto, dois fatores. Assim, num relacionamento de duas

entidades da classe *semente amarela*, os fatores da entidade resultante deste cruzamento serão AA, e, num relacionamento de dois verdes, vv. Note-se, além disso, que os esquemas genéticos usam letras maiúsculas para indicar *fatores dominantes* e minúsculas para os *recessivos*.

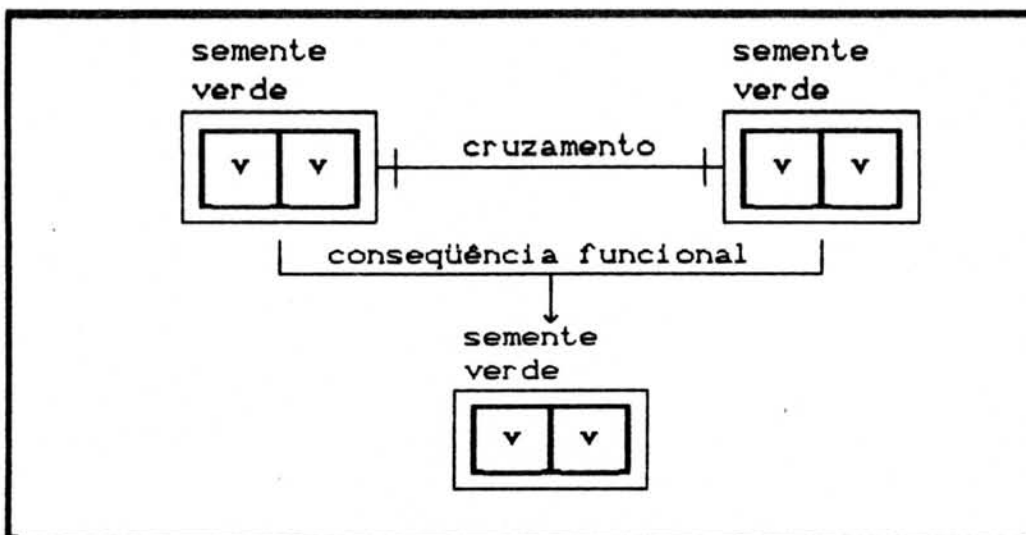


Figura 10.2 - Relacionamento entre duas entidades da classe *semente verde*.

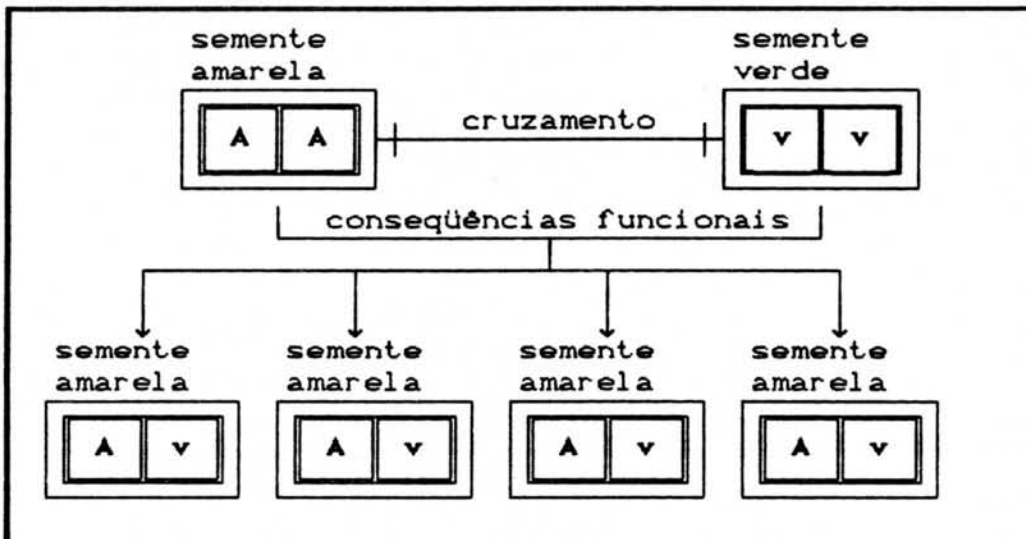


Figura 10.3 - Relacionamento entre uma entidade da classe *semente amarela* e uma *verde* (primeira geração).

A seguir, a Figura 10.3 mostra um relacionamento entre uma entidade da classe *semente amarela* com uma entidade da classe *semente verde*. Note-se que as sementes resultantes são *amarelas*, mas dentro delas estão ocultos os fatores *v* que produzirão os verdes da próxima geração.

Na segunda geração de híbridos, um quarto das sementes nascidas será verde, e só terão filhos verdes se continuarem a autofecundar-se. Serão sempre *vv*. Um quarto é de amarelos puros e será *AA*. Note-se, ainda, que dois quartos serão híbridos *Av*, que nas próximas gerações continuarão a decompor-se da mesma maneira. A Figura 10.4 mostra a segunda geração de sementes.

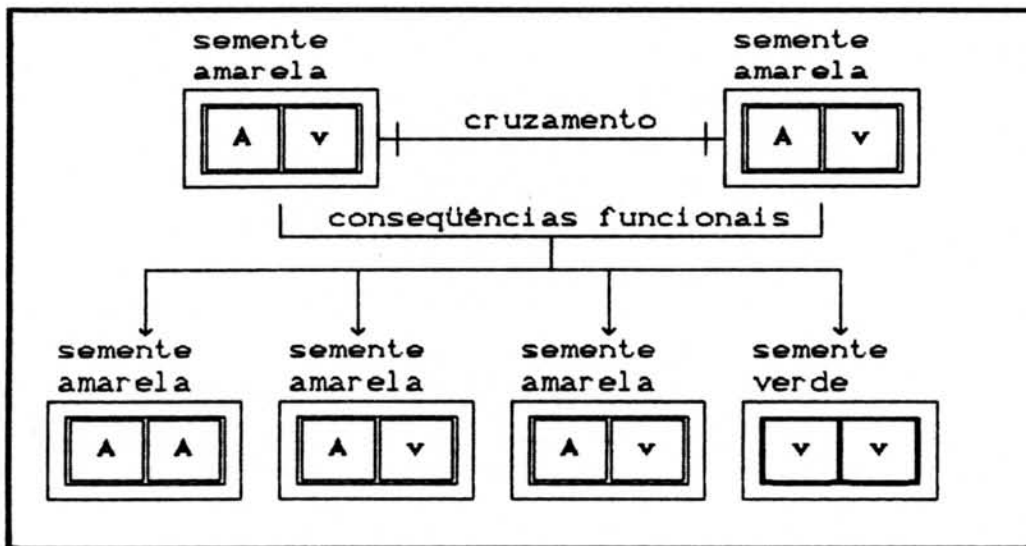


Figura 10.4 - Relacionamento entre duas sementes amarelas, na segunda geração.

Assim, os conceitos de hereditariedade vistos acima, podem ser programados utilizando-se a linguagem de ações, e as várias gerações de cruzamento de sementes podem ser vistas como níveis de decomposição de ações hereditárias. Dessa forma, utilizando-se este tipo de modelamento funcional, torna-se possível representar graficamente, na forma tridimensional (tudo o que existe na



natureza é tridimensional - a representação plana é apenas um tipo de representação, um modelo simplificado), as projeções para as ocorrências de sementes em vagens de ervilha, como pode ser visto esquematicamente na Figura 10.5.

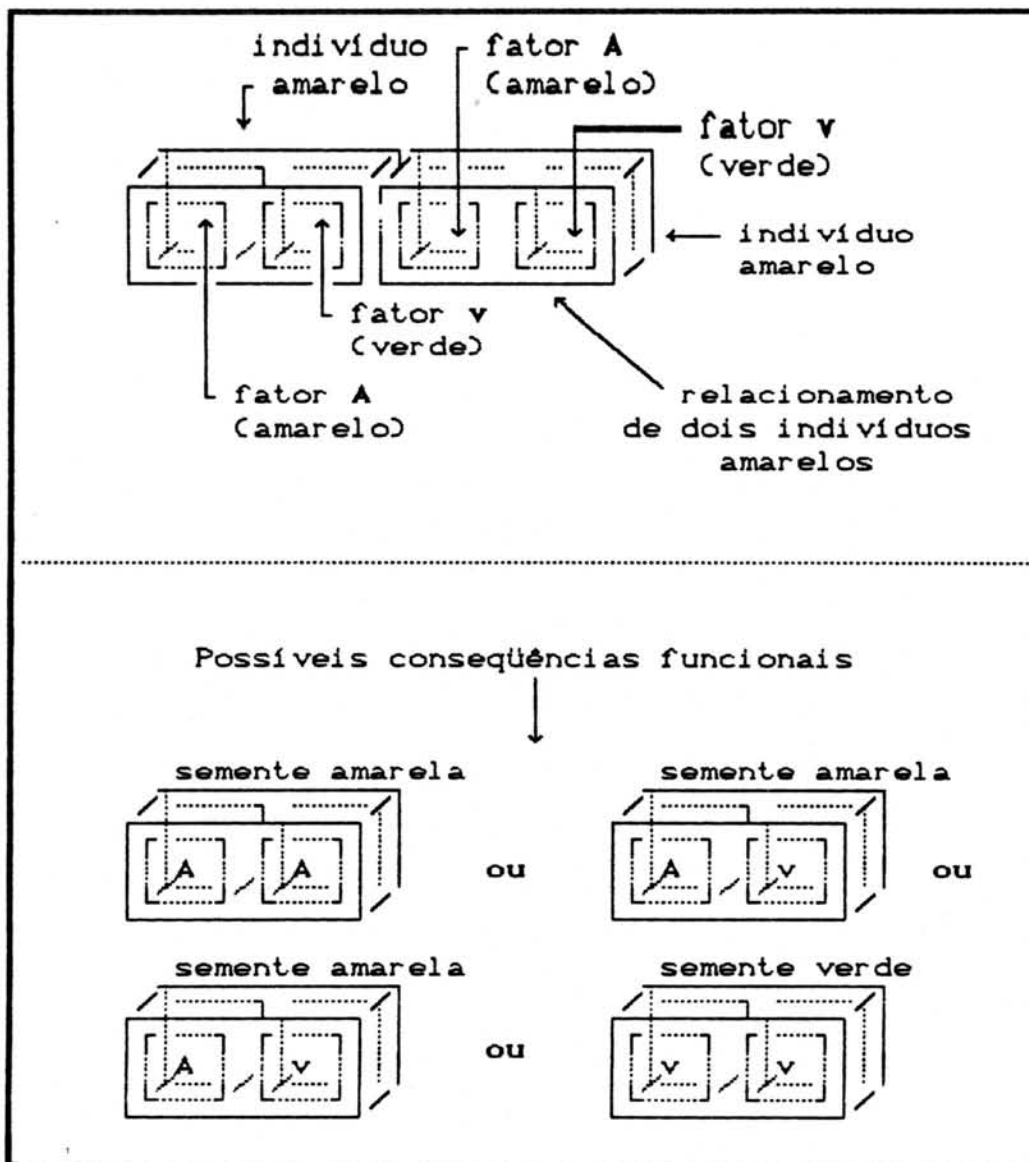


Figura 10.5 - Representação tridimensional de um cruzamento entre dois indivíduos amarelos híbridos.

Assim, os indivíduos verdes e amarelos puros ou híbridos poderiam ser representantes da classe:

Arquivo INDIVID. ATR:  
 I GERACAO  
 S FATOR\_GENET1  
 S FATOR\_GENET2  
 ...

→ Exemplos:

INDIVID. ENT:  
 INDIVID1 1 A v  
 INDIVID2 1 A v  
 INDIVID3 2 A A  
 ...

Da mesma forma, os relacionamentos (cruzamentos) entre os indivíduos podem ser representados pelas relações:

Arquivo INERENC. CLA:  
 INDIVID INDIVID IND&IND

Arquivo IND&IND. REL:

INDIVID1 INDIVID2 1  
 ...

↑  
 campo modificado para indicar a geração em que ocorre o relacionamento

Note-se, ainda, que nesta aplicação seria necessário criar um novo mecanismo que permitisse a escolha ao acaso do fluxo de ações, visto que, no exemplo citado, a ação de um cruzamento entre dois indivíduos pode produzir quatro conseqüências funcionais diferentes, mas somente uma delas deverá ocorrer.

## 10.2 Química

A química estabelece que a matéria é constituída de partículas extremamente pequenas, denominadas *moléculas*, separadas umas das outras por espaços vazios. Note-se que, neste caso, as moléculas também podem ser vistas como *objetos tridimensionais*. Além disso, as *entidades moleculares* são constituídas por partículas indivisíveis, denominadas *átomos*, i. é, os *elementos químicos* (H = hidrogênio, C = carbono, O = oxigênio etc). As moléculas e seus *elementos químicos* também apresentam propriedades especiais, *atributos químicos*, tais como: *massa atômica*, *massa molecular*, *volume atômico*, *volume molecular*, *valência* etc.

As moléculas das substâncias simples podem ser combinadas, gerando um *relacionamento químico*, onde devem ser considerados os *grupamentos funcionais* (*radicais* ou *átomos*) envolvidos no processo. Os relacionamentos químicos podem ser representados esquematicamente através de *reações químicas*, onde devem ser consideradas as fórmulas das moléculas das substâncias que reagem (*se relacionam funcionalmente*) para produzir outras substâncias. Assim, os efeitos produzidos pela reação, a partir do conhecimento de determinadas *regras* ou *leis da química*, como por exemplo, o *princípio da conservação dos elementos*, poderiam ser considerados utilizando-se *seqüências de ações moleculares*.

Portanto, as *ligações moleculares* entre os elementos químicos poderiam ser representadas na sua forma tridimensional, possibilitando dessa forma, a observação das estruturas moleculares espaciais ou *estereoquímicas* das substâncias químicas em sua disposição real no espaço. Como exemplo, pode ser citado o trabalho de *Le Bel* e *Van't Hoff*, que concluíram que as quatro valências do átomo de carbono não estão dispostas num mesmo plano, mas estão, ao

contrário, orientadas como retas que, do centro de gravidade de um tetraedro regular, se dirigem para os quatro vértices. Além disso, essas quatro direções formam entre si ângulos iguais, fato que traduz a equivalência das quatro ligações do carbono. Então, nessas condições, o metano,  $\text{CH}_4$ , apresenta a configuração espacial mostrada na Figura 10.6.

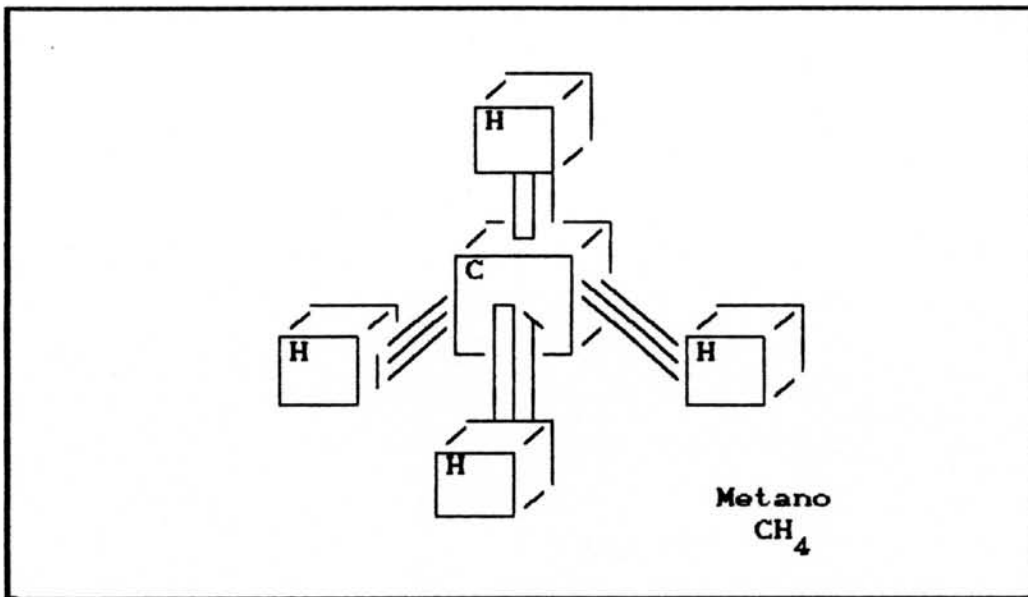


Figura 10.6 - Configuração espacial do metano.

A Figura 10.7 mostra a consequência funcional (numa representação simplificada plana) de uma tripla ligação molecular (triplo relacionamento entre entidades moleculares), onde três moléculas de acetileno  $\text{C}_2\text{H}_2$  produzem uma molécula de benzeno  $\text{C}_6\text{H}_6$ , numa reação conhecida como *polimerização*.

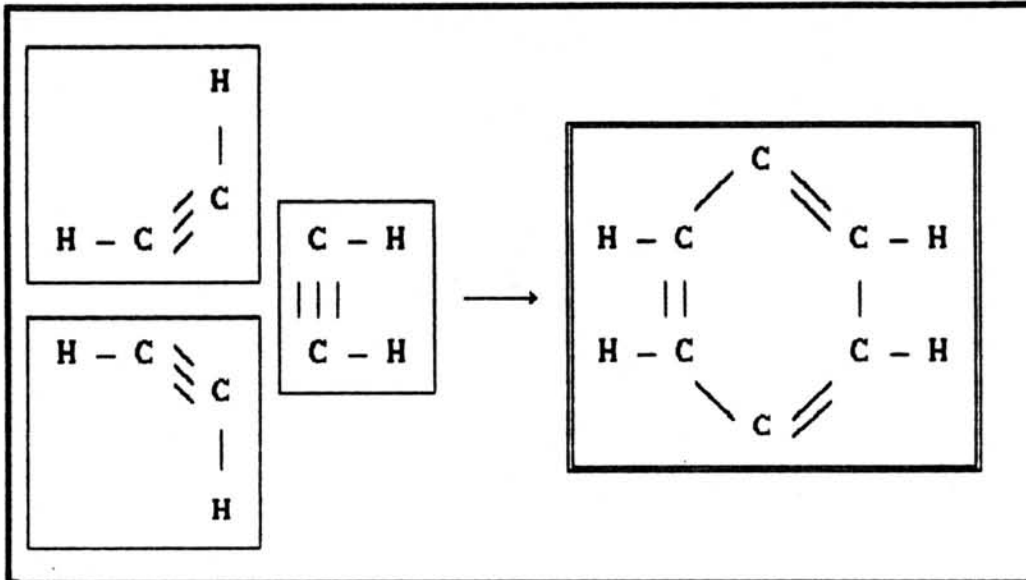


Figura 10.7 - Conseqüência funcional de um triplo relacionamento entre entidades moleculares.

Então, os elementos químicos e as moléculas poderiam ser representados pelas seguintes classes de objetos químicos funcionais:

<u>ELEMENTO. ATR:</u>	<u>MOLECULA. ATR:</u>
S NOME	S SUBSTANCIA
I NUMERO ATOMICO	S ELEMENTO1
R MASSA ATOMICA	S ELEMENTO2
...	S ELEMENTO3
	...

—→ Exemplos:

ELEMENTO. ENT:

ELEMENTO1	H1	1	1.008	...
ELEMENTO2	H2	1	1.008	...
ELEMENTO3	O1	8	16.00	...

...

MOLECULA. ENT:

MOLECULA1	H2O_1	H1	H2	O1	#	#	...
MOLECULA2	NaCl_1	Na1	Cl1	#	#	#	...

Da mesma forma, os relacionamentos químicos podem ser representados pelas relações:

<u>INERENC. CLA:</u>		
ELEMENTO	ELEMENTO	ELE&ELE
MOLECULA	MOLECULA	MOL&MOL
...		
<u>ELE&amp;ELE:</u>		<u>MOL&amp;MOL:</u>
H1	O1 H2O_1	H2O_1 NaCl_1 REACAO1
H2	O1 H2O_1	...
...	↑ campo modificado para indicar a substância	↑ campo modificado para indicar a reação

Vale salientar que a introdução das regras químicas a serem introduzidas no sistema (utilizando-se a linguagem de ações), de forma a enriquecê-lo com mais conhecimentos funcionais na área, deve ser feita com a colaboração de um *especialista químico*, que é a pessoa indicada para este tipo de tarefa, visto que o fator *experiência* é de fundamental importância para a obtenção de um produto.

### 10.3 Matemática

Também na *Computação Algébrica*, um *Sistema de Modelamento de Objetos Funcionais* pode ser aplicado a um conjunto de *termos algébricos* representados por figuras geométricas tridimensionais (*objetos algébricos*), para produzir um resultado, i. é, uma *consequência funcional matemática*. Nessa aplicação, os relacionamentos matemáticos são os *operadores matemáticos*, ativados pelos ícones

operacionais, que ao serem aplicados às entidades de termos algébricos podem produzir uma entidade-solução.

Para exemplificar as idéias apresentadas acima, considere-se a seguinte equação algébrica:

$$2 + X1 + 2 = 10$$

Assim, a classe UNIDADE deverá ser criada para representar as ocorrências 2 e 10, sendo a incógnita X1 uma entidade da classe X, como mostrado a seguir:

<u>UNIDADE.ATR:</u>	<u>X.ATR:</u>
SINAL	SINAL
VALOR_ABSOLUTO	VALOR_ABSOLUTO
...	...
<u>UNIDADE.ENT:</u>	<u>X.ENT:</u>
UNIDADE1 + 2 ...	X1 ? ? ...
UNIDADE2 + 2 ...	
UNIDADE3 + 10 ...	

Note-se, ainda, que as ocorrências de termos algébricos podem ser representadas por volumes tridimensionais. Para tal, será estabelecido o seguinte conceito inicial, que será utilizado como unidade básica para representação tridimensional de termos algébricos:

Unidade algébrica computacional: É um cubo cujas arestas são unitárias (Figura 10.8).

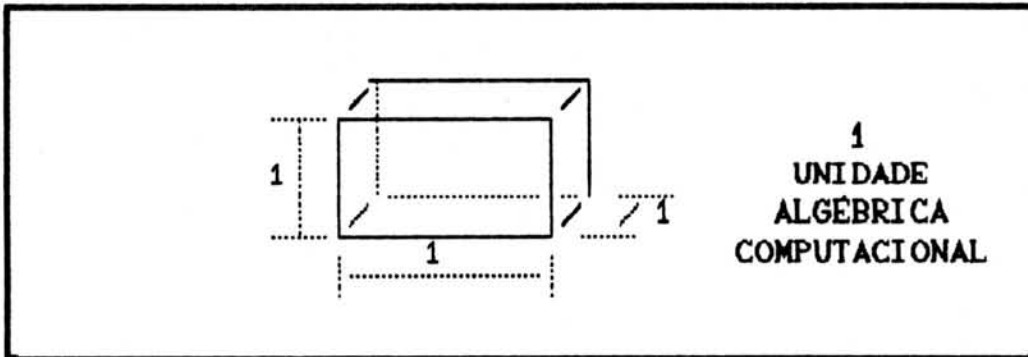


Figura 10.8 - Unidade algébrica computacional.

A Figura 10.9 mostra a representação gráfica tridimensional para os objetos matemáticos da equação algébrica do exemplo apresentado.

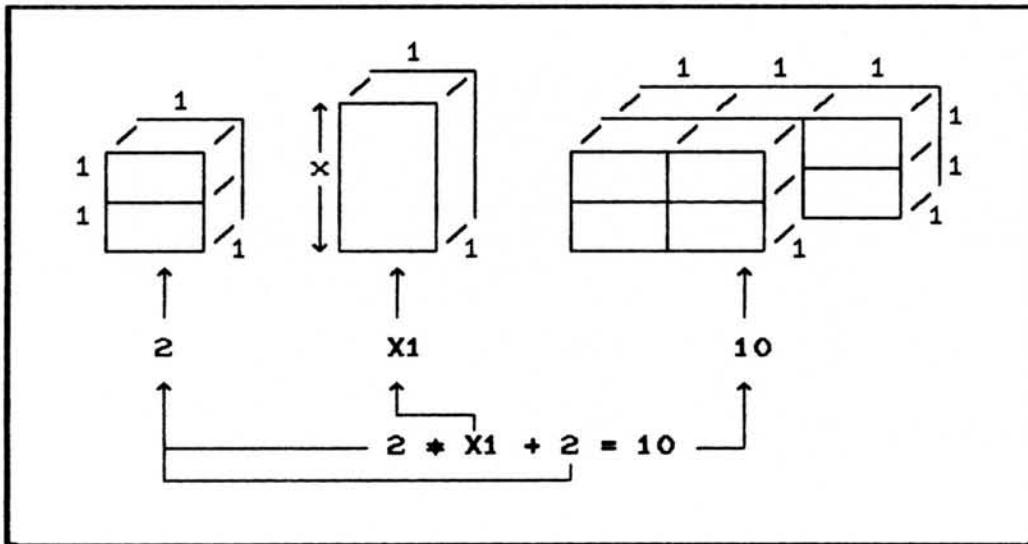


Figura 10.9 - Representação 3-D para os termos algébricos da equação  $2 * X1 + 2 = 10$ .

Analogamente, os termos algébricos  $X^2$  e  $X^3$  poderiam ser representados tridimensionalmente, como pode ser visto na Figura 10.10. Note-se, portanto, que o termo linear  $X$  equivale a um volume que possui duas dimensões unitárias e uma incógnita, o termo  $X^2$  equivale a uma placa de espessura unitária e o termo  $X^3$  equivale a um cubo cujas arestas possuem a dimensão incógnita  $X$ .



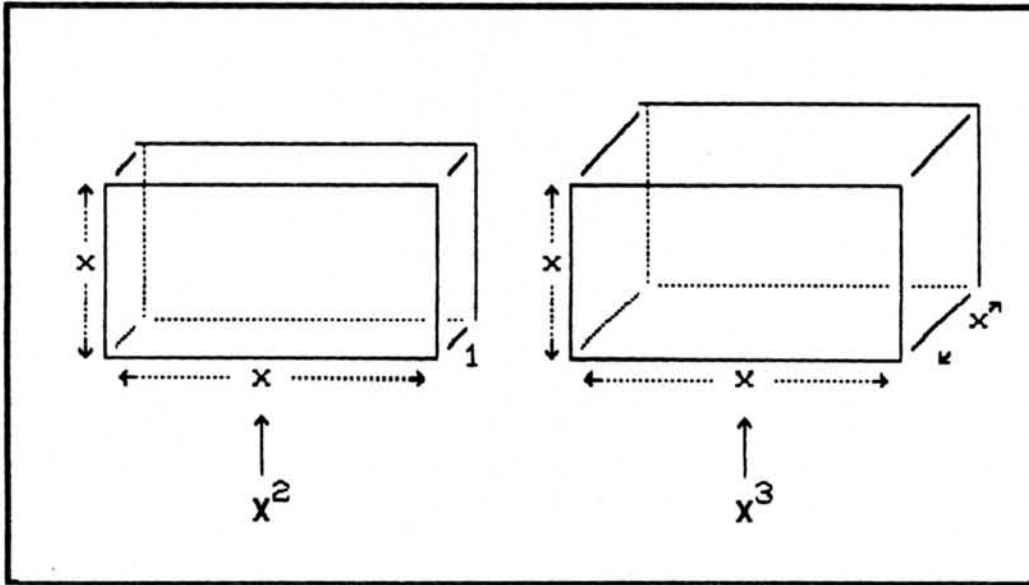


Figura 10.10 - Representação 3-D para os termos  $x^2$  e  $x^3$ .

Assim, as operações algébricas poderiam ser aplicadas aos objetos algébricos na forma de relacionamentos algébricos. Então, inicialmente, haveria quatro relacionamentos algébricos básicos (equivalentes às quatro operações básicas), mais o relacionamento de igualdade:

1	→	Relacionamento do tipo SOMA	→	+
2	→	Relacionamento do tipo SUBTRAÇÃO	→	-
3	→	Relacionamento do tipo MULTIPLICAÇÃO	→	*
4	→	Relacionamento do tipo DIVISÃO	→	/
5	→	Relacionamento do tipo IGUALDADE	→	=

Portanto, os relacionamentos podem ser representados pelas seguintes inerências entre classes:

INERENC. CLA:		
UNIDADE	X	UN&X
UNIDADE	UNIDADE	UN&UN
X	X	X&X

Para a equação  $2 \times X1 + 2 = 10$  podem ser estabelecidos os seguintes relacionamentos:

<u>UN&amp;X.REL:</u>			<u>UN&amp;UN:</u>		
UNIDADE1	X1	*G+	UNIDADE2	UNIDADE3	=G+
UNIDADE2	X1	+G-			

Note-se que os tipos de acoplamento (G+, G-, J+, J-, S+ e S-) também podem ser utilizados para indicar o sentido do relacionamento, estabelecendo dessa forma, qual o operador que está à esquerda e qual o que está à direita (no caso de um relacionamento do tipo G). Então, os elementos de uma matriz tridimensional podem ser representados por relacionamentos G, J e S (Figura 10.11):

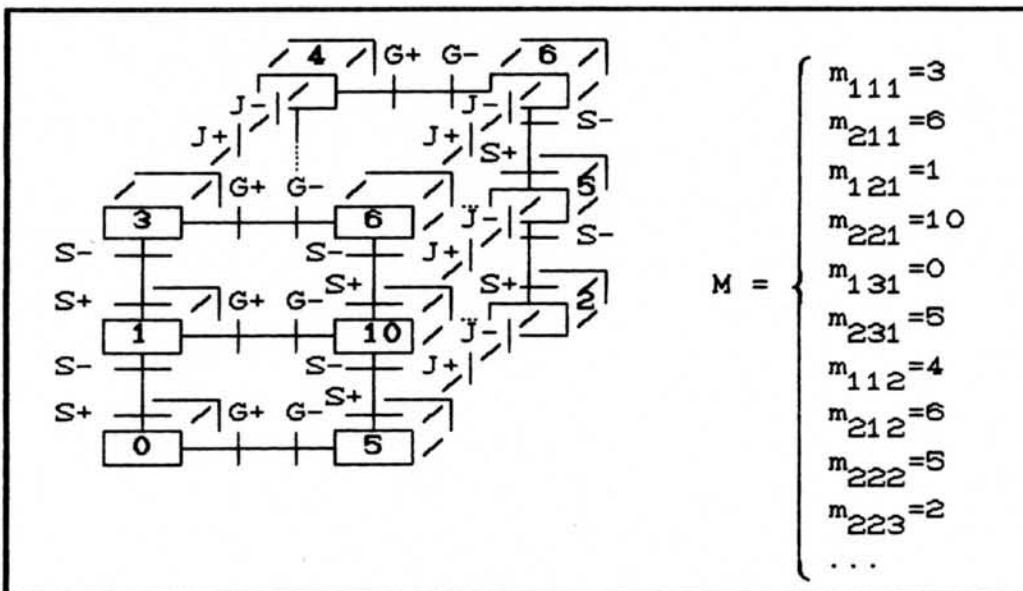


Figura 10.11 - Relacionamentos em uma matriz algébrica tridimensional.

Assim, visto que é possível gerar elementos algébricos e estabelecer os relacionamentos que estes elementos podem apresentar entre si, respeitando-se as

respectivas inerências algébricas, pode-se aplicar seqüências de ações matemáticas (regras algébricas) para operar os objetos de forma a obter uma solução para um dado problema. Como exemplos de regras podem ser citadas algumas propriedades algébricas para a adição:

- |   |  |
|---|--|
| 1 | → <u>Regra associativa:</u><br>$a + (b + c) = (a + b) + c$ |
| 2 | → <u>Regra comutativa:</u><br>$a + b = b + a$              |
| 3 | → <u>Regra do elemento neutro:</u><br>$a + 0 = a$          |
| 4 | → <u>Regra do elemento inverso:</u><br>$a + (-a) = 0$      |

Portanto, utilizando-se a linguagem de ações pode ser possível resolver, por exemplo, um sistema de equações algébricas tridimensionais, com seus elementos representados também na forma 3-D.

Os estudos mais aprofundados a respeito dos conceitos preliminares apresentados acima, assim como as suas aplicações, podem ser alvo de outros trabalhos de pesquisa, visto que fogem ao escopo deste documento, que visa lançar as sementes para a utilização de um modelamento de objetos funcionais em diferentes áreas de pesquisa. Portanto, um pesquisador da área de matemática computacional é o profissional indicado para aprofundar os estudos relativos às idéias iniciais que foram apresentadas nesta seção.

## 11. CONCLUSÕES

Ao longo deste trabalho procurou-se estabelecer mecanismos para a representação computacional na forma tridimensional de *objetos funcionais*. Tais elementos foram concebidos geometricamente como um agrupamento de volumes conectados uns aos outros por meio de operações de acoplamento, e existencialmente como entes que apresentam um conjunto de atributos ou *características funcionais*, sendo suas ocorrências representativas de determinados agrupamentos existenciais que foram conceitualmente denominadas *classes de objetos funcionais*. Além disso, foram definidos mecanismos, utilizando-se uma linguagem computacional apropriada, que permitem colocar os objetos funcionais dentro de uma *realidade funcional* (*universo real*), e estabelecer as *conseqüências funcionais* produzidas por uma ação desta natureza. Note-se que essas conseqüências são produzidas porque os objetos podem apresentar *relacionamentos funcionais* entre si, e ao ser introduzido um novo objeto dentro de uma realidade espacial, tais relacionamentos devem ser considerados e as conseqüentes *reações em cadeia* devem ser ativadas.

Assim, a geração de volumes pode ser realizada por meio de operações de acoplamento (*geminção, justaposição e superposição*), a partir de uma situação inicial, onde existe um volume inicial que deve ser gerado utilizando-se os mecanismos  $VP(Cx, Cy, Cz, T, C)$  e  $VI(NV)$ . Esta geração pode ser feita, utilizando o sistema principal SMOF.EXE, tanto na forma interativa (passo a passo) ou pela utilização dos mecanismos de geração de volumes, da *linguagem de ações*:  $V(NV, RG)$  e  $VR(NV, RG, NR)$ . No segundo caso, os comandos de geração ( $VP, VI, V$  e  $VR$ ) deverão ser escritos em um editor de textos na seqüência apropriada de geração, conforme mostrado nos exemplos apresentados

durante o trabalho, e gravados no disco com a extensão do nome do arquivo igual a .REG (Exemplo: JANELA.REG). Também, um ícone correspondente ao arquivo de ações deverá ser criado, de forma a permitir a ativação da seqüência de ações (Ex: JANELA.ICO). Alterações poderão ser realizadas utilizando-se os comandos da linguagem de ações apresentados no capítulo 5.

Note-se que a geração e manipulação (alterações) de objetos (volumes + atributos + relacionamentos funcionais) serão utilizados de acordo com os princípios de geração de uma *shape grammar*, como sugerido na introdução deste trabalho. Assim, pode-se fazer as seguintes comparações entre as idéias de *shape grammars* e os resultados obtidos neste trabalho (SMOF):

$\xrightarrow{1}$  *Shape grammars*: A geração de objetos é realizada a partir de uma forma (*shape*) inicial bidimensional.

SMOF: A geração de objetos é realizada a partir de um volume inicial, utilizando-se o mecanismo VI(NV), com as características geométricas definidas pelo mecanismo VP(Cx, Cy, Cz, T, C).

$\xrightarrow{2}$  *Shape grammars*: A geração de novas formas é realizada utilizando-se regras de uma gramática que são aplicadas a determinados rótulos dos elementos já existentes.

SMOF: A geração de novos volumes é realizada utilizando-se os mecanismos para geração de volumes V(NV, RG) e VR(NV, RG, NR). O parâmetro RG, nesse caso é o responsável pela aplicação das regras, onde os vértices das faces dos volumes são vistos como rótulos. A aplicação das regras pode ser controlada utilizando-se os mecanismos auxiliares da linguagem de ações.

$\xrightarrow{3}$  *Shape grammars*: Uma gramática de formas pode

ser utilizada para explicar e descrever uma determinada associação de objetos que possuem características comuns, ou para desenvolver novos objetos e testar as regras em que eles se fundamentam.

*SMOF*: Utilizando-se o modo interativo do sistema (mecanismos de geração e manipulação de volumes), pode-se desenvolver novos objetos. Um programa escrito utilizando-se a linguagem de ações pode ser utilizado para gerar e testar as regras em que um determinado objeto se fundamenta.

<sup>4</sup> → *Shape grammars*: São sistemas fundamentados em regras que, basicamente, consistem de um estágio inicial, um conjunto finito de elementos de um vocabulário, um conjunto de regras de produção (SE-ENTÃO) que são aplicadas aos elementos, e uma condição terminal.

*SMOF*: É um sistema fundamentado em regras de geração e manipulação (aplicáveis por meio de uma ou mais seqüências de ações, utilizando-se os mecanismos de uma linguagem de ações) que, basicamente, consiste de um estágio inicial E0 (onde nenhuma operação foi executada), um conjunto finito de elementos de um vocabulário (ativados por meio de ícones e gerados utilizando-se a linguagem de ações), um conjunto de regras de produção (reguladas pelo mecanismo de controle definido no capítulo 8 deste trabalho), e uma condição terminal que pode ser definida utilizando-se uma seqüência de ações para esta finalidade.

Dessa forma, pode-se concluir que as regras de *shape grammars* foram inseridas no contexto do trabalho sob a forma de seqüências de ações a serem programadas utilizando-se a linguagem de ações. Note-se, ainda, que todas as tarefas que são realizadas utilizando-se *shape grammars* podem ser realizadas pelo sistema *SMOF*, desde que um programa (que pode ser concebido em diversos módulos ativáveis pelos ícones) seja escrito utilizando-se a

*linguagem de ações.*

Outra questão que foi apresentada no início deste documento, foi a possibilidade de incorporação de *atributos especiais* (tais como *peso, temperatura, resistência a esforços físicos etc.*). Esta necessidade foi suprida no capítulo 9, onde foram criados mecanismos para a geração dos *atributos funcionais* de objetos. Nesse capítulo, criou-se uma forma de representação em disco (na forma de *relações*) de *classes* de objetos, e de suas *ocorrências* ou *entidades*. Além disso, foram estabelecidos mecanismos para a incorporação dos *relacionamentos funcionais* (*vitais*) entre os objetos tridimensionais modelados. Dessa forma, um objeto ao ser criado pode ter, também, suas *características funcionais* (*vitais*) geradas no disco e, principalmente, os seus *relacionamentos* com outros objetos existentes no universo de trabalho. Portanto, posteriormente, se for necessário determinar a existência de um determinado relacionamento entre dois objetos, este estará registrado em um arquivo específico para esta finalidade. Então, utilizando-se os mecanismos para leitura de relacionamentos, este poderá ser obtido para utilização pelos demais mecanismos auxiliares da *linguagem de ações*. Da mesma forma, as *relações* geradas poderão ser utilizadas para a formação de *relatórios*.

O capítulo 10 apresenta algumas sugestões de utilização dos conceitos apresentados neste documento, devendo merecer um tratamento mais aprofundado (em trabalhos futuros), pois o seu detalhamento foge ao escopo deste trabalho, que visa somente apresentar uma base introdutória à geração de objetos funcionais, fundamentada nas idéias de geração utilizando-se *shape grammars*. Portanto, o assunto não está esgotado e ainda há muito o que fazer, principalmente quanto a criação de aplicativos específicos para cada área do conhecimento, principalmente

quanto a sua utilização na área de arquitetura, de acordo com a idéia de aplicação original do modelo, conforme sugerido por Turkienicz em /TUR 90/.

Uma restrição quanto à utilização do SMOF, refere-se à forma cúbica adotada como elemento básico de geração, visto que elementos ou detalhes curvos não podem ser representados. Relativamente a essa restrição, deve-se evidenciar que o sistema não foi concebido para trabalhar com detalhes de projeto, mas sim, para manipular formas em um nível que permite a concepção volumétrica (volumetria) de um projeto, de forma a permitir maior liberdade de criação (o detalhamento seria uma fase posterior).

Finalmente, deve-se mencionar que o SMOF apresenta as seguintes vantagens em relação a um sistema convencional de CAD (sistema de edição gráfica limitado a uma descrição geométrica dos objetos):

- 1 → Proporciona a geração de objetos de forma mais atraente (grande interatividade com o usuário projetista).
- 2 → Torna mais ágil a tarefa de projeto, visto que o projetista pode dispor de elementos já prontos.
- 3 → Evita preocupações a nível de detalhamento, como acontece nos projetos convencionais em que detalhes de desenho devem ser considerados.
- 4 → É muito mais fácil gerar objetos utilizando-se um conjunto de elementos já prontos do que ter de construir cada um dos elementos durante o processo de projeto /TUR 90/.

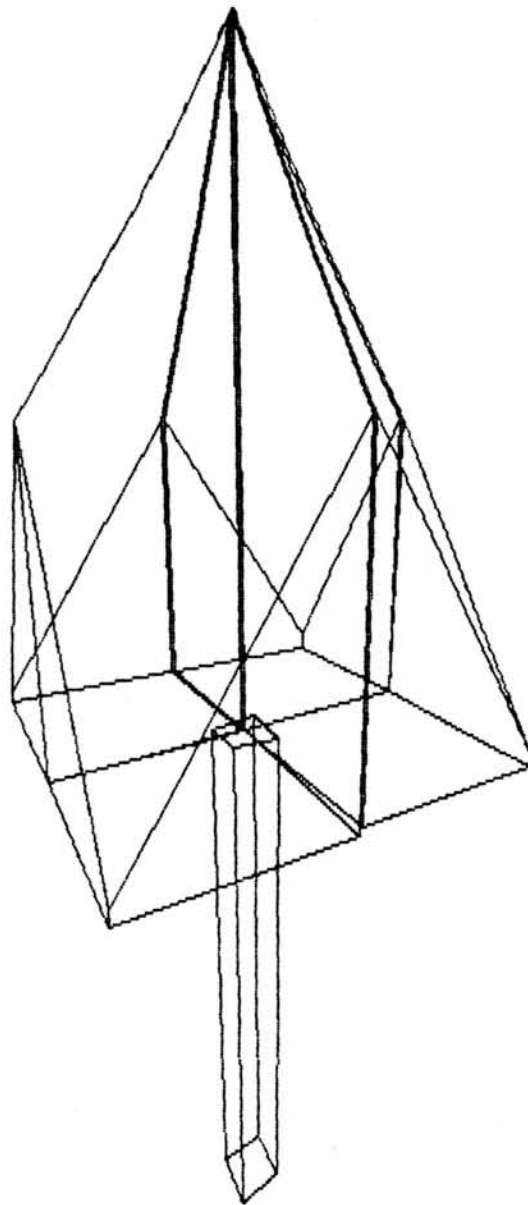


**ANEXOS**

**Exemplos de objetos gerados pelo SMOF**

ANEXO 1

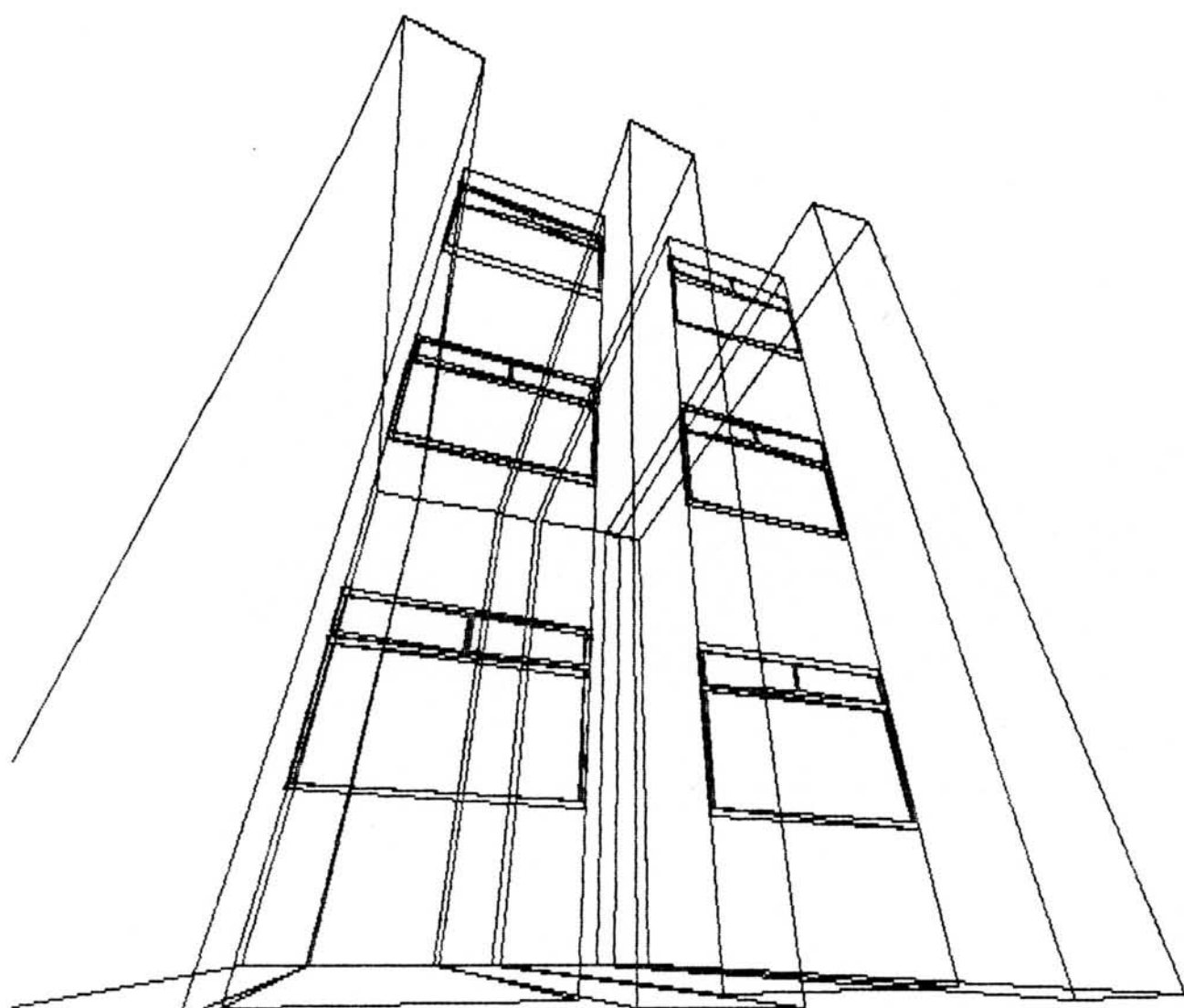
→ Arvore:



**ANEXO 2**

---

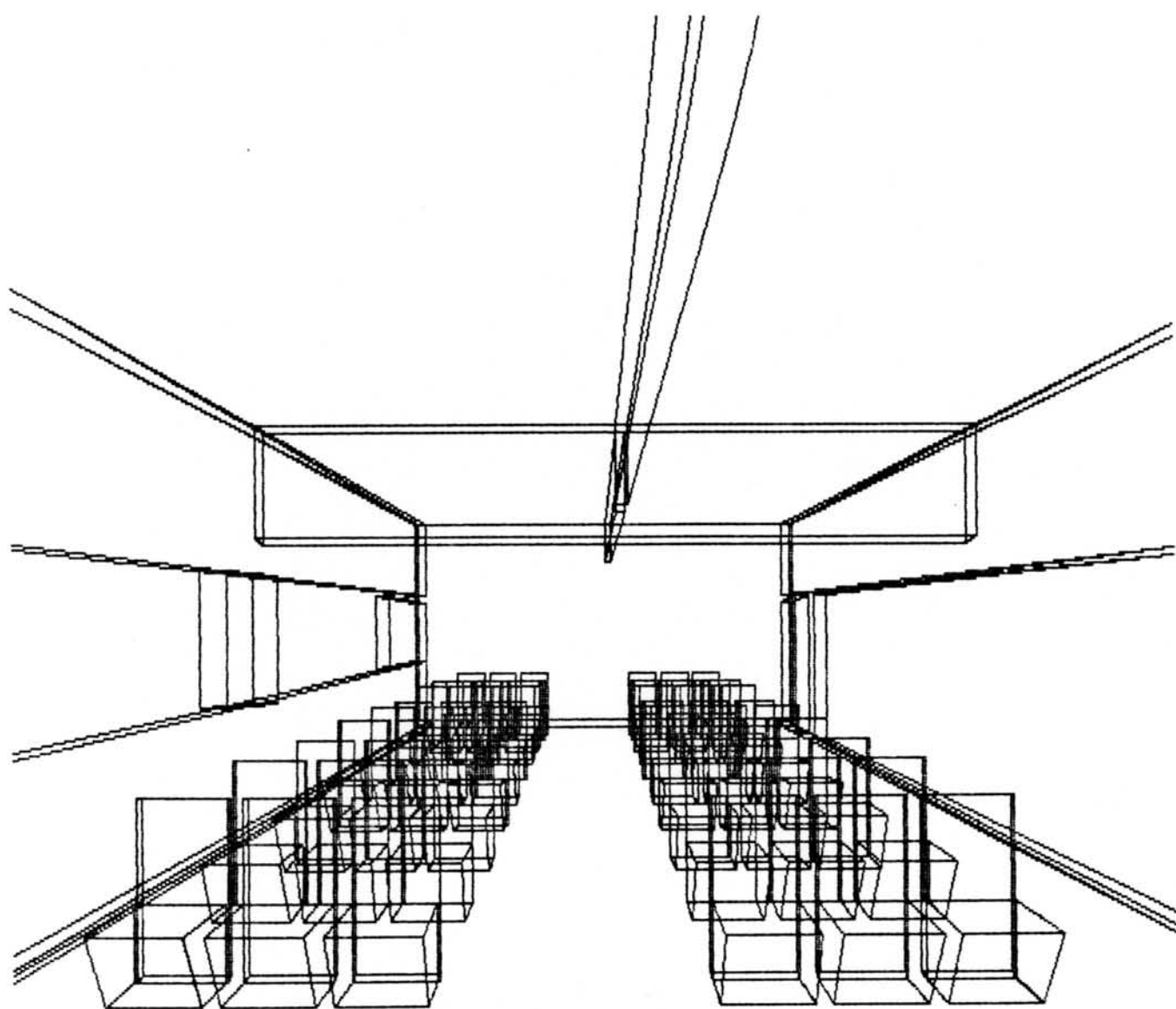
→ Edifício / Janelas:



**ANEXO 3**

---

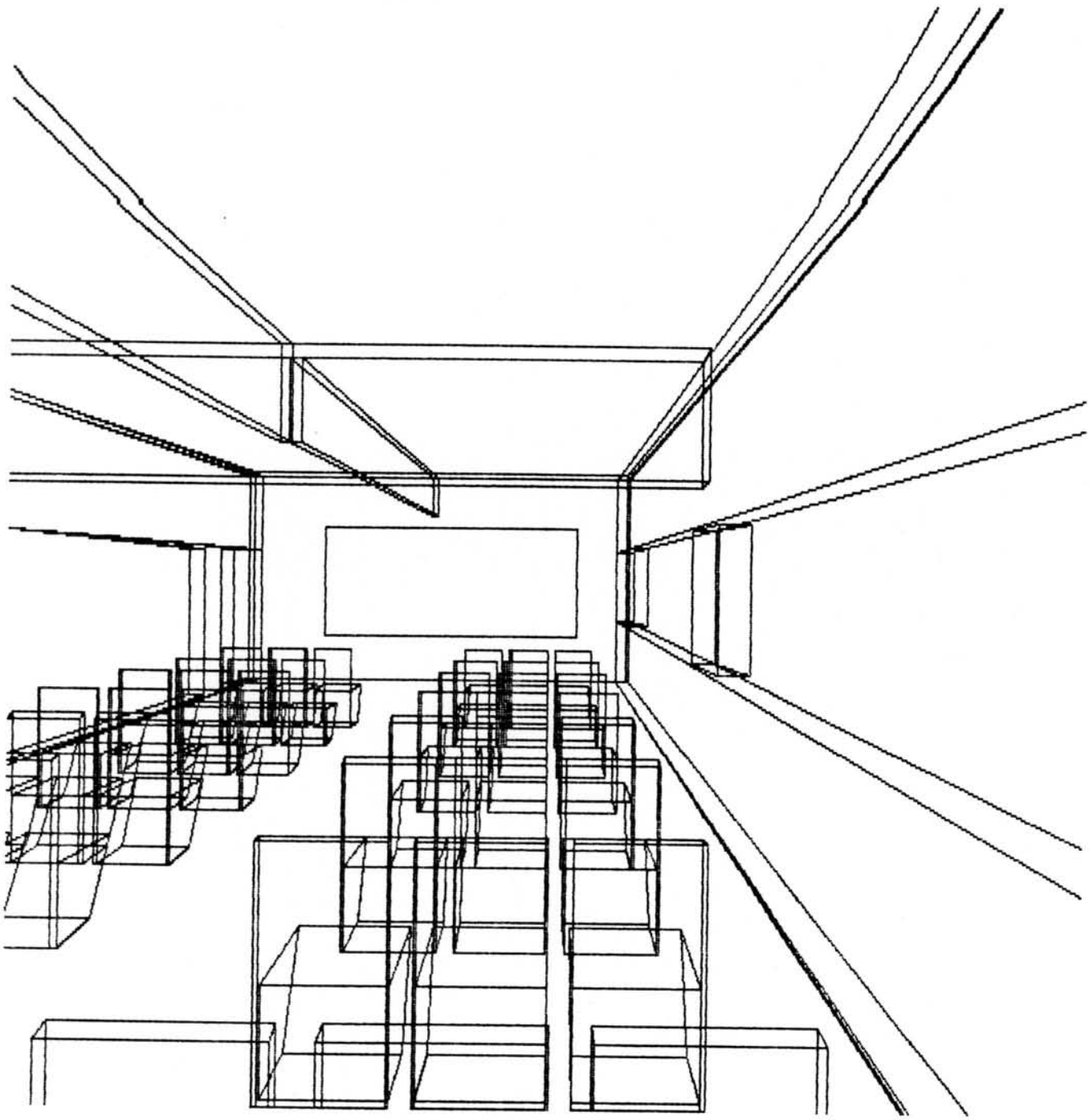
→ *Sala de aula (1):*



**ANEXO 4**

---

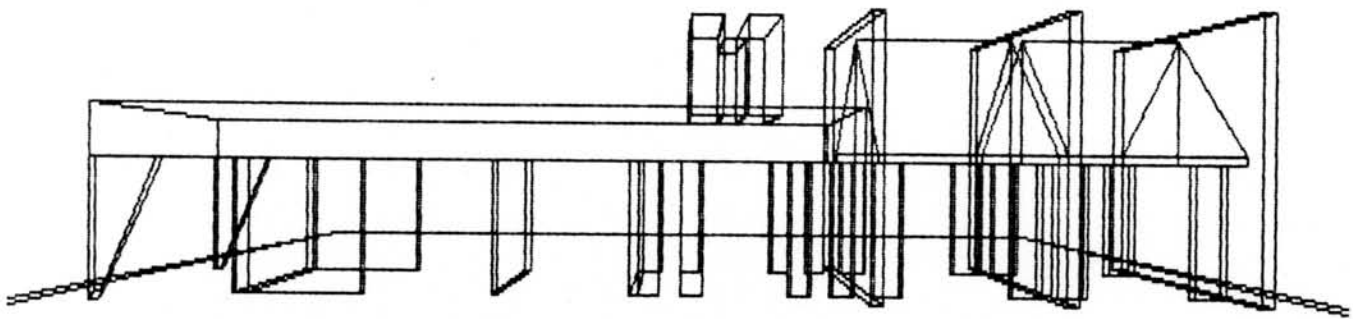
→ *Sala de aula (2):*



**ANEXO 5**

---

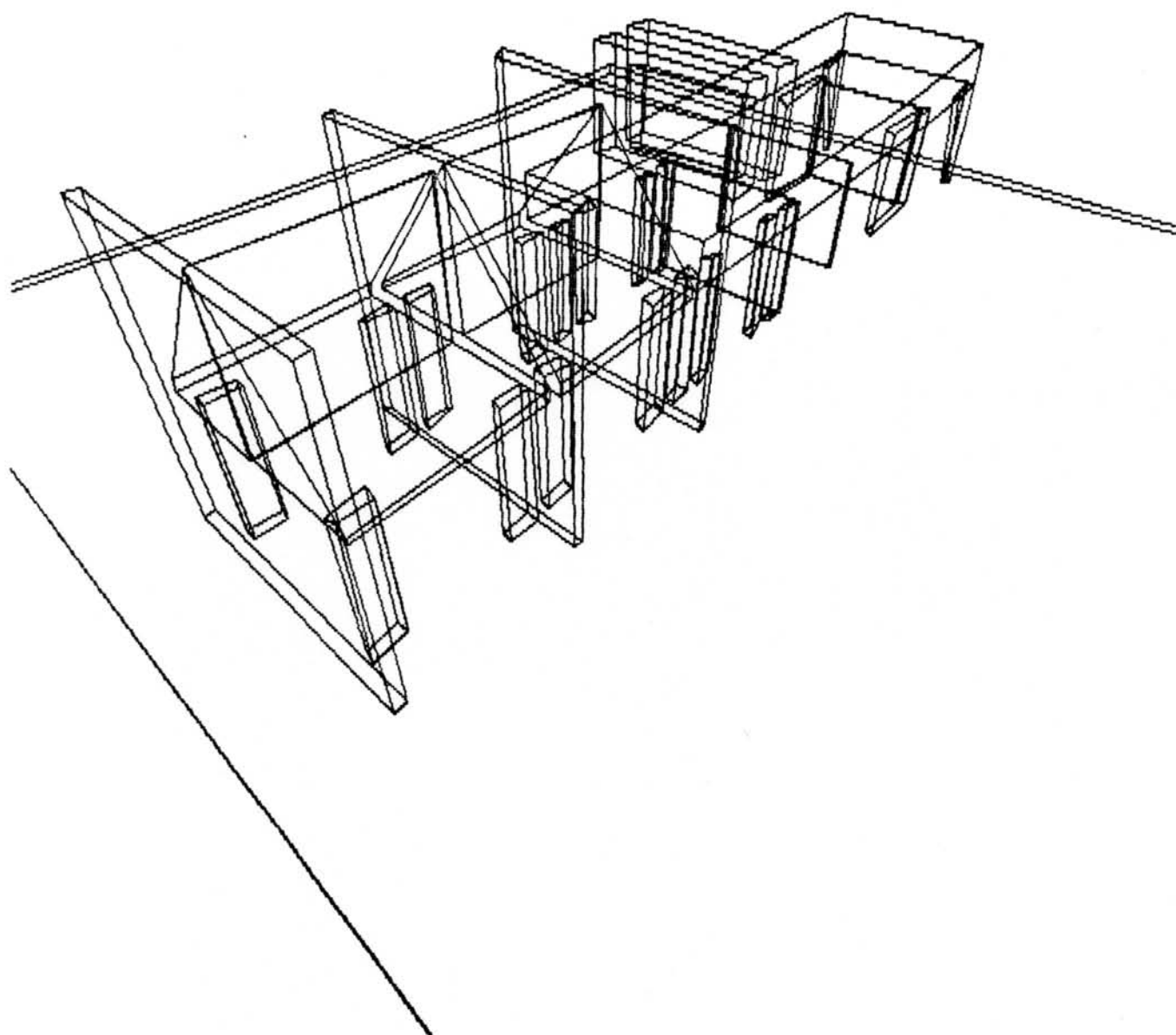
→ Casa (1):



**ANEXO 6**

---

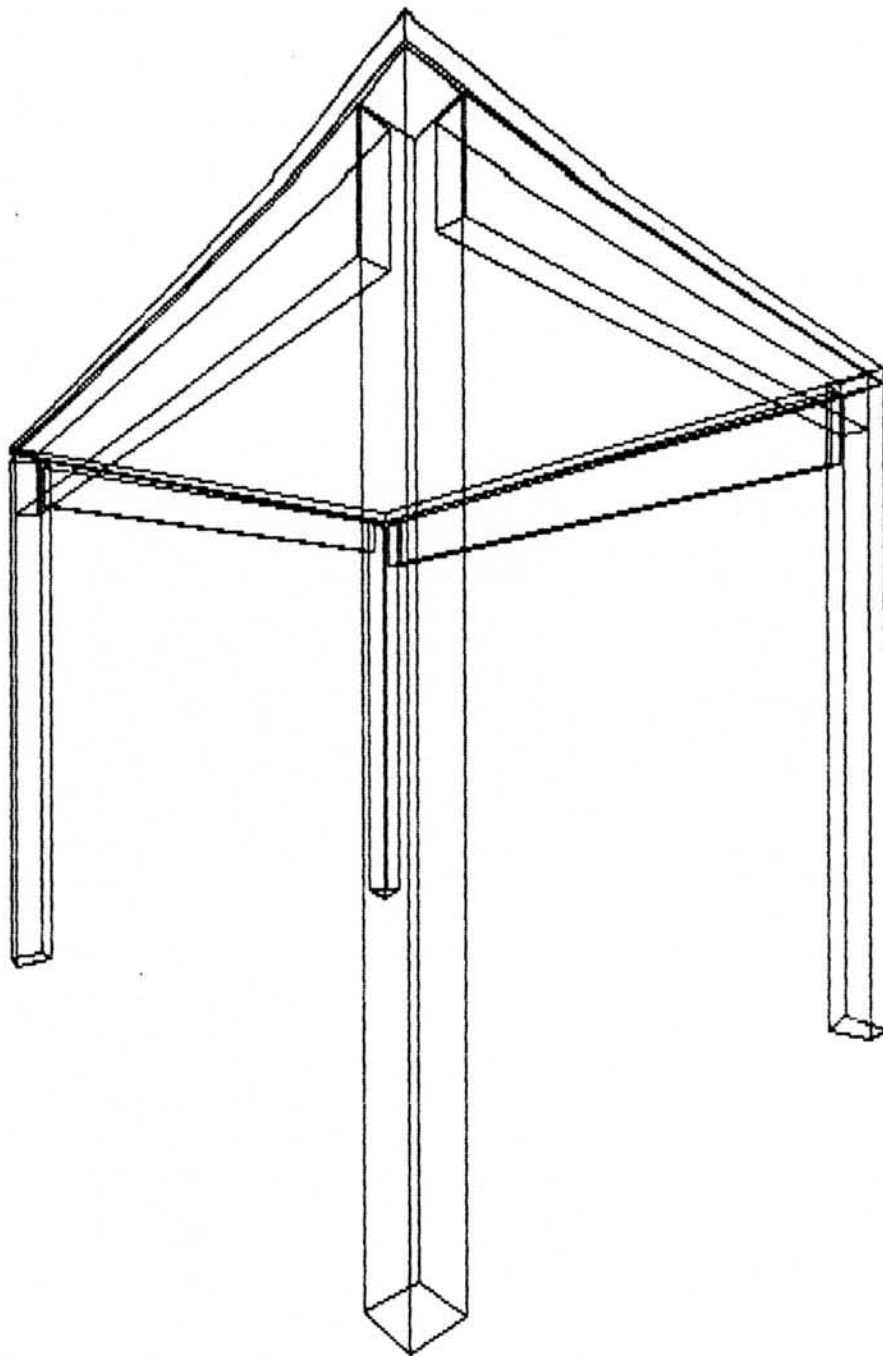
→ Casa (2):



**ANEXO 7**

---

→ *Laje inicial:*

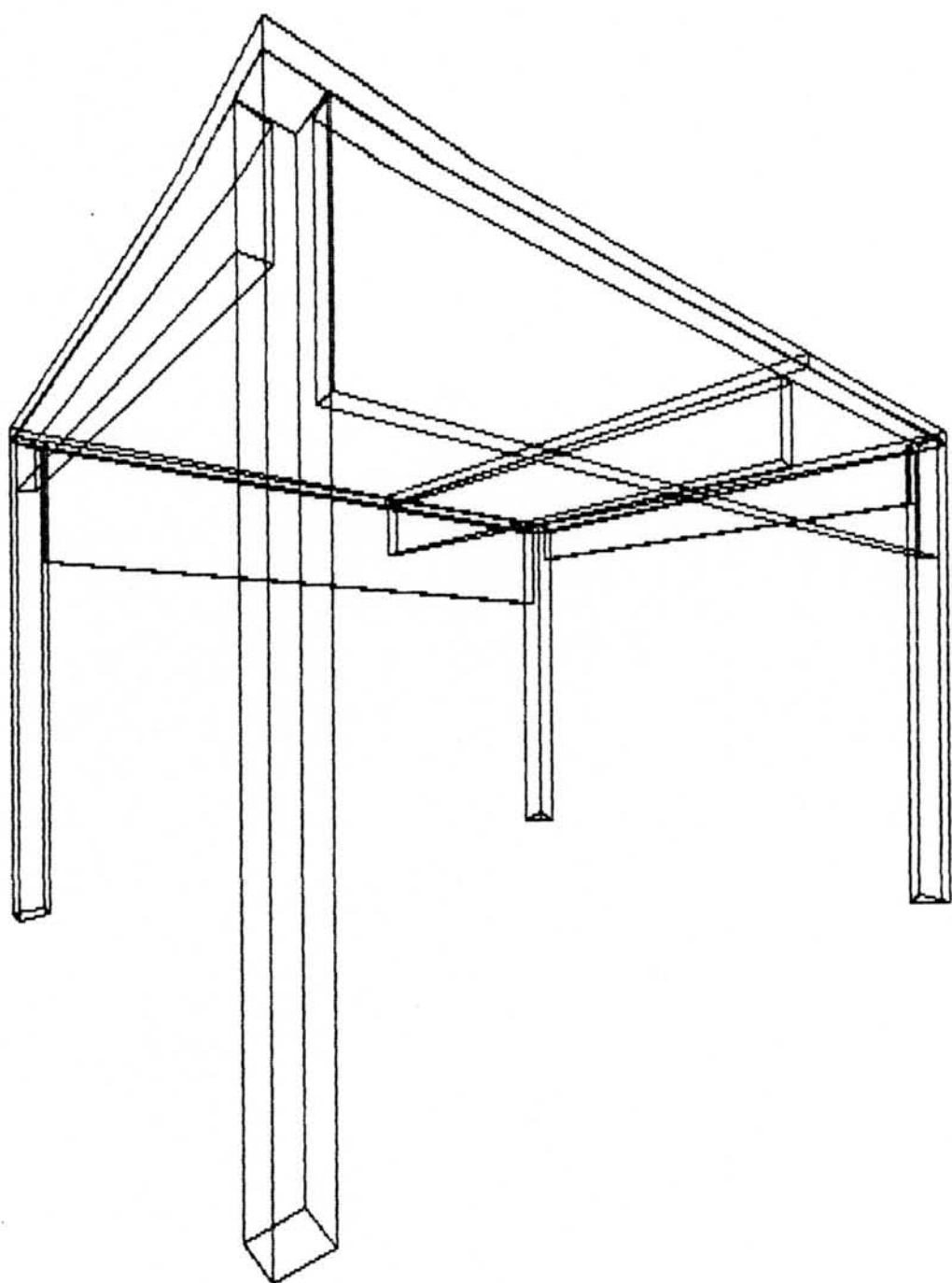




**ANEXO 8**

---

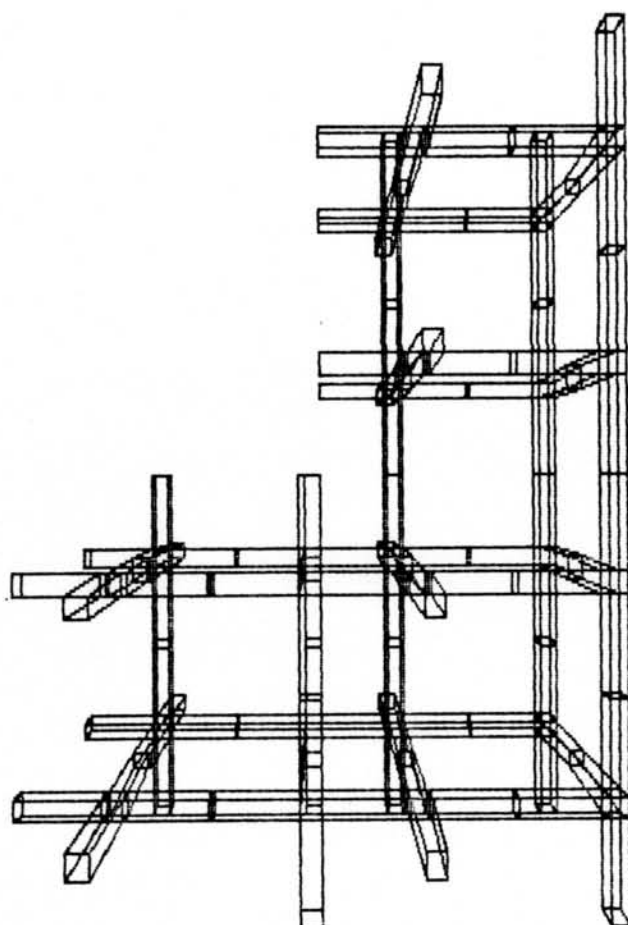
→ Laje / geminação positiva:



**ANEXO 9**

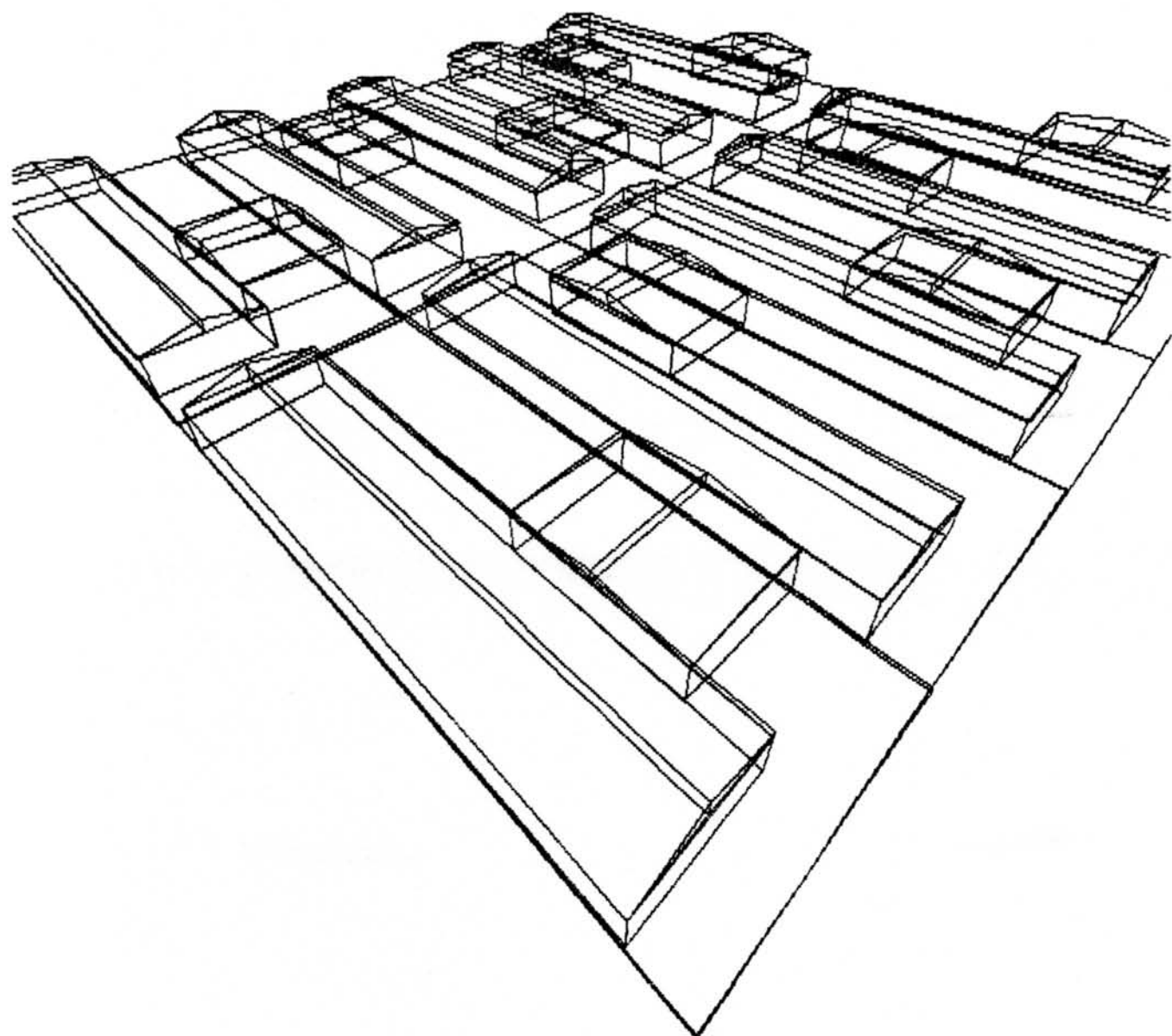
---

→ *Estrutura espacial:*



**ANEXO 10**

→ *Loteamento habitacional:*



## BIBLIOGRAFIA

- /ALV 87/ ALVES, Manoel Rodrigues & RUANO, Miguel. Towards meaningful computational descriptions of architectural form. Massachusetts, Massachusetts Institute of Technology, May 1987.
- /AND 81/ ANDRADE, Mônica Raposo. Manual de projetos de habitação popular. Recife, Secretaria de Habitação do Estado de Pernambuco, 1981.
- /BJO 89/ BJORK, B-G. Basic structure of a proposed building product model. Computer-Aided Design, Surrey, 21(2):71-78, May 1989.
- /COY 86/ COYNE, R. D. & GERO, J. S. Semantics and organization of knowledge in design. Design Computing, 1(1):68-89, Jan. 1986.
- /DAT 86/ DATE, G. J. An introduction to database systems. 4.ed. Reading, Addison-Wesley, 1986, v.1.
- /FER 78/ FERREIRA, Aurelio Buarque de Hollanda. Pequeno Dicionário Brasileiro da Língua Portuguesa. Rio de Janeiro, Civilização Brasileira, 1978.
- /FLE 86/ FLEMMING, Ulrich. The role of shape grammars in the analysis and creation of designs. Pittsburgh, Carnegie-Mellon University, Department of Architecture, 1986.
- /FOL 82/ FOLEY, J. D. & VAN DAM, A. Fundamentals of interactive computer graphics. Reading, Addison Wesley, 1982.

- /GER 85/ GERO, J. S. & COYNE, R. D. Logic programming a means of representing semantics in design languages. Planning and design, 12:351-369, May 1985.
- /GRA 89/ GRABOWSKI, Ing. H. & BENZ, Ing. T. Functional modelling in intelligent CAD-systems, Karlsruhe, University of Karlsruhe, 1989.
- /JAL 89/ JALOTE, Pankaj. Functional refinement and nested objects for object-oriented design. IEEE Transactions on Software Engineering, New York, 15(3): 264-270, May 1989.
- /JOR 84/ JOROFF, Michael L. & MOORE, James A. Case method teaching about design process management. JAE, 38/1, Fall, 1984, p. 14-17.
- /LAR 62/ LARROUSE Méthodique, Encyclopédie. Rio de Janeiro, Delta, 1962.
- /MAH 83/ MAHFUZ, Edson da Cunha. An investigation into the nature of relationship between the parts and the whole in architectural composition. Pennsylvania, 1983.
- /MAK 87/ MAKKUNI, Ranjit. A gestural representation of the process of composing chinese temples. IEEE Computer Graphics & Applications, Los Alamos, 7(12):45-61, Dec. 1987.
- /MAR 85/ MARTIN, James & MACLURE, Carma. Diagramming techniques for analysts and programmers. New Jersey, Prentice-Hall, 1985.

- /MOR 85/ MORTENSON, Michael E. Geometric Modeling. New York, John Wiley & Sons, 1985.
- /PAT 82/ PATNAIK, L. M. & RAMESH N. Implementation of an interactive relational graphics database. Computer & Graphics, New York, 6(3):93-96, 1982.
- /ROC 84/ ROCHA, Aderson Moreira da. Curso prático de concreto armado. São Paulo, Nobel. 1984, V.1.
- /ROG 85/ ROGERS, David F. Procedural elements for computer graphics. New York, McGraw-Hill, 1985.
- /ROO 87/ ROONEY, Joe. Principles of computer-aided design. Pitman/Open University, 1987.
- /SMT 77/ SMITH, John Miles and SMITH, Diane C. P. Database abstractions: agregation. Communications of the ACM, New York, 20(6):405-413, Jun. 1977.
- /SMI 89/ SMITHERS, T. AI-based design versus geometry-based design or why design cannot be supported by geometry alone. Computer Aided Design, Surrey, 21(3):141-150, Apr. 1989.
- /SPO 84/ SPOONER, David L. Database support for interactive computer graphics. In: ACM-SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, Boston, June, 18-21, 1984. Proceedings, New York, ACM, 1984. Publicado em: SIGMOD RECORD, New York, 14(2):90-99, June 1984.

- /STA 83/ STAA, Arndt von. Engenharia de Programas.  
Rio de Janeiro, LTC, 1983.
- /STI 85/ STINY, George Nicholas. Computing with form and  
meaning in architecture. JAE, 39/1, Fall,  
1985, p. 7-19.
- /WIE 86/ WIEDERHOLD, Gio. Views, objects, and databases.  
Computer, Los Alamitos, 19(12):37-44, Dec.  
1986.

#### OUTRAS FONTES

- /TUR 90/ TURKIENICZ, Benamy. "LEGO eletrônico". Artigo  
aceito para publicação na Revista AU: Arquite-  
tura e Urbanismo, em julho de 1990.

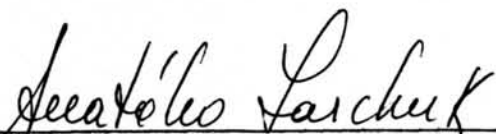
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Um sistema de modelagem  
de objetos funcionais

Dissertação apresentada aos Srs.



\_\_\_\_\_  
Profa. Carla M. Dal Sasso Freitas



\_\_\_\_\_  
Prof. Anatolio Laschuk



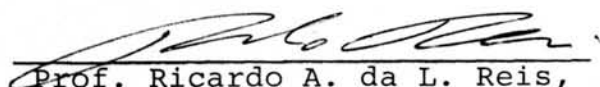
\_\_\_\_\_  
Prof. Dr. Clesio Saraiva dos Santos

Visto e permitida a impressão.  
Porto Alegre, ..13../.07.../.90.



\_\_\_\_\_  
Prof. Anatolio Laschuk

Orientador



\_\_\_\_\_  
Prof. Ricardo A. da L. Reis,

Coordenador do Curso de Pós-

-Graduação em Ciência

da Computação.