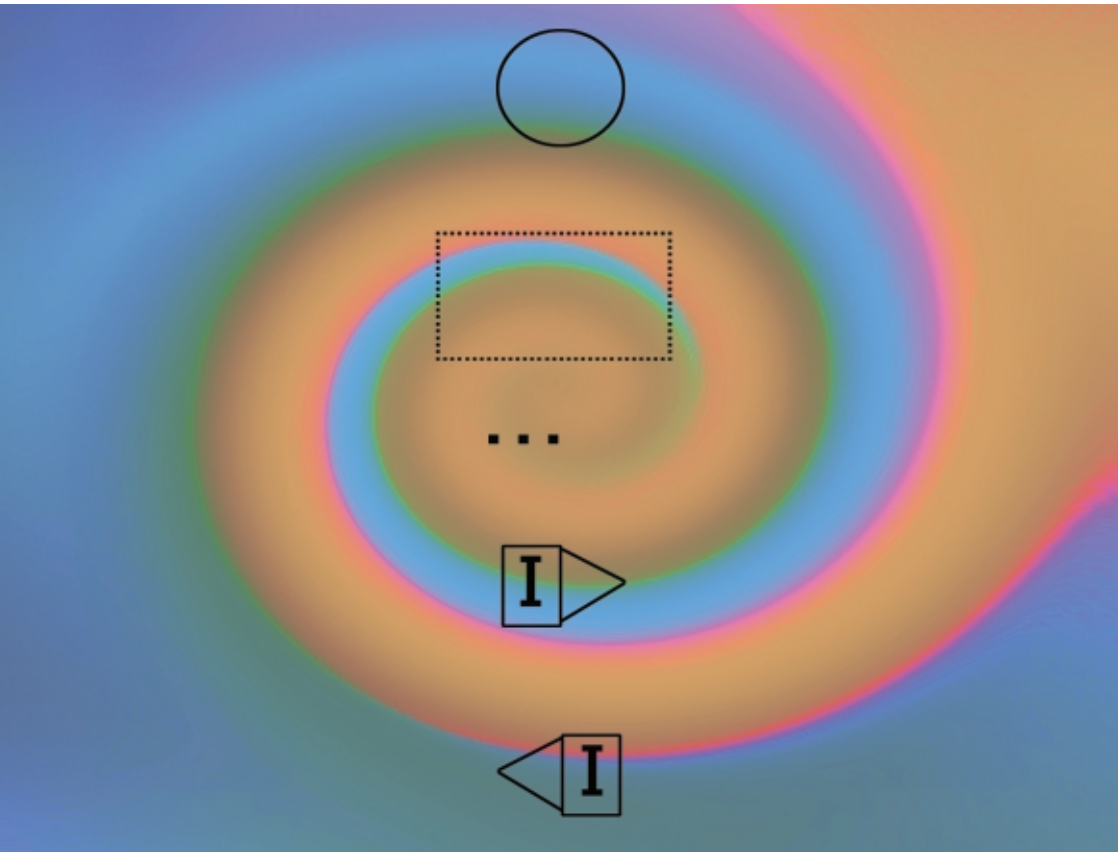
Ministério da Agricultura, Pecuária e Abastecimento

Relatório Técnico 14

ISSN 1517-0330
Dezembro, 2001

Modelo para Programação Visual de Matrizes (MVM): Uma Nova Abordagem para Visualização, Manipulação e Programação de Algoritmos Matriciais



Embrapa

ISSN 1517-0330
Dezembro, 2001

Empresa Brasileira de Pesquisa Agropecuária
Centro Nacional de Pesquisa Tecnológica em Informática para a Agricultura
Ministério da Agricultura, Pecuária e Abastecimento

Relatório Técnico 14

Modelo para Programação Visual de Matrizes (MVM): Uma Nova Abordagem para Visualização, Manipulação e Programação de Algoritmos Matriciais

Silvio Roberto Medeiros Evangelista

Campinas, SP
2001

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Av. Dr. André Tosello s/nº
Cidade Universitária "Zeferino Vaz" – Barão Geraldo
Caixa Postal 6041
13083-970 – Campinas, SP
Telefone/Fax: (19) 3789-5743
URL: http://www.cnptia.embrapa.br
Email: sac@cnptia.embrapa.br

Supervisor editorial: *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel Fugisawa Souza*
Capa: *Intermídia Publicações Científicas*
Editoração eletrônica: *Intermídia Publicações Científicas*

1ª edição

Evangelista, Silvio Roberto Medeiros.
Modelo para programação visual de matrizes (MVM): uma nova abordagem para visualização, manipulação e programação de algoritmos matriciais / Silvio Roberto Medeiros Evangelista. — Campinas : Embrapa Informática, 2001.
57 p. : il. — (Relatório técnico / Embrapa Informática Agropecuária ; 14)

ISSN 1517-0330

1. Programação visual. 2. Linguagens de programação visual. 3. Programação de matrizes. 4. Programação por demonstração. 5. Fluxo de dados. I. Título. II. Série.

CDD – 005.118 (21.ed.)

Sumário

Modelo para Programação Visual de Matrizes (MVM): Uma Nova Abordagem para Visualização, Manipulação e Programação de Algoritmos Matriciais

Silvio Roberto Medeiros Evangelista[1]

Resumo

Para muitos usuários, a programação visual é uma alternativa atrativa às linguagens de programação textuais. Uma das razões para esta atração é que a representação visual de um problema está muito mais próxima com a forma pela qual a solução é obtida ou entendida se comparada à representação textual. Este trabalho apresenta um modelo para a programação visual de matrizes baseado nos paradigmas de fluxo de dados e planilhas eletrônicas. O fluxo de dados e a planilha formam a base semântica da linguagem, enquanto as representações gráficas do grafo direcionado e de uma planilha fundamentam sua base sintática. Este modelo consiste em um conjunto de diagramas bidimensionais e de regras de transformação. Os processos são implementados como redes de fluxo de dados e os dados são representados por planilhas. As planilhas podem ser vistas como variáveis do tipo matriz que armazenam dados bidimensionais, ou como funções, que recebem e produzem valores utilizados por outros processos. Neste caso, as planilhas são programadas seguindo o paradigma de programação por demonstrações que incorpora um poderoso construtor de iteração, reduzindo significativamente a utilização de recursões e repetições. O modelo proposto pode ser utilizado em diversos domínios de aplicação, principalmente para simplificar a construção de modelos matemáticos de simulação e análise estatística.

Termos para indexação: Linguagem de programação visual; Programação por demonstração; Programação de matriz; Fluxo de dados; Fluxo de controle; Planilhas eletrônicas.

---

[1] Doutorando em Computação Aplicada, Pesquisador da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo – 13083-970 – Campinas, SP. (E-mail: silvio@cnptia.embrapa.br)

A Visual Programming Model for Matrix: A New Approach to Visualization, Manipulation and Programming of Matrix Algorithms

Abstract

For many users, visual programming is an attractive alternative to textual programming languages. One of the reasons for this attraction is that the visual representation of a solution to a problem is closer with the way the solution is conceived or understood if compared to text representation. This paper presents a model for matrix visual programming based on data flow paradigm and spreadsheets. The data-flow and the spreadsheet are the semantical base of the language, meanwhile the graphical representation of direct graphs and spreadsheets are the sintatical fundaments. This model consists of a set of bi-dimensional diagrams and transformation rules in which processes are implemented as data flow networks and the data are represented by spreadsheets. The spreadsheets can be seen as variables of matrix type, which store bi-dimensional data, or as functions, which receive and produce values, to be used by other processes. In this case, the spreadsheets are programmed following the by-example paradigm, which embeds a powerful iterator constructor, greatly reducing the usage of recursions and repetitions. The proposed model can be used in various application domains, mainly to simplify the building of mathematical simulation models and statistical analysis.

Index terms: Visual programming language; Programming by-examples; Matrix programming; Data-flow; Control-flow; Spreadsheet.

1. Introdução

Para muitos usuários, a programação visual é uma alternativa atrativa às linguagens de programação usuais. Uma das razões para esta atração é que a representação visual de um problema está muito mais próxima com a forma pela qual a solução é obtida ou entendida se comparada à representação textual. As linguagens de programação textuais possuem notoriedade em relação à dificuldade que muitas pessoas têm em usá-las. Na verdade, a facilidade que as linguagens textuais oferecem para descrever um determinado problema ou algoritmo está mais relacionada com a maneira pela qual os computadores operam e não ao processo cognitivo de programação (Brown & Kimura, 1994).

Diferentes tipos de gráficos e diagramas são utilizados pelos projetistas de software como resposta à lacuna existente entre os processos cognitivos e os processos computacionais. Assim, diagramas mostrando o inter-relacionamento entre sub-rotinas e as rotinas chamadoras tornaram-se comuns. A documentação de qualquer programa escrito com uma linguagem de programação modular é considerada incompleta sem um diagrama que mostre o inter-relacionamento entre os módulos do sistema. Da mesma maneira, as documentações de sistemas orientados a objetos inevitavelmente incluem diagramas mostrando as relações existentes entre classes e subclasses.

Notações bidimensionais e linguagens têm sido utilizadas livremente na matemática e na lógica simbólica. Alguns exemplos destas notações são os grafos e as árvores empregadas para definir e/ou ilustrar os relacionamentos entre objetos. Um outro exemplo bem conhecido é o diagrama de Venn, que foi estendido por Harel (1988) para permitir a representação visual dos relacionamentos.

A idéia de transformar diagramas em programas computacionais não é nova. Pesquisadores do M.I.T (*Massachusetts Institute of Technology*) exploraram a programação interativa por meio de *flowcharts* no inicio da década de 60 (Marriott & Meyer, 1998). Todavia, somente após o advento dos computadores pessoais e a queda significativa do custo do hardware gráfico é que a programação visual ganhou força. Desde então, muitas linguagens visuais têm sido propostas, desenvolvidas ou comercializadas

(Whitley, 2000; Leopold & Ambler, 1996; Baroth & Hartsough, 1995; Smith, 1994; Kimura, 1993; Yeung, 1988; Jaffar & Lassez, 1987; Vose & williams, 1986; Matwin & Pietrzykowski, 1985; Glinert & Tanimoto, 1984; Gould & Finzer, 1984).

Neste trabalho é descrito um modelo para a programação visual de matrizes, denominado de MVM. A base semântica do modelo é híbrida e fundamentada no fluxo de dados e na planilha eletrônica. A fundamentação da base sintática, a parte visual do MVM, é o grafo direcionado e o relacionamento entre células de uma planilha. Este modelo consiste em um conjunto de diagramas no plano bidimensional e em um conjunto de regras de transformação. Neste sentido, os processos são implementados como redes de fluxo de dados e os dados são representados por planilhas. As planilhas são programadas seguindo o paradigma de programação por demonstração.

A motivação deste trabalho surgiu pela constatação de que não existem ferramentas de programação visual apropriadas para atender às necessidades das pesquisas na área de matemática aplicada e estatística. Além do mais, para este domínio de aplicação, as poucas ferramentas de programação visual existentes no mercado não são de fácil utilização por usuários sem treinamento em computação e só podem ser aplicadas para resolução de problemas simples.

Este trabalho inicia com uma breve definição para o termo programação visual (Seção 2) e uma descrição dos paradigmas utilizados para a fundamentação da sua semântica (Seção 3). A Seção 4 apresenta brevemente alguns sistemas que influenciaram a modelagem do MVM. Na Seção 5 é apresentado formalmente o MVM. Neste sentido, a Seção 5.1 descreve como uma planilha é utilizada para modelar o objeto matriz . A Seção 5.2 relata como o MVM deixa explícito o fluxo de dados que existe entre as várias células de uma matriz. A Seção 5.3 generaliza a estratégia de programação por exemplo para permitir que a planilha seja encarada como uma função. A Seção 5.4 descreve o processo de partição e redução de uma matriz. Finalmente, na Seção 5.5 alguns destes conceitos são aplicados a um exemplo numérico.

2. Definição de Linguagem de Programação Visual

Os processos computacionais são tradicionalmente especificados por cadeias de caracteres unidimensionais. A programação visual, em contraste, utiliza diagramas e ícones em pelo menos duas dimensões (Brown & Kimura, 1994). Mais formalmente, um conjunto de diagramas e ícones que correspondam às sentenças válidas em uma determinada linguagem de programação. Cada diagrama, por sua vez, é uma coleção de símbolos no espaço bi ou tridimensional. A determinação da validade de uma sentença e o seu significado dependem do relacionamento espacial entre estes símbolos. De acordo com Shu (1988b), o objetivo da programação visual é dotar programadores e usuários com o entendimento do que o programa pode fazer, como ele funciona, porque ele funciona e quais são seus efeitos.

Shu (1988a) relata que os objetos manipulados na programação visual não possuem uma representação visual intrínseca. Entre estes objetos, têm-se os tipos de dados tradicionais (arranjos, pilhas, listas, filas, etc.) e aplicações orientadas por tipos de dados. Neste último caso, pode-se citar as aplicações orientadas por formulários, documentos, banco de dados, etc. Naturalmente, para se produzir uma interface amigável, estes objetos precisam ser apresentados visualmente. Da mesma maneira, a linguagem precisa ser retratada graficamente. Em resumo, os construtores de um programa, as regras de combinação dos construtores e os dados do programa necessitam de notação visual.

3. Fundamentação das Linguagens Visuais

Um grande número de diferentes paradigmas tem sido explorados para a definição da base semântica na qual é fundamentada uma linguagem de programação visual. Uma das primeiras classificações para as linguagens visuais foi feita por Myers (1986), que classificou, seguindo um critério ortogonal, os sistemas de programação em oito categorias diferentes:

- Programação Visual ou não;
- Programação baseada em exemplos ou não; e
- Interpretativas ou compiladas.

Este sistema de classificação é muito geral e foi suplantado pelo sistema proposto por Burnett & Ambler (1994), no qual as linguagens visuais são categorizadas em dois grupos. O primeiro agrega as linguagens visuais que foram derivadas ou adaptadas dos paradigmas das linguagens textuais tradicionais. O segundo grupo para os paradigmas que possuem representação naturalmente visual. A Fig.1 ilustra os diferentes paradigmas pertencentes a estes dois grupos.

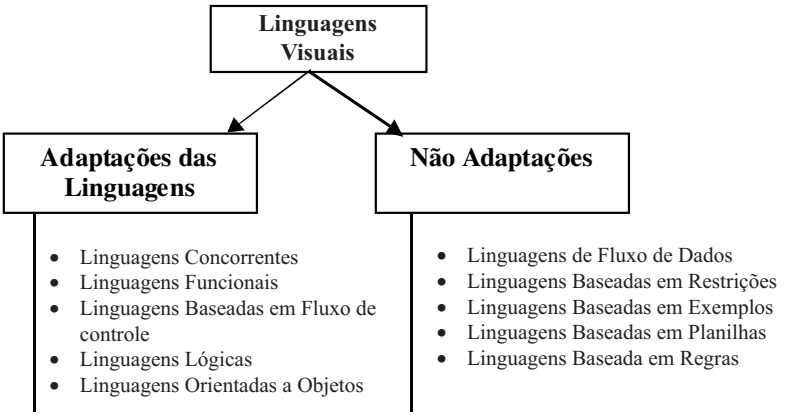


Fig. 1. Paradigmas para as linguagens visuais.

A primeira categoria inclui aquelas linguagens que não são intrinsecamente visuais, mas possuem uma notação visual imposta e podem ser classificadas como linguagens visualmente transformadas. As primeiras abordagens das linguagens para programação visual consistiam de editores visuais para as linguagens textuais tradicionais. Baseadas em fluxo de controle, estas linguagens utilizavam diagramas para representar os construtores da linguagem, como, por exemplo, os diagramas *flowchart* e *Nassi-Schneiderman* (Tripp, 1988). Mais tarde surgiram linguagens vi-

suais não vinculadas a qualquer notação textual anterior. Estas linguagens eram baseadas em paradigmas tradicionais, todavia não eram uma simples tradução de uma linguagem textual preexistente.

A segunda categoria engloba paradigmas de programação cuja representação de suas expressões possui representação visual natural. Um primeiro exemplo que pode ser citado como base semântica, na qual é fundamentada uma linguagem visual, é o paradigma de fluxo de dados. Em sistemas baseados em fluxos de dados, a seqüência em que as operações e funções são executadas não é especificada explicitamente. Ao invés disto, apenas a fonte dos dados de uma operação é especificada. Caso todos os dados de entrada de uma determinada operação estejam disponíveis, a operação ocorre. É bom salientar, contudo, que a maioria das linguagens de programação visuais baseadas em fluxo de dados utiliza algum construtor de fluxo de controle, segundo Hils (1992) verificou em um estudo.

Uma alternativa aos métodos baseados em fluxo é a abordagem apoiada em restrições[2]. Os sistemas desenvolvidos segundo esta abordagem permitem que o usuário especifique visualmente as propriedades invariantes dos objetos ou variáveis, bem como seus inter-relacionamentos (Graf & Neurohr, 1995). O sistema deve assinalar valores para as variáveis de maneira que todas as restrições sejam verdadeiras. Tais sistemas são, em alguns aspectos, análogos aos sistemas de programação lógica.

Existem linguagens visuais que são baseadas na programação por exemplo ou demonstração. Em tais sistemas, o usuário manipula amostras de dados ou representações visuais de suas estruturas para "ensaiar" o sistema em relação às operações que devem ser executadas. O sistema, então, emula as operações ensaiadas sobre um novo conjunto de dados.

Uma outra classe de linguagens de programação visual é fundamentada em formulários. Ambler e Burnett, que lideram as pesquisas neste paradigma, afirmam que a programação baseada em formulários pode ser pensada como uma generalização da programação em planilhas ele-

---

[2] Uma relação booleana, normalmente uma relação de igualdade ou desigualdade entre os valores de uma ou mais variáveis. Por exemplo, x>3 é uma restrição em x.

trônicas (Ambler & Burnett, 1989). A idéia básica é expandir visualmente os conceitos empregados em uma planilha para possibilitar a ampliação do domínio de aplicação. Uma definição para as classes de problemas que poderiam ser resolvidos com esta abordagem foi apresentada em Ambler (1987). Segundo os autores, qualquer problema que possa ser solucionado com a utilização de um lápis e folhas papel é um bom candidato para que esta abordagem seja aplicada. É importante observar que o sucesso das planilhas eletrônicas se deve aos métodos visuais utilizados para representar os relacionamentos entre os itens de dados (Amber & Burnett, 1989).

Finalmente, as linguagens visuais baseadas em regras possuem um conjunto de estados e regras que transformam o estado atual para algum estado futuro (Ambler et al., 1997). Cada regra possui uma pré-condição e uma especificação de transformação. Caso a pré-condição seja satisfeita, a especificação de transformação é executada. Na sua forma mais simples, assume-se que todas as regras são mutualmente exclusivas. Todavia, se o número de regras aumentar muito não se pode garantir que elas sejam independentes, o que requer um mecanismo de resolução mais complexo. Os sistemas fundamentados neste paradigma são particularmente efetivos para expressar controle sobre eventos que exigem algum tipo de reação.

Uma vez que a base semântica foi escolhida, a fundamentação sintática deve ser selecionada. Esta fundamentação sintática determina a representação real dos programas. Nos sistemas baseados em fluxo, a abordagem mais comum é o grafo direcionado (Brown & Kimura, 1994). Os nós desses grafos indicam operações ou células de dados, enquanto os arcos indicam fluxos de controle ou de dados. Algumas linguagens baseadas em fluxo, por exemplo, HI-Visual (Hirakawa, 1988), utilizam a justaposição de nós para indicar fluxo. Os nós podem ser representados como caixas, ícones ou qualquer outra forma. Os sistemas baseados em restrições normalmente empregam caixas e linhas como representação. Os sistemas baseados em formulários são uma extensão das planilhas eletrônicas, pois as suas representações visuais se assemelham às interfaces das planilhas eletrônicas. A representação visual e sintática dos sistemas fundamentados em exemplos ou demonstração são usualmente ligadas ao domínio de aplicação.

Após as bases semânticas e sintáticas estarem definidas, deve-se criar o conjunto básico de construtores computacionais para a linguagem. Estes construtores podem incluir representações para iterações, execuções seqüenciais, abstração de procedimento, recursões, checagem de tipo, etc.

4. Trabalhos Correlatos

Existem linguagens, como APL (Iverson & Falkoff, 1973) e MathLab (Kajler & Soiffer, 1995), que incorporam operadores para a construção e manipulação de matrizes. Estas linguagens são textuais e possuem uma notação densa, o que freqüentemente torna mais difícil o entendimento do programa. A estratégia adotada neste trabalho procura eliminar esta deficiência pela utilização de notação visual e manipulação direta de uma matriz.

A planilha eletrônica é possivelmente a forma mais conhecida para a manipulação direta de matrizes. Ela introduz o conceito de referência de células pelo apontamento direto das mesmas, além da replicação de fórmulas com a utilização do recurso de cópia. Todavia, ela não é adequada para a manipulação de submatrizes e, ademais, não possuem o conceito de abstração funcional.

A linguagem MPL (Yeung, 1988) é uma linguagem de manipulação de matrizes fundamentada nos paradigmas de restrições e lógica. Ela provê um conjunto de elementos para a notação gráfica de uma matriz, enquanto as cláusulas lógicas são informadas textualmente. Em MPL, uma matriz é formada por um retângulo fechado e um conjunto de sub-retângulos internos que delimitam várias submatrizes. O programador cria um programa (cláusulas Prolog) com um editor de texto modificado, no qual uma matriz pode ser manipulada como um texto. Conseqüentemente, um diagrama (matriz) pode ser mesclado com textos que representam as cláusulas. Dentro da matriz, o programador pode desenhar, mover, alterar dimensões, remover, duplicar e rotular sub-retângulos. Uma vez criado o programa, o programador invoca o MPL para traduzir os diagramas em predicados Prolog. Após a tradução, o programador precisa informar uma cláusula de averiguação para que o sistema possa executar. Pode-se afirmar que um programa em MPL é consideravelmente mais complexo que o obtido pela proposta deste trabalho. Isto acontece pela necessidade de se aprender Prolog para a utilização do MPL.

Uma outra linguagem para a manipulação de matrizes é o Forms/3 (Ambler et al., 1997). Existem algumas similaridades entre a proposta deste trabalho e o Forms/3. Ambos utilizam a manipulação da representação visual de matrizes para a descrição de algoritmos matriciais. Os dois fazem uso de cores para identificar e selecionar partes de uma matriz. Uma diferença fica por conta da estratégia adotada para a programação de uma matriz. Neste trabalho é utilizado o paradigma de programação por exemplos com o intuito de aumentar a produtividade na implementação de algoritmos matriciais. Uma outra diferença significativa é a adoção de estratégias que procuram reduzir a necessidade de se utilizar a recursão na programação, recurso fartamente utilizado no Forms/3.

Entretanto, a principal diferença deste trabalho em relação aos listados nesta seção é a definição de um modelo híbrido de programação visual de matrizes baseado em fluxo de dados, planilhas eletrônicas e programação por demonstração.

5. Modelo para Programação Visual de Matrizes

As linguagens visuais permitem que, de maneira simples e direta, os programadores esbocem e demonstrem os relacionamentos entre os dados e suas transformações, em vez de traduzirem estes mesmos relacionamentos em comandos textuais, ponteiros e variáveis. Esta simplificação promete tornar mais simples e acessível o modo de programar.

Utilizando-se um estilo de programação simples e concreto, que inclui suporte ao *feedback* imediato, o modelo de programação visual de matrizes (MVM) é fundamentado nos paradigmas de fluxo de dados e planilhas, objetivando, com isso, o provimento de meios computacionais expressivos para o domínio de cálculo matricial. Como a busca por estes objetivos dificulta a construção de uma linguagem visual efetiva para resolver problemas de larga escala (Burnett et al., 1995), procurou-se um equilíbrio na utilização de conceitos tanto concretos como abstratos na modelagem do MVM. Neste sentido, na sua modelagem, sempre foram consideradas as seguintes estratégias:

- Explicitação dos inter-relacionamentos dos elementos sintáticos da linguagem: evitou-se, assim, a construção de programas com dependências escondidas, aquelas que não são totalmente visíveis (explícitas) (Green & Petre, 1996). Por exemplo: o efeito colateral é possível em muitas linguagens de programação devido ao fato das relações entre as variáveis não serem explícitas ou não estarem presentes formalmente no processo de comunicação entre os procedimentos.

- Utilização de poucos conceitos para a programação: os compromissos principais são com a simplicidade e a legibilidade. Conceitos tais como ponteiros, alocação, arquivos e declarações não foram utilizados. Além disso, procurou-se reduzir ao mínimo a necessidade de recursão, um conceito amplamente utilizado nas linguagens visuais pois, na opinião deste autor, não é de fácil aplicação por parte dos usuários inexperientes.

- *Feedback* imediato: com este recurso, utilizado em muitas linguagens de programação visual, quando uma alteração é feita no programa, os resultados são automaticamente atualizados, o que permite a localização mais rápida, e de forma mais consistente, de eventuais erros.

- Programação concreta: é o oposto do conceito de abstração e significa expressar alguns aspectos do programa usando-se instâncias particulares do domínio de interesse (Burnett, 1999).

- Manipulação direta: pode ser definida como o sentimento que o programador experimenta ao manipular diretamente um objeto (Shneiderman, 1983). No contexto deste trabalho, a manipulação direta diz respeito à capacidade do programador de ler, explorar e alterar o programa e seus dados em tempo de desenvolvimento e/ou execução.

- Modelagem de uma linguagem visual heterogênea ou híbrida: esta linguagem combina a conveniência da notação visual com a força de abstração da notação textual (Erwig & Meyer, 1995). Como a modelagem da linguagem é de domínio específico, a notação visual pôde ser mais facilmente integrada à notação textual.

O modelo MVM consiste em um conjunto de diagramas no plano bidimensional e em um conjunto de regras de transformação. Os processos são implementados como redes de fluxo de dados e os dados são representados por planilhas. O programador utiliza a manipulação direta para construir e interagir com cada componente da rede. As planilhas são construídas a partir da definição de fórmulas ou valores para as suas células. As fórmulas podem incluir constantes ou referências a outras células (da mesma matriz ou não). Estas fórmulas podem ser executadas instantaneamente com os dados atuais ou posteriormente, em *batch,* com dados de outras planilhas (matrizes) permitindo, deste modo, que uma planilha possa ser visualizada como um processo (abstração funcional) e não apenas como um repositório de dados de uma matriz.

Os programas concebidos de acordo com o modelo MVM são produzidos como diagramas de fluxo de dados contendo pictogramas que, por sua vez, simbolizam dados e processos. Os dados são representados por retângulos. Já os processos, também descritos por retângulos, possuem duas barras verticais em suas extremidades que indicam, respectivamente, o ponto de entrada dos parâmetros e o ponto de saída dos valores produzidos internamente. Nos dois casos, pode-se usar um rótulo para nomeá-los. A Fig.1a ilustra os dois pictogramas básicos da linguagem. O retângulo representando um processo pode simbolizar tanto uma chamada a uma sub-rotina quanto uma expressão textual. A eventual utilização de texto irá permitir o desenvolvimento de programas mais sucintos. O MVM provê, ainda, um conjunto de recursos computacionais, retratados com ícones próprios, que visam dotá-lo de um instrumental necessário para o desenvolvimento de programas relacionados ao domínio matricial.

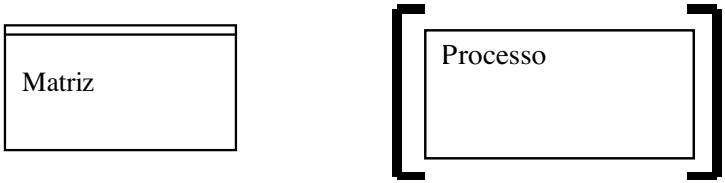


Fig.1a. Representação gráfica de uma matriz e de um processo em MVM.

O modelo MVM permite a abstração funcional por meio de módulos construídos a partir da composição funcional de outros módulos internos (disponíveis na linguagem) ou externos (construídos pelo programador)[3]. A Fig. 2 mostra a interface principal do sistema que implementa o MVM. Nesta Fig., pode-se notar a presença de quatro janelas (lista de matrizes persistentes, de módulos do programa em desenvolvimento, das funções reutilizáveis e, finalmente, do conteúdo do módulo em trabalho, ou seja, a rede de fluxo de dados), de um conjunto de botões e de um menu de opções. A janela que representa o módulo possui duas barras verticais que significam o ponto de entrada (barra à esquerda) e o ponto de saída (barra à direita). O conjunto de botões indica os elementos sintáticos que podem ser utilizados na composição de um programa.

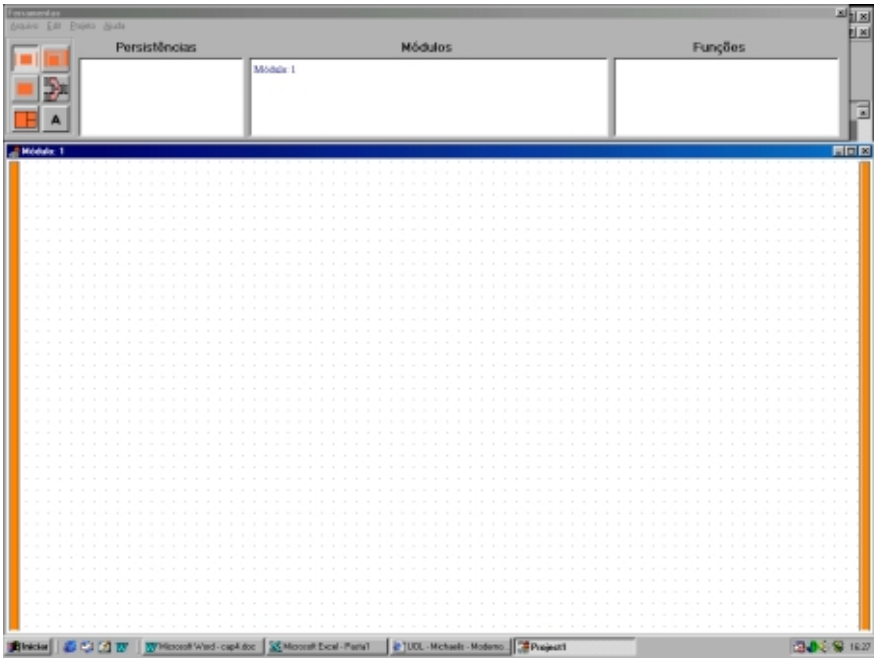


Fig. 2. Interface de processo do modelo MVM.

[3] Neste trabalho, os termos *processos* e *módulos* se referem ao mesmo objeto em diferentes momentos de abstração.

5.1 Matriz

A representação textual de uma matriz pode ser um tanto abstrata e ininteligível para usuários inexperientes e, principalmente, sem treinamento em programação. Por outro lado, a sua representação visual é bem entendida e, muitas vezes, utilizada com propósitos didáticos. Entretanto, para que esta representação visual seja efetiva, o usuário deve ser capaz de manipulá-la sem a necessidade de recorrer a uma notação textual.

Tratando-se de MVM, cada matriz é representada conforme mostrado na Fig. 3. Durante o seu processo de edição, todavia, revela-se graficamente na forma de duas planilhas posicionadas lado a lado que descrevem, respectivamente, os dados da matriz e uma visão sobre os mesmos. A área de dados permite a construção de uma matriz a partir de três estratégias diferentes. A primeira é a designação de valores/fórmulas para as células de uma matriz/planilha. A segunda permite a obtenção de uma matriz a partir da aplicação, a uma ou mais matrizes de entrada, de uma máscara pré-programada: que permitirá, por exemplo, que métodos matemáticos possam ser experimentados em matrizes de dimensões menores e, em seguida, aplicados a outras matrizes maiores. A terceira estratégia gera uma matriz a partir de cópias e reordenações espaciais de outras matrizes ou submatrizes.

A planilha que representa a visão irá descrever como os dados da matriz serão enviados para processos subseqüentes. Esta visão pode ser a própria matriz, um elemento, um vetor ou mesmo um conjunto de submatrizes, seja em um processo iterativo ou não.

As planilhas, em MVM, podem ser vistas como funções ou variáveis do tipo matriz: como funções, recebem valores via parâmetros de entrada e produzem valores que serão utilizados por outros processos; já no papel de variáveis, armazenam dados bidimensionais. Contudo, o sistema não diferencia qualquer um dos dois papéis que uma planilha possa assumir.

Uma matriz pode ser referenciada dentro do procedimento, como uma variável local, ou pode ser acessada globalmente. Neste último caso, é necessário informar que esta matriz é persistente e pode ser manipulada, não apenas dentro do próprio programa, mas também por outros programas, bastando que o usuário defina uma determinada matriz como persistente.

A Fig. 3 ilustra a estrutura geral de uma matriz (N x M). Nesta Fig., a planilha da esquerda descreve os dados de uma matriz que pode conter, opcionalmente: um nome temporário, dois atributos que simbolizam o seu número de linhas e colunas (retângulos na parte superior, à esquerda da Fig.) e setas rotuláveis, utilizadas para informar ou recuperar o número da linha e/ou da coluna em que o sistema está atuando. A planilha da direita, a visão, contém uma barra de saída e duas setas que recuperam, se necessário, o endereço da célula ativa em um processo iterativo.

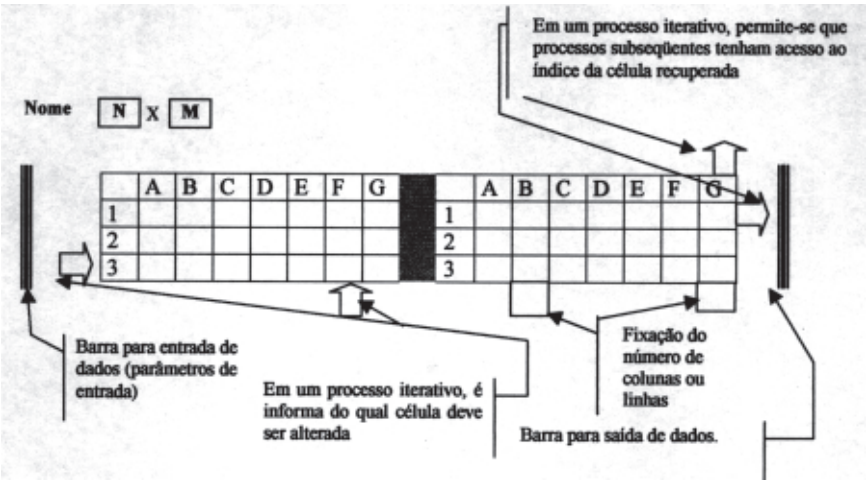


Fig. 3. Representação gráfica de uma matriz em tempo de edição.

Parâmetros de entrada

As planilhas podem receber dados de outras planilhas ou processos, bastando para tanto fazer a conexão de um ou mais fluxos de dados à barra de entrada de dados de uma planilha. Cada conexão recebe automaticamente uma determinada cor e, opcionalmente, um rótulo. Esta cor servirá para identificar os dados de entrada com a planilha de origem. Por exemplo: um determinado termo de uma fórmula, que possua a mesma cor de um dos parâmetros de entrada, indica que aquele termo se refere àquele parâmetro.

Parâmetros de saída

Uma planilha pode enviar dados para outros processos via barra de saída. Neste caso, cada identificador (rótulo ou cor) de algum elemento (escalar, submatriz ou índice) da parte correspondente à visão de uma matriz pode ser utilizado para se relacionar ao item desejado um determinado fluxo conectado à barra de saída.

Uma matriz pode ser criada ou alterada em três diferentes situações: construção a partir de dados de outras matrizes (Seção 5.1.1) e entrada de dados via planilha (5.1.2). As seções que se seguem detalham como ocorre cada uma destas formas descritas.

5.1.1 Criação de matriz via dados de outras matrizes ou funções

Uma matriz pode ser criada ou montada com dados de outras matrizes. Neste caso, cada parâmetro de entrada deve ser nomeado e/ou colorido para que seja unicamente identificado. Cada parâmetro alimentará com dados uma respectiva partição da matriz. Cada partição ocupa uma área retangular que não pode ser sobreposta a outras áreas. A Fig. 4 retrata um exemplo de criação de matriz por composição de partições.

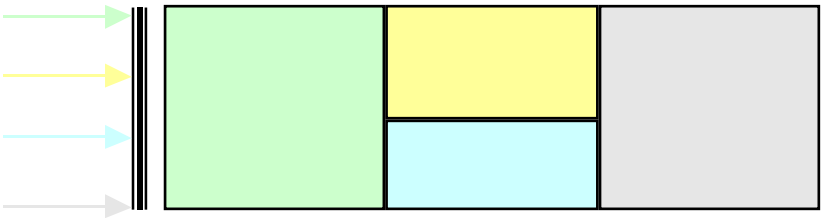


Fig. 4. Criação de uma matriz por composição de quatro partições e quatro parâmetros (matrizes) de entrada.

No processo de edição da matriz, o programador informa se deseja entrar com fórmulas ou proceder a montagem de uma matriz via composição com outras matrizes. Caso a segunda opção seja a escolhida e as matrizes de entrada ainda não existam, já que só serão criadas em tempo

de execução, o sistema verificará a consistência da operação (isto é, se as matrizes possuem dimensões compatíveis com a composição desejada) apenas quando estiverem todas disponíveis. No exemplo expresso pela Fig. 4, as matrizes verde e cinza devem possuir o mesmo número de linhas, enquanto a soma do número de linhas das matrizes amarela e azul deve ser igual ao número de linhas existentes na de cor verde. Por outro lado, se as matrizes de entrada estiverem à disposição em tempo de composição (edição), esta validação de compatibilidade é imediatamente efetivada.

Com o intuito de tornar mais simples esta verificação, tanto para o sistema quanto para o usuário, as partições precisam ser criadas de forma hierárquica. Inicialmente, são criadas as partições maiores com o toque do *mouse* em qualquer uma das linhas fronteiriças: se a linha da fronteira for vertical, cria-se uma partição horizontal; caso contrário, obtém-se uma nova partição vertical. Estas novas partições podem ser divididas novamente seguindo-se o mesmo processo. Quando todas as partições já estiverem definidas, deve-se atribuir a cada uma delas um dos identificadores dos parâmetros de entrada (cor ou nome). A Fig. 5 exemplifica este processo.

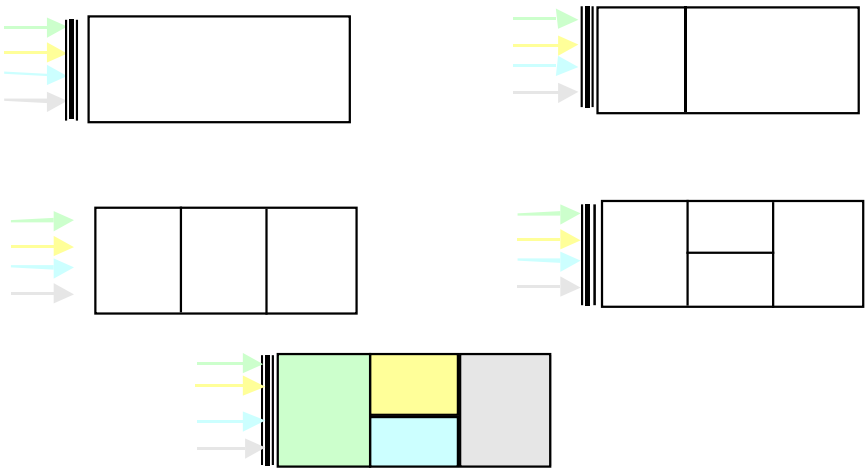


Fig. 5. Passos para a criação de uma matriz a partir de dados fornecidos por outras matrizes.

Os seguintes passos foram realizados na Fig. 5:

1. criação da matriz e de seus parâmetros de entrada;
2. primeira partição é criada com um toque do *mouse* em uma determinada posição da linha superior ou inferior do retângulo que representa a matriz, obtendo-se duas partições;
3. aplicando-se o mesmo processo do passo dois, em outra posição, obtém-se outra partição vertical;
4. com o toque do *mouse* em uma das linhas verticais que delimitam a partição central, geram-se duas partições horizontais;
5. atribuição dos parâmetros de entrada (cores) às partições obtidas.

Criação de uma matriz via padrão de iteração

Uma matriz também pode ser criada iterativamente por meio de dados recebidos de outras matrizes ou funções. Estes dados são repassados via algum processo iterativo, isto é, os dados são recebidos paulatinamente e, para cada dado que chega, um elemento ou bloco de elementos deve ser inserido na nova matriz. Neste sentido, uma matriz pode conter um padrão de iteração para geração de seus elementos. Este padrão pode incluir valores de determinados itens, os seus índices, as dimensões envolvidas e, primordialmente, a ordem como ocorrerá a iteração no processo de geração. Um padrão de iteração contém os itens relacionados na Fig. 6.


representa o valor de um único elemento


representa um conjunto de valores (linhas, colunas, ...)

indica a direção do padrão de iteração


permite que o usuário indique a célula que será gerada


permite que o usuário tenha acesso ao índice da matriz durante o processo de iteração

Fig. 6. Itens para especificar um padrão de iteração.

A Fig. 7 ilustra um vetor que será criado a partir do envio de vários elementos (A e B). Cada par de elementos recebido será multiplicado e uma nova célula do vetor estará definida. Quando não existirem mais dados a serem recebidos, a geração do vetor é finalizada e, com isso, este pode alimentar algum outro processo.

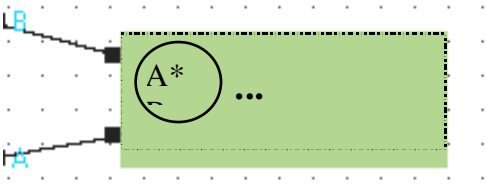


Fig. 7. Geração iterativa de um vetor.

5.1.2 Criação de matriz via entrada direta com planilha

O MVM admite que uma matriz seja criada ou editada manualmente pelo usuário com o auxílio de uma planilha. Cada célula da planilha pode conter constantes ou fórmulas. As fórmulas podem possuir constantes, referências a outras células da mesma planilha ou referências às células de outras planilhas representadas pelos parâmetros de entrada. Diferentemente das planilhas tradicionais[4], a planilha do MVM não utiliza o conceito de endereçamento absoluto. Esta diferença tornou mais simples e intuitivo o processo de cópia de célula que possui fórmula. Uma outra diferença é a exposição da estrutura de fluxo de dados envolvidos nas fórmulas da planilha. Ou seja, a planilha do MVM deixa explícita a estrutura de dependência existente entre as células.

O esquema de cópia de fórmulas adotadas pelas planilhas tradicionais não é muito adequado para as necessidades de usuários discricionários[5], isto é, usuários que necessitam dividir a planilha em uma série de blocos[6] e aplicar uma função para cada um destes blocos. Por exemplo, considere um problema em que uma série de números são colocados na

---

[4] Para este trabalho, as planilhas tradicionais são aquelas baseadas no paradigma do VISICALC, isto é: EXCEL, LOTUS 123, etc.

[5] Usuários que trabalham com dados discretos ou descontínuos.

[6] Um bloco é retangular e formado por um conjunto de células.

primeira linha de uma planilha comercial para se proceder ao agrupamento desses dados em blocos de três e computar a sua soma (ver Fig. 8). A resolução deste problema requer uma boa capacidade de programação por parte de usuários destas planilhas, uma vez que várias funções precisam ser combinadas em uma fórmula razoavelmente complexa.

	A	B	C	D	E	F	G	H	I	J	K	L
1		3	3	4	2	1	3	4	2	1	4	...
2												
3		10	6	7	...							
4		B1	E1	H1	...							
5		D1	G1	J1	...							

Fórmulas associadas às células da coluna B que serão copiadas para outras colunas (c,d,...):

B3 → =SOMA(INDIRETO(B4):INDIRETO(B5))

B4 → =ENDEREÇO(COL(A1);COL(A1)+3*(COL()-COL(A1));4)

B5 → =ENDEREÇO(COL(A1);COL(A1)+3*(COL()-COL(A1))+3-1;4)

Fig. 8. Soma de sucessivos blocos de tamanho 3 orientados horizontalmente.

A Fig. 8 ilustra uma solução particular para a planilha Excel; soluções similares são requeridas em planilhas de outros desenvolvedores. Esta solução exige a criação, ou programação, de uma fórmula complexa, obrigando o usuário a conhecer detalhes acerca de funções e seus parâmetros para resolver o problema sugerido acima. Para outros tipos de problemas relacionados à aplicação de fórmulas a diferentes formatos de blocos, outras soluções igualmente complexas precisariam ser formuladas.

A Fig. 9 retrata graficamente alguns exemplos de blocos e sua respectiva direção para formação de novos valores da planilha. No exemplo 1, a distância entre os blocos[7] é maior que 1. Nos exemplos 2, 3 e 5, a distância entre os blocos é exatamente igual a 1. Finalmente, no exemplo 4, a distância entre os blocos é menor que 1.

[7] A distância entre blocos é aqui definida como o número de células que separam dois blocos.

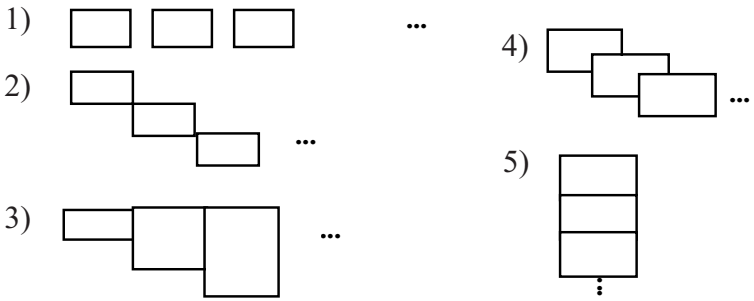


Fig. 9. Exemplos de padrões para geração de valores em uma planilha.

A razão pela qual a soma de blocos não pode ser resolvida com os recursos padrão de cópia de células está relacionada com a forma segundo a qual as referências das células são atualizadas no processo de cópia. As planilhas tradicionais ajustam automaticamente os endereços existentes nas fórmulas pela distância da fórmula fonte da cópia. Para inibir estes ajustes, o usuário pode especificar uma célula com endereçamento absoluto (com a utilização do símbolo '$'). Todavia, a colocação deste símbolo não é uma tarefa intuitiva, o que exige a verificação do resultado obtido. A formalização deste processo, baseada no trabalho de Hendry (1995), mostra a limitação da notação tradicional utilizada pelas planilhas e, em adição, sugere um caminho para aperfeiçoá-la.

Quando uma fórmula é copiada e colada em uma região da planilha, diversas variáveis devem ser mantidas pelo sistema.

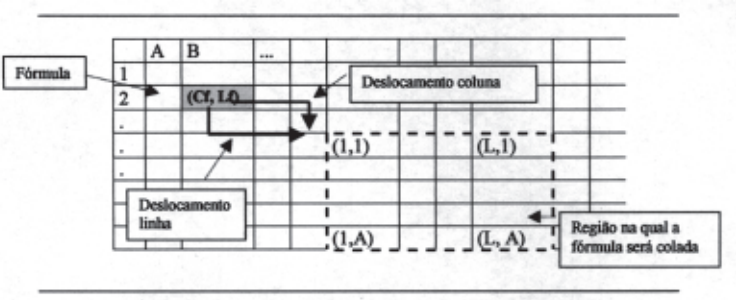


Fig. 10. Variáveis envolvidas no processo de cópia de fórmulas nas planilhas tradicionais.

Na Fig. 10 tem-se a representação esquemática das variáveis envolvidas no processo de cópia de fórmulas nas planilhas tradicionais. A fórmula a ser copiada é representada pela região com endereço: Cf e Lf, que são os endereços da linha e da coluna da fórmula de entrada (por exemplo, para referência à célula B2, tem-se: Cf = B e Lf = 2). Os deslocamentos de linha e coluna indicam a distância da fórmula à região de saída (colagem da fórmula), enquanto L e A representam, respectivamente, a largura e a altura da região de saída. Quando uma fórmula é colada, é assentada em cada célula da região de colagem, mas as referências são atualizadas para refletir a distância desta célula à fórmula de entrada. Assumindo que cada nova referência nas células da região de saída é representada pelo par ordenado (Ci, Lj), a derivação da semântica da operação de cópia pode ser especificada por equações que computam valores corretos de $C_0, C_1, C_2, ..., C_L$ e $L_0, L_1, L_2, ..., L_A$ da região de saída. Assim:

$$C_0 = C_f + \text{deslocamento da coluna} \quad (1)$$

Os endereços das colunas à direita de C_0 podem ser computados a partir de C_0, produzindo-se:

$$C_1 = C_0 + 1,\ C_2 = C_0 + 2, ..., \quad C_L = C_0 + L\text{-}1. \quad (2)$$

O mesmo procedimento pode ser adotado para computar os endereços das linhas na região de saídas, desta forma:

$$L_0 = L_f + \text{deslocamento da linha}, \quad (3)$$

$$L_1 = L_0 + 1,\ L_2 = L_0 + 2, ..., \quad L_A = L_0 + A\text{-}1 \quad (4)$$

As fórmulas apresentadas adequadas para descrever o processo de atualização dos endereços relativos. Por outro lado, se o endereço na fórmula é absoluto, a distância de colagem da cópia não tem qualquer significado, pois os endereços utilizados não devem ser atualizados, assim:

$$C_0 = C_{f0}, \quad C_1 = C_{f1}, \quad ..., C_L = C_{fL} \quad (5)$$

$$L_0 = L_{f0}, \quad L_1 = L_{f1}, \quad ..., L_L = L_{fL} \quad (6)$$

A partir das fórmulas citadas, pode-se derivar as seguintes relações para cálculo dos endereços das colunas e linhas:

$$C_i = \begin{cases} C_f + \text{deslocamento coluna} + i & \text{Caso } C_f \text{ com endereço relativo} \\ C_f & \text{Caso } C_f \text{ com endereço absoluto} \end{cases} \quad (7)$$

$$Li = \begin{cases} L_f + \text{deslocamento linha} + j & \text{Caso } L_f \text{ com endereço relativo} \\ L_f & \text{Caso } L_f \text{ com endereço absoluto} \end{cases} \quad (8)$$

Onde: $0 \le i \le L\text{-}1$ e $0 \le j \le A\text{-}1$.

De (7) e (8), pode-se produzir as seguintes fórmulas análogas:

$$C_i = C_f + \sum_{k=1}^{DeslocamentoColuna} dc + \sum_{k=1}^{i} dc \quad (9)$$

$$L_j = L_j + \sum_{k=1}^{DeslocamentoLinha} dl + \sum_{k=1}^{j} dl \quad (10)$$

Onde: Se endereçamento relativo, $dc = 1$ e $dl = 1$

Se endereçamento absoluto, $dc = 0$ e $dl = 0$

$0 \le i \le L\text{-}1$ e $0 \le j \le A\text{-}1$

Analisando a fórmula apresentada e tendo em mente as planilhas tradicionais, pode-se observar que os valores das variáveis *dc* e *dl* estão restritos a 0 e 1, indicando o endereçamento absoluto ou relativo (inserção ou não de $ nos endereços das fórmulas). Todavia, estas mesmas variáveis podem ser utilizadas para produzir um processo de cópia mais expressivo, que consiga gerar progressões que foram exemplificadas na Fig. 9.

Hendry (1995) propôs uma técnica baseada em um estilo simples de programação por demonstração. Nesta técnica, três passos são necessários:

1. escrever as duas primeiras fórmulas da progressão;
2. selecionar a região que delimita a região de geração da progressão;
3. executar a cópia da fórmula para as células da região demarcada.

Para aplicar esta técnica, basta calcular os valores de *dc* e *dl*, que são derivados das duas primeiras fórmulas fornecidas no passo 1, desde que possuam o mesmo número de termos. Assim, dadas duas fórmulas seqüenciais (C_0, L_0) e (C_1, L_1), por exemplo, pode-se resolver os valores de *dc* e *dl* a partir das seguintes expressões:

$$dc = C_1 - C_0 \quad (11)$$
$$dl = L_1 - L_0 \quad (12)$$

A partir deste cálculo, todo o processo de indução pode ser iniciado. A Fig. 11 exemplifica alguns casos de progressão que, baseados nos dois valores iniciais, admitem que uma série de computações sejam produzidas. Neste exemplo, o retângulo tracejado indica resultado da cópia.

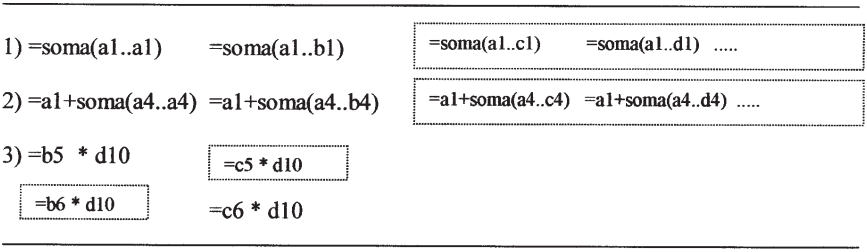


Fig. 11. Exemplos que não podem ser expressos com o esquema usual de cópia utilizado pelas planilhas tradicionais.

No exemplo 1 da Fig. 11, uma soma cumulativa é computada por coluna. O cálculo de *dc, dl* por diferença de termos produz: termo1, (*dc, dl*) = (a-a, 1-1)= (0,0); termo2, (*dc, dl*) = (b-a,1-1) = (1,0).

O segundo exemplo calcula uma soma cumulativa na qual, a cada termo, adiciona-se uma constante (termo1, (*dc, dl*) = (a-a,1-1) = (0,0); termo2, (*dc, dl*) = (a-a,4-4) = (0,0); termo3, (b-a, 4-4) = (1,0)).

Finalmente, o terceiro exemplo multiplica uma constante a cada célula do bloco de saída (termo1, (*dc, dl*) = (c-b, 6-5) = (1, 1); termo2, (*dc, dl*) = (d-d, 10-10) = (0,0). Este exemplo possui uma outra particularidade a ser observada em relação às fórmulas (9) e (10). A mudança do componente coluna (C*i*) só ocorre se *i* é alterado. Similarmente, Lj altera se *j* for alterado. Esta característica significa que, se as duas primeiras fórmulas forem colocadas horizontalmente, o passo 2 do processo de indução só pode ser horizontal. O mesmo ocorre se as fórmulas iniciais do processo indutivo forem colocadas verticalmente. Todavia, se o processo de indução é diagonal, tanto Ci como Lj podem ser alterados.

Esta estratégia de indução elimina totalmente a necessidade de se utilizar algum símbolo especial (por exemplo, $) para fixar uma referência em um processo de cópia. Por outro lado, introduz um aspecto negativo que se refere a uma maior dificuldade de se saber, de maneira direta, que um determinado termo da fórmula é uma constante. Este aspecto contrário pode ser minimizado pela utilização de algum recurso visual que indique que um termo é constante.

O modelo MVM introduz um processo de indução gráfica apoiado na proposta de Hendry, contudo estendido para trabalhar com o conceito de abstração procedimental. Por este processo, o usuário deve editar dois exemplos consecutivos para que o sistema apresente uma proposta gráfica de indução, que pode ser aceita ou modificada. Esta proposta é gerada com base nas duas primeiras fórmulas e é estruturada em três partes: a fórmula-base, o fluxo de dados e a região a ser preenchida, que pode possuir opcionalmente um parâmetro de limitação. Este parâmetro de limitação é fundamental quando este processo de indução for utilizado como função (ver Seção 5.3). A implementação atual do MVM trabalha com os passos relacionados a seguir:

1. Edição textual das duas primeiras fórmulas indutoras. O número de termos existentes nestas duas fórmulas deve ser igual, entretanto o número total de termos pode ser desconhecido;

2. Ativação do botão para preenchimento da região. O sistema sugere uma fórmula para o terceiro elemento da série. Neste ponto, o usuário pode reeditar os dois primeiros itens da série;

3. Aceitação do terceiro elemento da série. Será proposta ao usuário uma área para preenchimento da seqüência. Esta área pode ser editada gráfica ou textualmente, permitindo ao usuário a sua delimitação;
4. O usuário pode requerer que a fórmula deva ser aplicada a todo o bloco, linha a linha ou coluna a coluna, de acordo com a direção das duas primeiras fórmulas do passo 1. Caso isto seja desejado, será necessária a edição textual da fórmula de transição de linha para coluna ou de coluna para linha (ver Seção 5.3);
5. Se os dados de entrada estiverem disponíveis, o preenchimento é executado em tempo real. Caso contrário, serão preenchidos no futuro com dados de outra planilha.

No passo 3 o usuário precisará confirmar ou alterar a proposta de região de preenchimento. Inicialmente, o sistema propõe uma área a ser preenchida. Caso o usuário deseje modificar esta região, pode seguir três caminhos. O primeiro consiste na modificação do ponto final de indução com a utilização do *mouse*. A segunda possibilidade é a definição do número total de células da série que serão geradas: esta informação pode ser um número absoluto, uma fórmula, um parâmetro recebido pela matriz na estrutura de fluxo de dados do programa, ou uma condição. Finalmente, pelo não estabelecimento de qualquer limite pré-definido. Em qualquer um dos dois últimos casos, deve-se utilizar o símbolo visual que indica a direção da repetição com ou sem limite estabelecido. A Tabela 1 lista exemplos de composições que definem a direção e a condição de parada. A seguir, na Tabela 2, tem-se um sumário dos símbolos de delimitação.

A utilização do construtor de repetição é particularmente útil quando os termos das fórmulas iniciais dependem de matrizes externas, isto é, fornecidas como parâmetros para a planilha que contém as fórmulas. Neste caso, cada termo da fórmula precisaria ter a sua cor definida de acordo com as cores dos parâmetros ou possuir um identificador extra de acordo com o rótulo do parâmetro de entrada desejado. Os termos da fórmula que tenham cor preta e não possuam identificador de parâmetro são referências à célula da mesma matriz.

A Fig. 12 retrata dois exemplos de fórmulas que manipulam dados de duas matrizes de entrada. O primeiro exemplo utiliza cores para distinguir a origem dos termos utilizados na fórmula, enquanto o segundo utiliza os nomes das matrizes de entrada. Em ambos os casos, no entanto, a direção da indução será preestabelecida pela direção das duas primeiras fórmulas.

Fig. 12. Exemplos de utilização de parâmetros nas fórmulas.

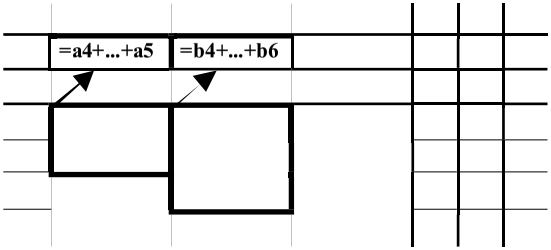
Tabela 1. Exemplos de utilização do construtor de repetição para geração de valores em uma planilha.

Exemplo	*Descrição*
... {10}	Aplicar a fórmula de indução em dez colunas consecutivas.
...{N}	Aplicar a fórmula de indução em N colunas consecutivas. O valor de N deve ser um dos parâmetros da planilha.
...	Aplicar a fórmula de indução até a última coluna da planilha. Observe que, às vezes, a dimensão da planilha só é conhecida em tempo de execução, dependendo das dimensões das planilhas de entrada.
{N*2} ⋮	Aplicar a fórmula de indução em N x 2 linhas consecutivas da planilha.
⋱	Aplicar a fórmula de indução até o último elemento da diagonal pretendida.
...{← = 1}	Aplicar a fórmula de indução até que a célula horizontal imediatamente anterior tenha valor igual a 1.
{↑=0} ⋮	Aplicar a fórmula de indução até que a célula vertical imediatamente anterior tenha valor igual a zero.
...{•=←}	Aplicar a fórmula de indução até que dois valores consecutivos sejam iguais, isto é, o valor da célula atual seja igual ao valor da célula anterior.
...{←+ > 100}	Aplicar a fórmula até que a soma de todos os valores gerados na horizontal seja maior que 100.
{↑+ > 100} ⋮	Aplicar a fórmula de indução até que a soma de todos os valores anteriores na vertical seja maior que 100.

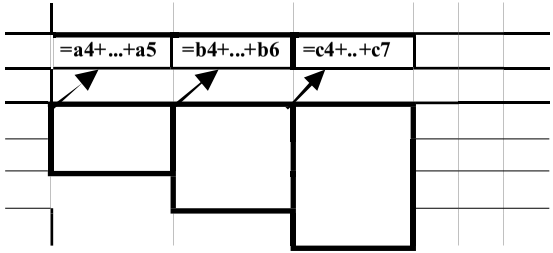
Tabela 2. Símbolos visuais que podem ser utilizados para delimitar o número de células a serem geradas pelo processo de indução.

Símbolo	*Derivação*	*Descrição*
...		Direção horizontal da indução: se colocada após as duas fórmulas-base, a direção será horizontal da esquerda para a direita.
⋮		Direção vertical da indução: se colocada após as duas fórmulas-base, a direção será vertical de cima para baixo.
{P}		O processo de indução cessará quando P for satisfeito.
	P :: Nulo |	O processo de indução não tem limite preestabelecido, depende apenas da quantidade dos dados de entrada.
	P :: Número |	O processo de indução terminará quando o número de células geradas for igual à variável ou ao número indicado.
	P :: Variável |	O processo de indução terminará quando o número de células geradas for igual ao valor armazenado pela variável indicada. Esta variável pode ser um parâmetro explícito ou uma das variáveis internas de uma matriz ou região (LI, LF, CI, CF).
	P :: Fórmula |	O processo finalizará se o número de fórmulas geradas for igual ao valor produzido pela fórmula.
	P :: Condição	O processo cessará se a condição for satisfeita.
←		Indica valor da célula imediatamente anterior na horizontal.
↑		Indica valor da célula imediatamente anterior na vertical.
•		Indica valor da fórmula atual, ou seja, da célula que está sendo gerada.
←+		Representa o somatório de todos os valores horizontais anteriores calculados pela aplicação da fórmula de indução.
↑+		Representa o somatório de todos os valores que foram gerados e antecedem, na vertical, a célula atual.

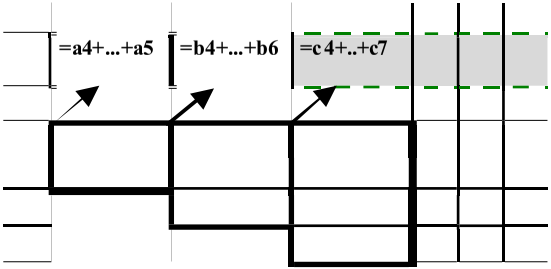
A Fig. 13 mostra um exemplo de indução criado pelo mecanismo de indução do MVM. Primeiramente, o usuário precisa informar as duas primeiras fórmulas da indução (1). O sistema deverá propor uma fórmula como terceiro elemento da série (2). A seguir, o usuário será inquirido a acatar ou modificar a sugestão da região que será preenchida (retângulo tracejado em 3). Finalmente, em (4), o usuário deverá informar se o preenchimento será feito no futuro, com dados de outra matriz, ou no instante atual.



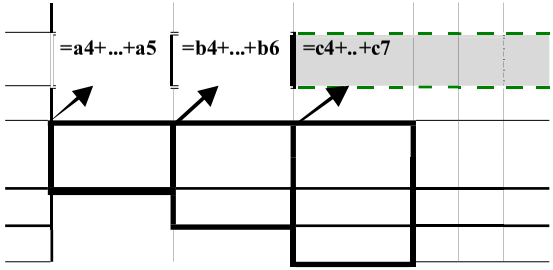
1) Entrada das duas primeiras fórmulas.



2) Indução do terceiro elemento da série a partir dos dois primeiros.



3) Sugestão da área de preenchimento



4) Usuário decide se a indução será executada no momento ou não.

Fig. 13. Seqüência esquemática dos passos envolvidos no processo de indução de fórmulas.

Os fluxos de dados envolvidos entre as células e as fórmulas das planilhas tradicionais são invisíveis. Esta invisibilidade é uma fonte potencial de problemas, uma vez que o usuário não sabe, sem investigar o conteúdo da célula, quais células afetam e são afetadas por uma determinada fórmula. Um problema correlato ocorre quando o usuário precisa entender uma planilha criada por outro usuário. A seguir, será detalhado como o sistema proposto torna explícito o fluxo de dados em uma planilha.

5.2 Explicitando os Fluxos de Dados das Fórmulas

Várias soluções foram propostas e implementadas para tornar mais perceptíveis as relações existentes em uma fórmula. Por exemplo: a planilha Excel da Microsoft™, uma das planilhas com mais recursos existentes no mercado, inclui duas técnicas que provêm uma visualização limitada do fluxo de dados existente em uma dada célula. A primeira é invocada quando o campo que mostra o conteúdo da célula é editado. Nesta situação, cada endereço envolvido na fórmula recebe uma cor e as regiões correspondentes a estes endereços recebem a mesma cor. A segunda é ativada quando se utiliza a ferramenta de auditoria, que desenha arcos entre a célula que contém a fórmula e seus ascendentes e descendentes.

Estes recursos tentam tornar o fluxo de dados, que se encontra escondido, visualmente acessível. Todavia, precisam ser explicitamente ativados via menu ou barra de ferramentas, e são limitados a uma única célula por vez. Não existe qualquer forma de se mostrar todo o fluxo de dados da planilha de uma única vez.

A planilha do MVM possui duas técnicas adicionais, que visam minorar os problemas acarretados pela invisibilidade do fluxo de dados. Em primeiro lugar, tem-se a visualização transitória, que retrata a estrutura de fluxo de dados associada com células individuais. A segunda compreende a visualização global com o objetivo de permitir a representação de todo o fluxo de dados da planilha de uma única vez. A seguir, será descrito o modo de operação dessas duas técnicas.

Visualização transitória

A técnica de visualização transitória permite que o usuário veja parte do fluxo de dados associado com a célula fórmula que se está interagindo. Esta técnica é baseada na proposta de Igarashi & Mackinlay (1998) e mostra tanto o fluxo das células que afetam a fórmula como as células que são afetadas por esta fórmula. O sistema distingue visualmente estes dois tipos de células com a utilização de arcos. Regiões adjacentes a células que afetam uma fórmula são agrupadas com um retângulo. A Fig.14 ilustra um exemplo da estrutura de fluxo de dados envolvido em uma fórmula. Neste exemplo, a célula E5 foi criada pela soma da diagonal A1 até D4, enquanto a célula E1 foi obtida pela multiplicação por 2 do resultado da célula E5

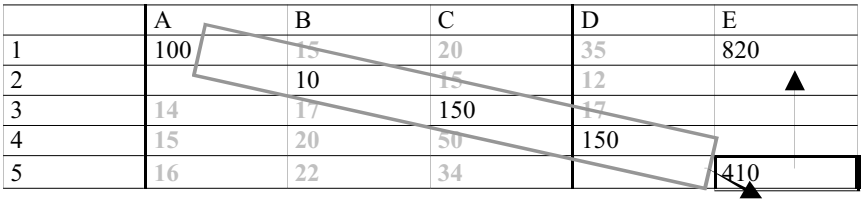


Fig. 14. Exemplo de representação de fluxo de dados em planilhas com fórmula.

O nome de transitória que denomina esta técnica recai na forma pela qual a estrutura de fluxo de dados é ativada pelo usuário. Nas aplicações convencionais de planilha, o usuário deve mover o cursor até a célula de interesse e ativar, via *mouse,* uma opção do menu que ativa o desenho do fluxo de dados envolvido em uma fórmula. No MVM, o usuário especifica a célula de interesse movendo o cursor sobre a célula. Quando o cursor está em cima de uma célula, o grafo do fluxo de dados aparece gradualmente. Os valores (números) envolvidos nos cálculos da fórmula são realçados, enquanto os dados não-envolvidos são levemente apagados. Este grafo desaparece gradualmente quando o cursor é movido para outra célula. Desta maneira, o usuário pode explorar a estrutura de fluxo de dados da planilha movendo o cursor pelas células da planilha. O usuário, assim, detém o controle do nível de informação que o sistema disponibilizará.

A visualização transitória não requer qualquer operação especial para mostrar graficamente o fluxo de dados, permitindo que o usuário mantenha seu foco de atenção apenas no conteúdo da planilha e no movimento do cursor. Todavia, fica limitada a uma única célula por vez, não permitindo a visualização da estrutura de fluxo de dados de toda a planilha. Neste tipo de visualização, o MVM estendeu a proposta de Igarashi possibilitando que toda a matriz, independentemente de sua dimensão, esteja representada na janela correspondente. A seguir é descrita esta extensão.

Visualização global

Existem situações em que é necessária a visualização de todo o fluxo de dados existente na planilha de uma única vez. Por exemplo, o usuário pode proceder a uma rápida revisão de toda a estrutura de fluxo de dados de uma planilha montada por outro usuário.

Neste procedimento, todos os arcos que representam fluxos de entrada e saída são representados na tela. Para minimizar o número de arcos desenhados, células podem ser representadas como grupos. Estes grupos podem ser arranjados de maneira vertical, horizontal ou diagonal. Cada um destes arranjos possui uma cor predeterminada. Estes arranjos têm como finalidade fornecer uma idéia de como a planilha está estruturada. A Fig. 15 retrata um exemplo de planilha na qual o valor do último elemento de cada linha e coluna e da diagonal principal constitui a soma dos elementos das respectivas linhas, colunas e diagonal.

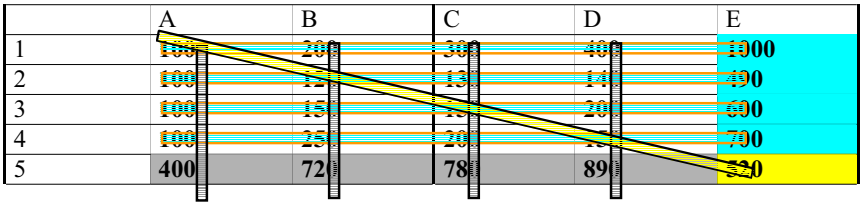


Fig. 15. Exemplo de visualização global.

A visualização global pode ser requisitada apenas para uma parte da planilha, bastando para isso selecionar um conjunto de linhas, colunas ou células individuais e requerer que o sistema mostre o fluxo de dados de entrada e saída das células escolhidas. É uma adaptação da proposta de Igarashi & Mackinlay (1998). A principal diferença ocorre na visualização de uma matriz grande que, em MVM, pode ter a sua visualização reduzida para caber dentro da janela correspondente. Neste caso, o usuário teria uma espécie de fotografia das relações existentes entre todas as células e não o seu conteúdo.

Mesmo com o agrupamento de células, a sobreposição de várias partes do grafo é inevitável e pode dificultar a determinação do fluxo de dados de uma determinada célula. Todavia, espera-se que a técnica de visualização global forneça um esboço geral de como a planilha está estruturada apenas para facilitar o seu entendimento.

5.3. Generalização de Uma Matriz: Um Nível de Abstração Procedimental

O MVM permite que o usuário defina uma matriz como uma função que recebe valores genéricos, processando-os e produzindo uma outra matriz como resultado desta operação. O usuário poderá programar as células desta matriz com o objetivo de produzir a funcionalidade desejada, e a programação poderá ser efetuada de duas maneiras diferentes. A primeira constitui uma programação estática na qual a dimensão da matriz é determinada em tempo de programação e os valores de suas células são calculados a partir dos parâmetros de entrada. As matrizes correspondentes aos parâmetros de entrada podem estar disponíveis em tempo de compilação[8] ou de execução. A validação de conformidade só será possível quando as matrizes de entrada forem criadas.

A segunda maneira de programação é mais genérica e envolve a criação dinâmica de uma matriz em que a dimensão final pode ou não ser de-

[8] No MVM os termos tempo de compilação e programação referem-se ao mesmo espaço de tempo.

terminada pelo usuário, e as células serão inseridas na planilha em tempo de execução. Esta inserção dependerá de um processo indutivo baseado em exemplos definidos para algumas células pré-inseridas. Além disso, o processo de cálculo dos valores das células pode envolver alguma relação de recorrência que precisa ser programada pelo usuário. Todavia, com o intuito de tornar mais fácil o processo de programação das relações de recorrência da planilha, o MVM abrange um método de programação por exemplos, baseado na estrutura geral de cópias de células já descrita na Seção 5.1.2.

Na definição das relações de recorrência, a regra a ser seguida pelo usuário é a definição das duas primeiras fórmulas de um conjunto de valores a serem gerados. A consistência desta regra é evidentemente baseada na pressuposição de que as duas fórmulas programadas pelo usuário devem possuir os mesmos tipos de objetos (operadores, endereços e constantes) e na mesma ordem, isto é, as fórmulas devem possuir um padrão estrutural regular. As únicas diferenças admitidas entre estas fórmulas (exemplos) são os valores das constantes e os endereços envolvidos nos cálculos. A principal conseqüência desta regra é que o sistema deverá repetir o exemplo com possíveis mudanças de valores, mantendo-se, contudo, a estrutura das fórmulas iniciais. Baseando-se no mesmo princípio, a este método de geração de células foi acrescentado o conceito de geração de blocos. Desta forma, uma matriz pode ser definida como sendo uma composição de blocos aninhados ou hierarquizados que possuem o seu próprio modelo de geração de valores para as suas células.

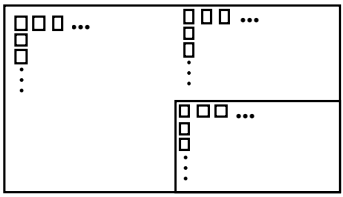
Para um maior esclarecimento, convém considerar a questão relacionada à multiplicação de duas matrizes. Uma possível solução em MVM seria a programação das duas primeiras células da primeira linha da matriz, e a posterior geração da fórmula indutora. Esta solução limitaria as situações em que a matriz poderia ser utilizada como função. Por exemplo: é perfeitamente possível a criação de uma função para multiplicação de duas matrizes usando-se o conceito descrito na Seção 5.1.2, porém não seria viável, sem a introdução dos blocos, a construção de uma função para o produto de Kronecker[9].

---

[9] Produto matricial de duas matrizes (An_xm e Bp_xq) no qual cada elemento da matriz A é multiplicado por todos elementos das matriz B, produzindo uma matriz de dimensão np x mq.

A generalização de uma matriz é uma extensão do conceito de geração de fórmula por indução. A principal diferença está na forma de utilização da planilha. O processo de indução descrito anteriormente é aplicado a uma planilha já estruturada, mesmo que a sua dimensão não seja conhecida. A generalização permite a estruturação da planilha por meio da inserção, em uma planilha vazia, de blocos, de sub-blocos (divisões do bloco), de célula e, finalmente, da especificação das fórmulas indutoras para as células dos blocos mais internos. Os blocos, por sua vez, podem ser estáticos ou dinâmicos. Os blocos estáticos não são replicados, apenas as células internas aos mesmos. Por outro lado, os blocos dinâmicos são replicados de uma maneira similar à replicação das células. A Fig. 16 exemplifica ambos os casos.

Em qualquer dos casos descritos anteriormente, continua possível a criação de blocos aninhados que obedeceriam às mesmas regras de formação. Convém observar que este método de geração de matrizes baseado em exemplos visa a reduzir a utilização do conceito de recursão e, em adição, funciona como um poderoso construtor de repetição, uma vez que pode embutir vários padrões de repetição (ver comandos de repetição da Fig.16*)*, aninhados ou não, que atuam nos dados de uma matriz.

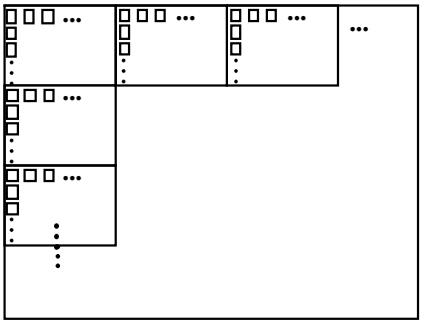


Esquema de repetição implementado por esta matriz:

```
For .... do begin
...
end;

For ... do begin
...
end;

For ... do begin
...
end;
```



Esquema de repetição implementado por esta matriz:

```
For .... do begin
        For ... do begin
                For ... do begin
                        ...
                end;
        end;
end;
```

Fig. 16. Exemplos de inserção de blocos, células e fórmulas indutoras para criação de matrizes.

A Fig. 16 merece alguns comentários. A matriz da esquerda utiliza três blocos estáticos, nos quais um processo indutor será responsável pela geração das células. A matriz final será obtida após o processamento dos três blocos. A matriz da direita possui apenas um bloco, que será replicado por linha e por coluna, seguindo-se algum critério definido. Para cada um dos blocos, foi inserido um indutor para geração dos valores dos mesmos. Abaixo de cada matriz existe um fragmento de código, que retrata o esquema de repetição que está embutido em cada matriz do exemplo.

A Fig. 17 ilustra o exemplo para multiplicação matricial. Convém observar neste exemplo que existem três diferentes fórmulas de indução: a primeira está relacionada com a geração de fórmulas para uma linha; a segunda trata da transição de uma linha para outra; por fim, a terceira fórmula atua dentro de uma célula e indica como será a geração de novas fórmulas para cada célula, uma vez que ainda não se sabe as dimensões das matrizes de entrada. A programação desta planilha obedeceria a uma seqüência de passos, sendo que o primeiro consistiria em escrever as fórmulas que formariam a base da indução. O sistema responderia com uma proposta de fórmula. O usuário, então, determinaria quantas células seriam geradas (no caso, número indeterminado), e forneceria a fórmula de transição de uma para outra linha da matriz. Finalmente, o usuário iria requerer o encapsulamento da função e a nomearia. O sistema responderia com um ícone na janela de funções, com os respectivos pontos para ancoradouro dos parâmetros e com suas respectivas cores derivadas das fórmulas (em (2)). A seta à direita constitui apenas uma indicação visual de que o processo de indução será executado linha a linha. Convém notar, ainda, que as caixas com as bordas vermelhas retratam as fórmulas indutoras que foram geradas pelo sistema.

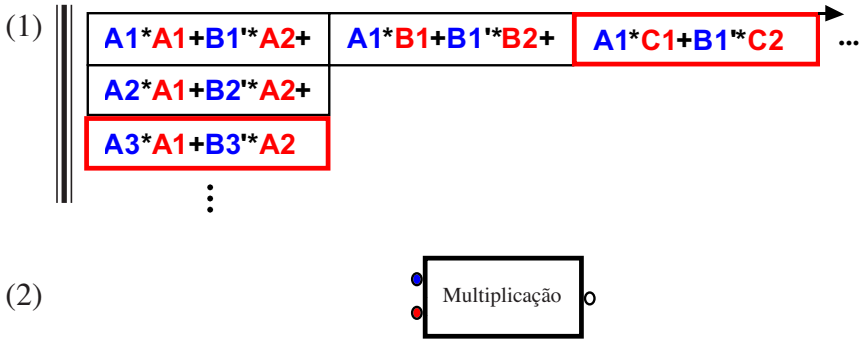


Fig. 17. Exemplo de uma função, baseada em planilha, para a multiplicação de duas matrizes.

Um outro fator relevante é a verificação da consistência das dimensões envolvidas com a operação desejada. Novamente considerando o exemplo anterior, convém notar que as referências às células das duas matrizes de entrada ocorrem em pares. Como o número de termos que serão gerados depende das dimensões das matrizes, a fórmula só é expandida a bom termo se, e somente se, o número de colunas da matriz identificada pela cor azul for igual ao número de linhas da matriz registrada em vermelho. Em outras palavras, a verificação da conformidade das dimensões das matrizes de entrada com a operação desejada somente será válida se as fórmulas internas puderem ser expandidas.

A Fig. 18 exemplifica a programação de uma planilha para produzir a multiplicação de Kronecker de duas matrizes. O usuário precisou definir, primeiramente, três blocos externos com as respectivas direções das iterações. Para cada um dos blocos, definiu células, fórmulas de indução e direção da indução. Observa-se que a seta indica que todo o processo será executado por linha. Novamente, os diagramas com bordas em vermelho simbolizam fórmulas e/ou blocos gerados automaticamente pelo sistema para que se possa validar o indutor. Se a matriz azul possuir dimensão *m x n* e a matriz vermelha possuir dimensão *r x q*, a dimensão da matriz resultado será: *mr x nq*.

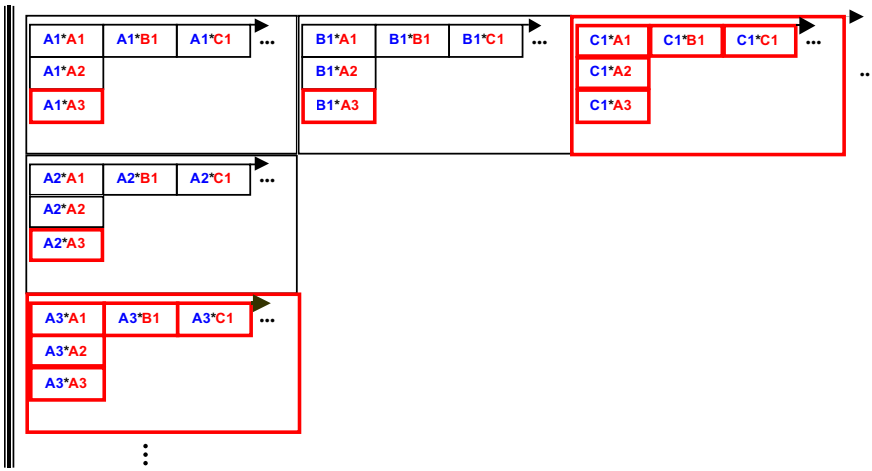


Fig. 18. Programação de uma planilha para produzir a multiplicação de Kronecker de duas matrizes.

A dimensão final da matriz produzida será diretamente proporcional ao número de iterações aninhadas que serão executadas. Este número de iterações pode ser delimitado com alguma condição de parada. Caso nenhum delimitador tenha sido especificado, o sistema deve prever quantas iterações serão executadas. Além do mais, mesmo que uma determinada condição de parada tenha sido fornecida, é necessário o cálculo máximo de iterações que um determinado modelo de indução poderá gerar sem causar erros de endereçamento.

O número máximo de iterações, ou seja, número de vezes que a fórmula indutora será executada, pode depender das dimensões das matrizes de entrada, do número de termos existentes na fórmula indutora e da distância dos índices envolvidos nos termos de duas fórmulas consecutivas do modelo de indução.

5.4. Redução e Partição de Uma Matriz

O MVM possui dois construtores básicos que podem reduzir a dimensão de uma matriz de forma dinâmica ou estática. São os construtores para redução e partição de matrizes. O construtor de redução é utilizado quando uma operação é para ser aplicada de maneira seletiva. A seleção pode ser assentada na posição do elemento como, por exemplo, para selecionar todos os elementos que estejam posicionados na diagonal superior da matriz, ou pode ser firmada no valor do elemento como, por exemplo, a escolha de todos os elementos que sejam maiores que zero. Este construtor tipicamente reduz a matriz em termos de cardinalidade.

O construtor de partição divide uma matriz, inicialmente definida, em várias regiões independentes. Estas regiões podem ser utilizadas para descrever outras matrizes ou podem ser usadas em outros processamentos ou fórmulas. Estes dois operadores visam fornecer uma alternativa ao conceito do construtor de repetição existente nas linguagens de programação imperativas pois, na maioria das vezes, estas repetições são utilizadas para assinalar valores a um conjunto de elementos de uma estrutura ou para selecionar (acessar) um conjunto de elementos de uma estrutura.

5.4.1 Redução

O operador de decisão visa selecionar um conjunto de dados (fluxo) que deverão ser encaminhados para a parte verdadeira ou falsa do operador. Esta seleção é efetuada pela aplicação de operadores lógicos e booleanos aos argumentos de entrada do operador. Se a expressão for avaliada como verdadeira, o fluxo de dados é desviado para a parte verdadeira do construtor. Caso contrário, o fluxo é enviado para a parte falsa. A princípio, todo o fluxo de entrada é disponibilizado tanto na saída da parte verdadeira quanto na saída da parte falsa, ficando a cargo do usuário a conexão destes fluxos.

O operador condicional possui vários pontos de entrada. Para cada ponto de entrada definido, será gerada uma porta de saída na parte verdadeira e na parte falsa do construtor. Uma expressão booleana avalia se o fluxo de dados da entrada deve seguir pela porta verdadeira ou falsa do operador. Algumas portas de saída podem permanecer desconectadas. A Fig. 19 retrata o pictograma deste construtor:

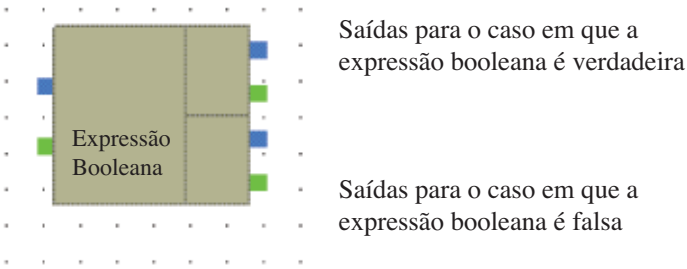


Fig. 19. Construtor de decisão.

O autor deste trabalho acredita que esta notação é mais condizente com os princípios de fluxo de dados do que a maioria das notações utilizadas nas linguagens visuais de programação. A razão para tal afirmação é o fato de que esta notação não interrompe o fluxo de dados, apenas o redireciona em função de uma avaliação booleana.

O operador de decisão pode ser utilizado para selecionar elementos, linhas ou colunas de uma matriz que satisfaçam algum critério definido pelo usuário. Dentre os critérios possíveis, tem-se a separação de uma submatriz cujos valores obedeçam a algum critério pré-definido ou, ainda, a obtenção de uma nova matriz a partir de um conjunto de índices. A funcionalidade do operador de seleção é obtida com a aplicação do operador condicional dentro de uma matriz, visando a seleção dos valores a serem escolhidos da matriz. Neste caso, apenas um ponto de saída é permitido em cada parte do construtor.

O exemplo da Fig. 20 reflete a criação de uma nova matriz, cujos valores foram obtidos pela seleção de elementos maiores que zero de uma matriz de entrada. Caso a expressão booleana seja falsa, o valor selecionado será a constante *0*.

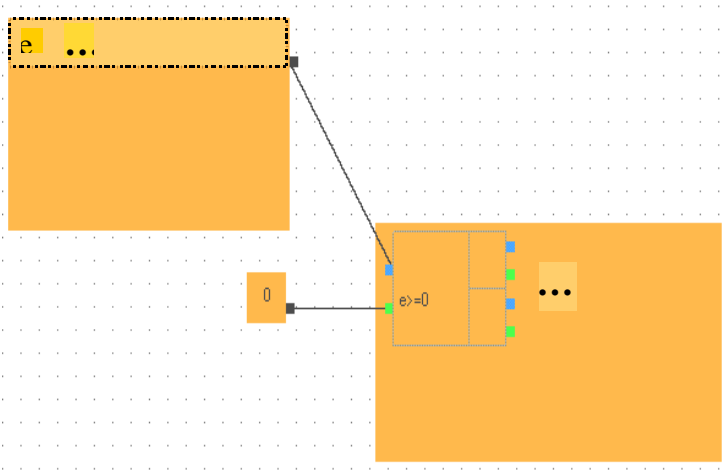


Fig. 20. Seleção de elementos de uma matriz.

É bom salientar, entretanto, que a geração de uma nova matriz baseada na seleção de elementos poderia produzir uma matriz inválida, uma vez que uma possibilidade seria a produção de uma "matriz" com números diferentes de elementos em cada uma de suas linhas. Este problema é contornado pela restrição adotada pelo MVM, que obriga a existência de valores conectados na parte verdadeira e falsa do construtor. Esta restrição não se aplica para a seleção de toda uma linha ou coluna.

5.4.2 Partição

Como foi visto na Seção 5.1, uma matriz é composta por duas planilhas que permitem, respectivamente, a manipulação dos dados de entrada e a construção de visões apropriadas para processamento. A visão pode ser um simples elemento, uma submatriz (partição), um conjunto de submatrizes ou mesmo toda a matriz. Pode-se, ainda, preparar um processo iterativo de maneira que elementos ou submatrizes sejam obtidos e enviados pela saída, obedecendo-se a ciclos de um controle interno de repetição. Ou seja, o componente visão de uma matriz pode ser enxergado como um poderoso comando de repetição que tenta suprir parte da necessidade por um construtor de repetição, comum nas linguagens de programação imperativas. Todavia, a confecção de uma visão diferente dos dados de entrada é opcional. Se nada diferente for requerido, os dados de entrada serão enviados integralmente dentro de uma única matriz para os processos conectados à saída da planilha.

Uma matriz pode ser seccionada em áreas retangulares chamadas regiões, que não podem ser sobrepostas, na parte da planilha reservada para diferentes visões (Seção 5.1). A Fig. 21 retrata um exemplo de partição de uma matriz em seis distintas regiões. A partição de uma matriz ocorre de maneira hierárquica. Uma região pode ser dividida em regiões menores, que a cortam vertical e horizontalmente. Três cortes verticais e dois cortes horizontais foram realizados para produzir a partição da Fig. 21. Qualquer uma destas regiões pode ser subdividida em outras regiões utilizando-se o mesmo conceito.

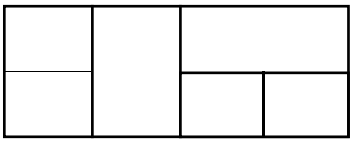


Fig. 21. Uma partição de uma matriz.

As dimensões destas partições podem ser informadas ou obtidas da mesma forma que a matriz da Fig. 22, isto é, utilizando-se as células de dimensionamento (quadrados externos desenhados nas bordas das regiões) para informar a dimensão de cada uma das partições efetuadas.

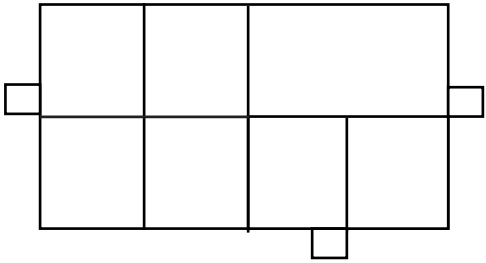


Fig. 22. Dimensionamento das partições.

Para partições complicadas, o usuário pode determinar se todas as células de dimensionamento serão mostradas ou não. Além disto, o sistema pode descobrir as dimensões de suas regiões a partir de dimensões já determinadas de regiões adjacentes. Observem que nem todas as regiões da Fig. 22 possuem células de dimensionamento. Ademais, as células de dimensionamento podem ser nomeadas para que seus valores possam ser utilizados em computações posteriores. Os valores associados às células de dimensionamento podem ser constantes, variáveis ou uma expressão aritmética. Os desenhos dos retângulos que representam as células de dimensionamento podem ser preenchidos com uma determinada cor para facilitar a associação de células de dimensionamento que devem obrigatoriamente possuir os mesmos valores. Adiante serão exemplificadas algumas partições, enquanto na Seção 5.5 ela será ilustrada em um exemplo prático.

Uma partição pode ser nomeada ou colorida para que possa ser referenciada pelos nós diretamente conectados a esta matriz no diagrama. Neste caso, o pictograma (matriz ou processo) pode conter uma expressão que utilize as cores ou nomes definidos anteriormente no diagrama. A Fig. 23 retrata a geração de uma nova matriz com os dados de quatro partições de outra matriz. Estas quatro regiões são recebidas pela matriz da direita, que trocará a ordem de duas delas (A1 e A4).

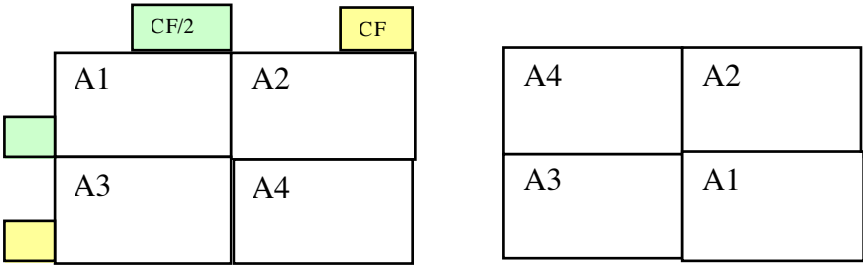


Fig. 23. Obtenção de quatro partições, todas com a mesma dimensão, denominadas A1, A2, A3 e A4.

Uma matriz pode conter, também, um padrão de iteração para que seja possível acessar os valores dos seus elementos. Este padrão pode incluir valores de determinados itens, os seus índices, as dimensões envolvidas e, primordialmente, a ordem segundo a qual ocorrerá a iteração na matriz.

Na Fig. 24 está representado um exemplo em que a matriz será percorrida da esquerda para direita e de cima para baixo. Neste exemplo, o padrão possibilitará o acesso a todos os itens da matriz (variável e) e ao número da linha correspondente (variável i) na ordem especificada.

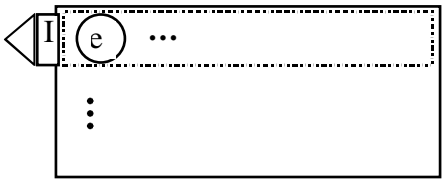


Fig. 24. Exemplo de padrão de iteração.

Exemplos de partições

Vale a pena exemplificar algumas partições de maneira a mostrar a força e a flexibilidade deste conceito do MVM. Como primeiro exemplo, considere-se a obtenção de quatro regiões, todas com a mesma dimensão, denominadas A1, A2, A3 e A4, que constam da Fig. 25. Estas quatro regiões são passadas para o processamento posterior.

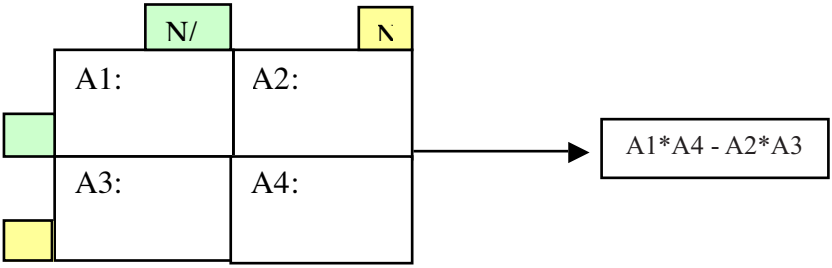


Fig. 25. Partição em quatro regiões.

A Fig. 26 ilustra um exemplo em que a partição anterior é preparada para obtenção dos elementos de cada região e para o posterior processamento destes elementos obtidos. Observe-se que, para cada região, os elementos serão lidos linha a linha.

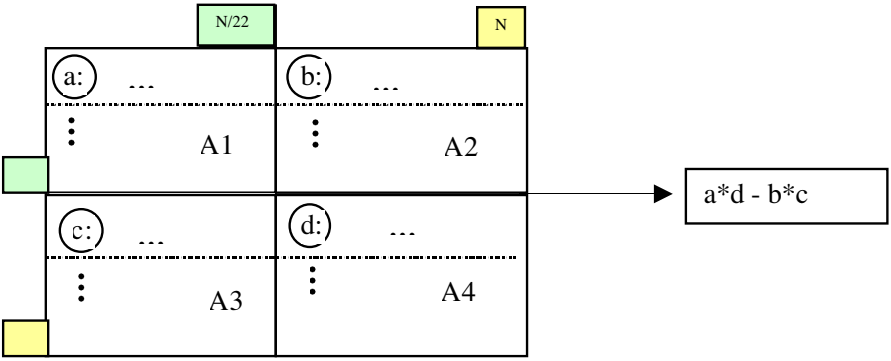


Fig. 26. Obtenção dos elementos de cada região.

Finalmente, o diagrama da Fig. 27 representa a passagem de cada uma das quatro regiões, uma por vez, para o próximo nó da rede.

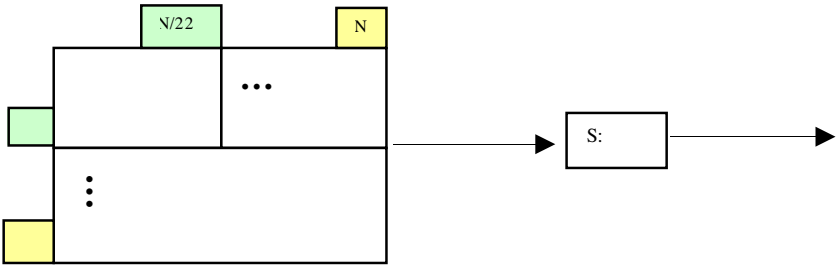


Fig. 27. Obtenção de uma região por vez.

Note-se a diferença entre as três implementações. A primeira criou quatro regiões, nomeou-as e passou todas para a expressão seguinte. A segunda dividiu a matriz em quatro regiões independentes, criou um padrão de iteração para cada uma e enviou quatro elementos, um de cada região, para o nó que a sucede. Este processo é repetido até que todos os elementos sejam enviados para o devido processamento. O último exemplo obtém cada região (uma por vez) de dimensão N/2 x N/2, copiando a região gerada para a matriz S. Vale a pena observar, ainda neste exemplo, que a matriz em questão é uma matriz quadrada, N x N, e que o sistema se encarrega de verificar se os dados de entrada são compatíveis com as dimensões esperadas.

5.5. Exemplo: Determinante de Uma Matriz

O exemplo da Fig. 28 retrata a resolução em MVM para o cálculo do determinante de uma matriz. A solução foi dividida em três diferentes casos de execução. O primeiro refere-se a uma matriz de entrada com dimensão (coluna ou linha) maior que 1. Nesta situação, a matriz é subdividida com a utilização do recurso de partição de matrizes. Cada submatriz é obtida pela eliminação da primeira linha e da coluna correspondente à posição do elemento "e:" no processo iterativo. As submatrizes S1 e S2 são enviadas para o próximo nó do diagrama, que consta agora com o recurso de geração de matriz. Os elementos desta nova matriz ("M") são computados a partir da fórmula ali especificada, que contém uma chamada recursiva para a função determinante, todavia a função determinante receberá uma matriz formada pela concatenação vertical das submatrizes S1 e S2. O resultado final será obtido pela soma de todos elementos contidos na matriz "M". O segundo caso retorna o valor da única célula da matriz de entrada cuja dimensão é unitária. Finalmente, o terceiro caso tem como resultado o valor 1 para uma matriz de dimensão zero.

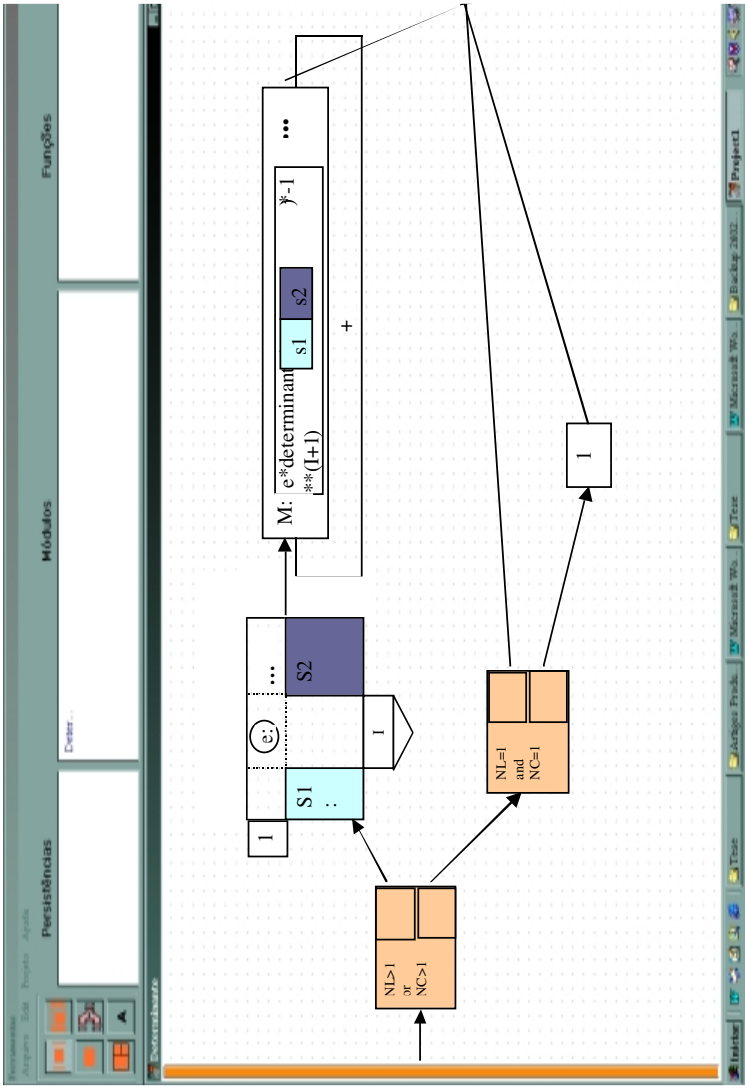
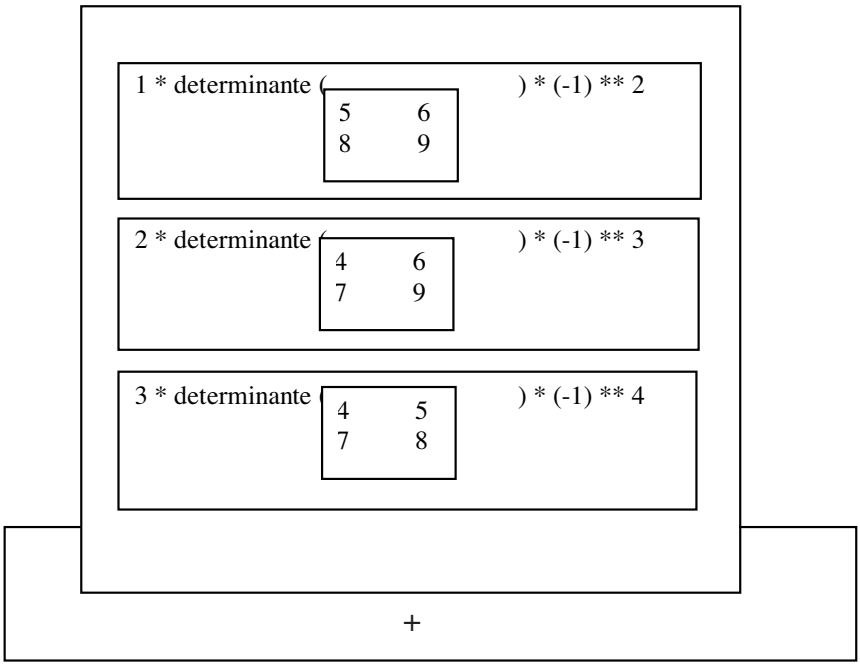


Fig. 28. Determinante de uma matriz. As variáveis rotuladas de NL e NC são internas de cada matriz e representam o número de linhas e colunas da matriz.

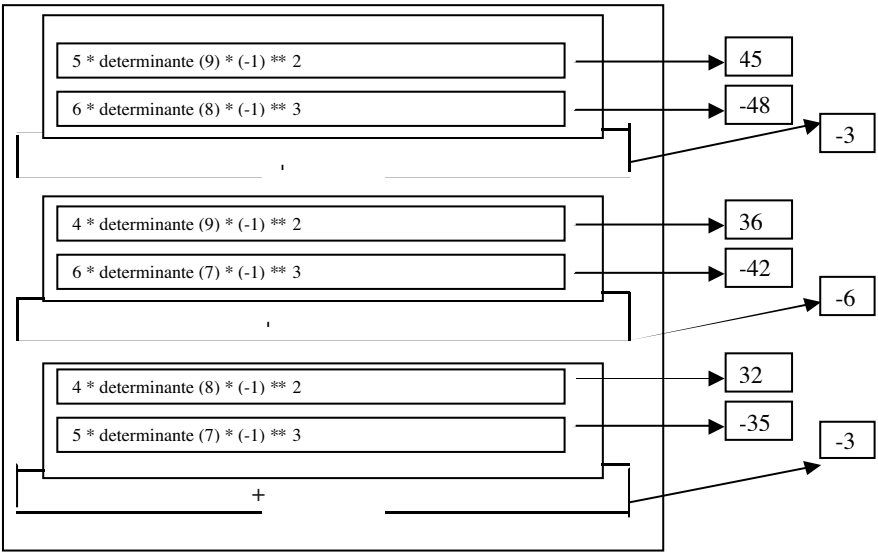
Considere a execução do programa da Fig. 28 para a seguinte matriz.

1	2	3
4	5	6
7	8	9

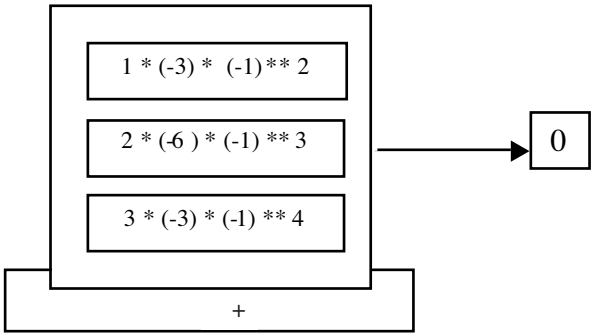
Como é uma matriz com três colunas e três linhas, o primeiro caso de execução será ativado, gerando, inicialmente, três computações que podem ser executadas em paralelo.



O passo seguinte é a chamada recursiva do determinante das três computações acima, gerando seis expressões para serem executadas.



Os resultados apresentados retornam para os respectivos locais das chamadas recursivas do primeiro ciclo da computação, produzindo o seguinte resultado.



Após as três expressões serem executadas e somadas, tem-se o resultado do determinante representado por um escalar de valor zero.

Vale a pena salientar que o ponto chave deste exemplo é a segmentação da matriz em diversas submatrizes dentro de um processo iterativo. Esta segmentação permitiu que o programa obtido fosse compacto e sem perder em expressividade, pois é fácil verificar que o determinante da matriz é obtido pelo pivotamento da matriz de entrada.

6. Conclusão

Foram discutidos neste trabalho alguns aspectos fundamentais relacionados às linguagens de programação visual. Em primeiro lugar, buscou-se uma definição precisa para o termo programação visual. Segundo, listou-se os paradigmas mais utilizados pela comunidade científica para descrever a base semântica das linguagens visuais. Finalmente, foi apresentado um modelo para programação visual de matrizes que traz algumas contribuições importantes para trabalhos na área.

Uma destas contribuições é a utilização do conceito de planilha para manipular uma matriz sem a necessidade de recorrer a uma notação textual. A representação visual proposta tende a ser mais fácil de ser entendida e manipulada por usuários não treinados em programação. Ao conceito de planilha foi acrescentado a programação por demonstração, a visualização transitória dos fluxos de dados existentes entre as várias fórmulas da planilha, a partição de matrizes e um construtor de iteração.

A programação por demonstração facilita a tarefa de implementação de algoritmos matriciais sem a necessidade, em muitos casos, de se recorrer a construtores usuais de uma linguagem de programação. Neste sentido, foi possível demonstrar que uma planilha pode ser utilizada como um procedimento. Ademais, um procedimento implementado tem baixa complexidade de entendimento por parte do usuário final.

A técnica de visualização transitória, por sua vez, foi baseada no trabalho de Igarashi & Mackinlay (1998) e permite que o usuário veja parte do fluxo de dados associado com a fórmula que se está interagindo. Ela mostra tanto o fluxo das células que afetam a fórmula como as células que são afetadas por esta fórmula.

Finalmente, uma matriz pode ser particionada em várias regiões independentes. Estas regiões podem ser utilizadas para descrever outras matrizes ou podem ser usadas em outros processamentos ou fórmulas. A partição pode ser um simples elemento, uma submatriz ou um conjunto de submatrizes. Estas partições podem ser enviadas para outros processos dentro de um processo iterativo, obedecendo-se a ciclos de um controle interno de repetição. Ou seja, a planilha embute um poderoso comando de repetição.

Entretanto, a principal contribuição deste trabalho é o estabelecimento de um modelo híbrido de programação visual de matrizes baseado em fluxo de dados, planilhas eletrônicas e programação por demonstração, pois possibilitam que uma vasta gama de aplicações possam ser implementadas com redução na utilização de construtores computacionais relacionados à repetição e à recursão. Desta forma, a intuição é que programas de domínio matricial possam ser construídos de maneira mais fácil e com um nível de abstração mais alto. Vale ressaltar, entretanto, que esta suposição precisa ser validada por um experimento a ser realizado que compare a utilização e entendimento de programas obtidos com o modelo MVM com outras linguagens textuais e visuais do mesmo domínio de aplicação.

O MVM foi validado com uma implementação na linguagem Delphi e utiliza a biblioteca OpenGL para dar suporte gráfico ao editor visual. O Delphi foi escolhido pelo rico conjunto de recursos que oferece, tornando a tarefa de desenvolvimento mais fácil e rápida. Ele é voltado para criação de interfaces gráficas de usuário (GUI – Graphic User Interface), em contraponto às linguagens clássicas orientadas a objetos (por exemplo, C++). O Delphi oferece, ainda, controle de exceções e gerenciamento de *threads*, fundamentais para implementação da máquina virtual de execução.

O estágio atual da implementação inclui um editor visual de programa, um editor de matrizes e uma máquina para execução de fluxo de dados. Os editores de programa e de matrizes são totalmente dirigidos pela sintaxe, isto é, o processo de edição é dirigida pela estrutura de um programa MVM. Ademais, foram impostas algumas regras adicionais para forçar a construção de programas sintaticamente válidos.

7. Referências Bibliográficas

AMBLER, A. Forms: expanding the visualness of sheet languages. In: WORKSHOP ON VISUAL LANGUAGES, 1987, Linkoping. Proceedings...[Linkoping: s. n., 1987]. p. 105-117.

AMBLER, A.; BURNETT, M. Influence of visual technology on the evolution of language environments. Computer, v. 22, n. 10, p. 9-22, Oct. 1989.

AMBLER, A.; GREEN, T.; KIMURA, T.; REPENNING, A.; SMEDLEY, T. Visual programming challenge summary. In: SYMPOSIUM ON VISUAL LANGUAGES, 1997, Capri. Proceedings ... [Capri: s. n., 1997].

BAROTH, E.; HARTSOUGH, C. Visual programming in the real world. In: BURNETT, M.; GOLDBERG, A.; LEWIS, T. (Ed.). Visual object-oriented programming: concepts and environments. Greenwich: Manning Publication, 1995. Cap. 2, p. 21-42.

BROWN, T.; KIMURA, T. Completeness of a visual computation model. Software - Concepts and Tools, v. 15, p. 34-48, 1994.

BURNETT, M. Visual programming: encyclopedia of electrical and electronics engineering. New York: John Wiley, 1999.

BURNETT, M.; AMBLER, A. Interactive visual data abstraction in a declarative visual programming language. Journal of Visual Languages and Computing, v.5, n. 1, p.29-60, Mar. 1994.

BURNETT, M.; BAKER, M.; BOHUS, C.; CARLSON, P.; YANG, S.; ZEE, P. van. Scaling up visual programming languages. Computer, v. 28, n. 3, p. 44-54, Mar. 1995.

ERWIG, M.; MEYER, B. Heterogeneous visual languages – integrating visual and textual programming. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 11., 1995, Darmstadt. Proceedings... [Los Alamitos: IEEE Computer, 1995].

GLINERT, E. P.; TANIMOTO, S. L. Pict: an interactive graphical programming environment. Computer, v. 17, n. 11, p. 7-25, Nov. 1984.

GOULD, L.; FINZER, W. Programming by rehearsal. Byte, v. 9, n. 6, p. 187-210, June, 1984.

GRAF, H. W.; NEUROHR, S. Constraint-based layout in visual program design. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 11., 1995, Darmstadt. Proceedings... [Los Alamitos: IEEE Computer, 1995]. p. 116-117.

GREEN, T.; PETRE, M. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. Journal of Visual Languages and Computing, v. 7, n. 2, p. 131-174, June, 1996.

HAREL, D. On visual formalisms. Communications of the ACM, v. 31, n. 5, p. 514-530, 1988.

HENDRY, D. G. Display-based problems in spreadsheets: a critical incident and a design remedy. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 11., 1995, Darmstadt. Proceedings... [Los Alamitos: IEEE Computer, 1995]. p. 284-290.

HILS, D. Visual languages and computing survey: data flow visual programming languages. Journal of Visual Languages and Computing, v. 3, p. 69-101, 1992.

HIRAKAWA, M. A framework for construction of icon system. In: IEEE WOKSHOP ON VISUAL LANGUAGES, 1988, Pittsburgh. Proceedings... [Los Alamitos: IEEE Computer, l988. p. 45-51.

IGARASHI, T.; MACKINLAY, D. Fluid visualization of spreadsheet structures. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 1998, Halifax, Nova Scotia. Proceedings.... [Los Alamitos: IEEE Computer, 1998].

IVERSON, K. E.; FALKOFF, A. The design of APL. IBM Journal of Research and Development, v. 17, n. 5, p. 324-334, July, 1973.

JAFFAR, J.; LASSEZ, J. Constraint logic programming. In: ACM SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, 14., 1987, Munich. Conference record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages: papers presented at the symposium... New York: The Association; Baltimore: ACM, 1987.

KAJLER, N.; SOIFFER, N. Survey of user interfaces for computer algebra systems. Journal of Symbolic Computation, v. 11, p. 1-33, 1995.

KIMURA, T. D. Hyperflow: an uniform visual language for different levels of programming. In: ACM CONFERENCE ON COMPUTER SCIENCE (CSC 93), 1993, Indianapolis. Proceedings... [New York: ACM, 1993]. 13 p.

LEOPOLD, J.; AMBLER, A. L An users interface for the visualization and manipulation of arrays. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 12., Boulder, 1996. Proceedings... [Los Alamitos: IEEE Computer, 1996]. p. 54-55.

MARRIOTT, K.; MEYER, B. Visual language theory. New York: Springer-Verlag, 1998. 381 p.

MATWIN, S.; PIETRZYKOWSKI, T. Prograph: a preliminary report. Computer Language, v. 10, p. 91-126, 1985.

MYERS, B. Visual programming, programming by example and program visualization: a taxonomy. In: COMPUTER HUMAN INTERACTION (CHI 86) - HUMAN FACTORS IN COMPUTING SYSTEMS, 1986, Boston. Conference proceedings... [Boston: s. n.: 1986]. p. 59-66.

SHU, N. C. Principles of visual programming: systems Shi-Kuo Chang. New York: Van Nostrand Reinhold; Englewood Cliffs: Prentice Hall, 1988a.

SHU, N. C. Visual programming. New York: Van Nostrand Reinhold, 1988b. 315 p.

SMITH, D. KidSim: programming agents without a programming language. Communications of the ACM, v. 37, n. 7, July, 1994.

SHNEIDERMAN, B. Direct manipulation: a step beyond programming language. Computer, v. 16, n. 80, p. 57-69, Aug. 1983.

TRIPP, L. A survey of graphical notations for program design: an update. ACM SIGSOFT Software Engineering Notes, v. 13, n. 4, p. 39-44, Apr. 1988.

VOSE, G.; WILLIAMS, G. LabView: Laboratory Virtual Instrument Engineering Workbench. Byte, v. 11, n. 9, p. 84-96, Sept. 1986.

WHITLEY, K. N. Empirical research of visual programming languages: an experiment testing the comprehensibility of LabVIEW. 2000. 321 f. Dissertation (PhD) - Graduate School of Vanderbilt University, Nashville.

YEUNG, R. MPL - A graphical programming environment for matrix processing based on logic and constraints. In: IEEE WORKSHOP OF VISUAL LANGUAGES, 1988. Proceedings... [Los Alamitos: IEEE Computer, 1988]. p. 137-143.

Informática Agropecuária

MINISTÉRIO DA AGRICULTURA,
PECUÁRIA E ABASTECIMENTO