UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO DEPARTAMENTO DE ELETRÔNICA E MICROELETRÔNICA

Uma Ferramenta Automatizada para Análise e Projeto de Circuitos Digitais Multi-Valores

Luciana Prado do Nascimento

Campinas, abril de 2001

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO DEPARTAMENTO DE ELETRÔNICA E MICROELETRÔNICA

DISSERTAÇÃO DE MESTRADO

Uma Ferramenta Automatizada para Análise e Projeto de Circuitos Digitais Multi-Valores

Luciana Prado do Nascimento

Orientador: **Alberto Martins Jorge**

Banca Examinadora:

Prof. Dr. Alberto Martins Jorge, Presidente - FEEC/UNICAMP
Prof. Dr. Elnatan Chagas Ferreira - FEEC/UNICAMP
Prof. Dr. José Antonio Siqueira Dias - FEEC/UNICAMP
Profª. Dra. Maria Nídia Ramos D. Yacoub - UNISAL

Dissertação apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte do requisito exigido para obtenção do título de Mestre em Engenharia Elétrica.

Abril - 2001

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

N17f

Nascimento, Luciana Prado do
Uma ferramenta automatizada para análise e projeto de circuitos digitais multi-valores / Luciana Prado do Nascimento. --Campinas, SP: [s.n.], 2001.

Orientador: Alberto Martins Jorge.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Lógica a múltiplos valores. 2. Projeto lógico. 3. Circuitos lógicos. 4. Projeto auxiliado por computador. I. Jorge, Alberto Martins. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

À Marília, por seu maravilhoso sorriso e a toda a minha família.

Agradecimentos

Ao Prof. Dr. Alberto Martins Jorge pela maravilhosa orientação, pelo cuidado nas críticas, pela paciência e dedicação sempre presentes.

Ao Prof. Dr. José Antonio Siqueira Dias por ter desenvolvido com o meu orientador a base conceitual para a realização deste trabalho.

Ao professor Dr. Elnatan Chagas Ferreira, do Departamento de Eletrônica e Microeletrônica da Faculdade de Engenharia Elétrica e de Computação, pelas valiosas sugestões ao trabalho.

Ao meu amigo e namorado, Paulo Henrique Machado, pela enorme paciência, colaboração e ajuda sempre que necessária.

Aos meus pais, José Roberto e Catarina, que tanto sempre fizeram para que eu pudesse concluir esse e tantos outros trabalhos na minha vida.

A todos os amigos e colegas do Departamento, em especial Alcino Biazon Filho, Carlos Cavalcanti, Fernando Ferreira de Melo, Jonatas Fred Rossetto, José Carlos da Silva, Reinaldo Blas e Ademilde Félix, secretária do DEMIC, pelas ótimas idéias e pelos momentos de descontração.

Resumo

No desenvolvimento de projetos de circuitos lógicos Multi-valores, é necessário obter o resultado de expressões lógicas muito extensas e complexas. Algumas vezes precisamos comparar o resultado de funções lógicas desenvolvidas por métodos diferentes e esse processo pode se tornar exaustivo.

Uma solução apresentada é uma ferramenta automatizada para a obtenção da tabela-verdade dada uma função lógica multi-valor. O programa é capaz de calcular a tabela-verdade para expressões lógicas ternárias e quaternárias com até três variáveis de entrada.

A utilização do programa em projetos de circuitos ternários confirma a necessidade de tal ferramenta. Com o auxílio do programa um teorema na lógica multi-valores que foi comprovado, é apresentado neste trabalho.

A álgebra Multi-valores não-clássica, baseada na álgebra de Post, a qual foi utilizada neste trabalho é mostrada. Assim como os operadores e a síntese de funções lógicas.

O programa mostrou-se uma ferramenta amigável superando as expectativas em velocidade e facilidade. E poderá ser usado em trabalhos futuros, onde a síntese e a simplificação de funções lógicas é feita.

Palavras-Chave: Lógica Multi-valores, Álgebra de Post, circuitos digitais, circuitos ternários e quaternários, análise e projeto de MVL.

Abstract

In the development of logical circuits projects in Multivalued Logic (MVL), it is necessary to obtain the result of logical expressions that sometimes are very large and complex. Sometimes, we need to compare the results of different logical functions and, this process can be exhaustive.

The presented solution is an automated tool to obtain a true-table to a given logical function. The software is capable of calculating the truth-table for expressions in ternary or quaternary levels for up to three input variables.

The use of the program in a design of ternary logic circuit confirms its necessity. Making use of the program led to a formulation of a theorem, which is presented in this work.

The non-classical algebra of MVL based on Post algebra, used in this work is shown, as well as the operators and the synthesis of the logical functions.

The software proved to be a feasible tool, surpassing the expectations of velocity and facility. This software is a tool that can be used in a future work in which synthesis and simplification of logical functions will be performed.

Key Words: Multivalued logic, Post algebra, ternary and quaternary digital circuits, analysis and design of MVL.

CONTEÚDO

CAPÍTULO 1

LÓGICA MULTI-VALORES E MÉTODO DOS OPERADORES

1.1 Introdução

A lógica Multi-valores (MVL) foi apresentada pela primeira vez por Lukasiewicz [10] no ano de 1920. No ano seguinte, Post [6] descreveu uma álgebra funcionalmente completa com uma operação de mínimo (AND) e deslocamentos para as variáveis criando uma lógica cíclica, na qual a lógica Multi-valores apresentada neste trabalho foi baseada. Epstein [8], em 1960, realizou várias pesquisas sobre MVL, onde vários artigos descrevem a lógica, sua funcionalidade e uma nova síntese para funções lógicas com n valores, utilizando dois operadores (AND e OR) e funções auxiliares. Muitos pesquisadores começaram a explorar essa álgebra. Até os anos 70, houve muitas pesquisas na área, inclusive a emulação de um computador ternário, que tinha grande compatibilidade em termos de preço e velocidade com os computadores binários. Com o desenvolvimento em massa de circuitos binários, a utilização de circuitos ternários foi deixada para trás. Mas as pesquisas continuaram e até hoje encontramos artigos explorando a utilização de circuitos ternários e quaternários.

Recentemente, Thoidis [9], desenvolveu circuitos quaternários para MVL utilizando a tecnologia CMOS. Um circuito ternário utilizando uma porta CMOS universal foi projetado e testado por Biazon [3], mostrando grande eficiência e viabilidade.

CAPÍTULO 1 – LÓGICA MULTI-VALORES E MÉTODO DOS OPERADORES

MVL hoje é muito importante para a ciência da computação; deu origem à lógica fuzzy e é utilizada em inteligência artificial. Olmsted [5] descreve brevemente a história da MVL e a origem da lógica fuzzy.

Na área de inteligência artificial, a MVL está sendo muito pesquisada e seus conceitos estão sendo continuamente utilizados. Cardoli [14] faz uma explanação de como a MVL pode ser uma ferramenta para gerar bases de conhecimento em projetos de inteligência artificial.

Explorando a MVL e a álgebra proposta por Serran [15], foi verificada a necessidade de automatizar o processo de solucionar expressões lógicas, de forma que o trabalho se tornasse menos exaustivo e mais confiável. A solução de expressões lógicas é necessária no processo de síntese de funções lógicas, onde temos expressões muito complexas e difíceis de serem simplificadas. No processo de circuitos digitais ternários e quaternários podemos ter em algumas saídas, expressões que precisam ser calculadas e/ou comparadas a algum valor esperado.

Um programa foi projetado e desenvolvido para realizar o cálculo das expressões lógicas multi-valores. Neste trabalho apresentamos todo o estudo, projeto e apresentação dessa ferramenta automatizada que obtém a saída de expressões lógicas nos níveis ternário e quaternário para até três variáveis de entrada.

No Capítulo 1 descrevemos as álgebras de Post e a Lógica Multi-valores utilizada no trabalho assim como a síntese de funções lógicas utilizando-se o Método dos Operadores e o Método proposto por Epstein [8]. O Capítulo 2 mostra a necessidade da ferramenta apresentada neste trabalho. No Capítulo 3 é feita uma descrição completa de como o software apresentado neste trabalho gera as tabelas de

saída, apresentando os algoritmos principais e exemplos. Também são apresentadas as telas de entrada e saída do programa. O Capítulo 4 apresenta um manual completo para a utilização do programa e um arquivo de ajuda.

1.2 Exposição da Álgebra de Post e Lógica Multi-valores utilizando o Método dos Operadores

Uma apresentação breve da álgebra utilizada no trabalho é exposta. A álgebra contém um conjunto de variáveis (x, y, z,...), as quais assumem *n* valores lógicos de um conjunto definido dado $P = \{t_1, t_2, ..., t_n\}$, onde $t_i > t_j$, se $i > j$.

Temos uma rotação cíclica, que segue a negação cíclica de Post. Para nível ternário, com *n* igual a três, verificamos dois deslocamentos:

- deslocamento unitário, o qual chamaremos de topo, também utilizado no nível quaternário, como mostra a equação 1.1.

$$\overline{t_i} = \begin{cases} t_i + 1, & \text{se } i \neq n \\ t_1, & \text{se } i = n \end{cases} \qquad (1.1)$$

- deslocamento para a esquerda, chamado de base, mostrado por 1.2.

$$\underline{t_i} = \begin{cases} t_i - 1, & \text{se } i \neq n \\ t_n, & \text{se } i = 1 \end{cases} \qquad (1.2)$$

Para nível quaternário temos os seguintes deslocamentos:

- deslocamento unitário, topo: idêntico ao nível ternário.

- deslocamento duplo, chamado de duplo topo, como mostra a equação 1.3.

$$\underset{t_i}{=} \begin{cases} t_i + 2, & \text{se } i < 2 \\ t_i - 2, & \text{se } i > 2 \end{cases} \tag{1.3}$$

- deslocamento para a esquerda, base, equivalente ao deslocamento para a esquerda (base) para nível ternário.

A conjunção binária AND $(\cdot)$, como na álgebra clássica, é dada pela equação 1.4.

$$t_i \cdot t_j = \min\{t_i, t_j\} \tag{1.4}$$

Essa operação foi chamada de operação α. E é dada pela tabela 1.1.

A α B		**B**			
		0	**1**	**2**	**3**
A	**0**	0	0	0	0
	1	0	1	1	1
	2	0	1	2	2
	3	0	1	2	3

Tabela 1.1: Operação α

Outras operações foram definidas, fazendo um deslocamento na operação de mínimo (operador α). As outras operações, mostradas nas tabelas 1.2, 1.3 e 1.4, são: β, γ, e δ , esta apenas se nível for quaternário.

Para cada operação temos um elemento principal, um secundário, um ternário e um indiferente. Esses elementos possuem diferentes níveis de prioridade, como podemos observar:

- elemento principal: maior nível;

- elemento secundário: um nível abaixo do elemento principal;

- elemento ternário: um nível abaixo do elemento secundário;

- elemento indiferente: menor nível.

A operação β é um deslocamento unitário da operação α. Tem como elemento principal o valor 1, como secundário o valor 2, como elemento ternário o valor 3 e como elemento indiferente o valor 0.

A β B		**B**			
		0	**1**	**2**	**3**
A	**0**	0	1	2	3
	1	1	1	1	1
	2	2	1	2	2
	3	3	1	2	3

Tabela 1.2: Operação β

A operação γ sofre um deslocamento duplo em relação à operação α. Os valores referentes aos elementos principal, secundário, ternário e indiferente são 2, 3, 0 e 1, respectivamente.

A γ B		B			
		0	**1**	**2**	**3**
A	**0**	0	0	2	3
	1	0	1	2	3
	2	2	2	2	2
	3	3	3	2	3

Tabela 1.3: Operação γ

A operação δ, aparece apenas se nível for quaternário, pode ser vista como deslocamento triplo a operação α. Seus valores correspondentes aos elementos principal, secundário, ternário e indiferente são, respectivamente: 3, 0, 1 e 2.

A δ B		B			
		0	**1**	**2**	**3**
A	**0**	0	0	0	3
	1	0	1	1	3
	2	0	1	2	3
	3	3	3	3	3

Tabela 1.4: Operação δ

É possível mostrar que qualquer função multi-valores é dedutível às operações α e topo (MÍNIMO e NEGAÇÃO), portanto a álgebra é funcionalmente completa.

1.3 Síntese de Funções Digitais utilizando o Método dos Operadores

Uma função de variáveis digitais é uma correspondência entre os elementos de um conjunto A, em relação aos elementos do conjunto básico que é definido dentro da álgebra em questão. Assim temos que: $\mathbf{A(x_i) = y_i}$

Os elementos de y_i são os do mesmo conjunto de x_i, podendo conter todos ou nem todos os valores de x_i.

O processo de síntese é a obtenção, a partir de uma tabela, de uma expressão de uma função em uma ou mais variáveis usando, nesta expressão, as operações definidas na álgebra proposta.

1.3.1 Síntese de Funções Digitais em Quatro Níveis

Para demonstrarmos o procedimento de síntese de funções digitais utilizaremos um exemplo, de forma que fique mais clara a sua compreensão.

A função C de duas variáveis, A e B, é definida pela tabela 1.5.

C		B			
		0	**1**	**2**	**3**
A	**0**	0	0	0	0
	1	0	0	1	1
	2	0	2	2	2
	3	3	3	3	3

Tabela 1.5 Função exemplo C

O processo de síntese de funções lógicas multi-valores foi obtido através da verificação e comparação com a síntese de funções lógicas binárias. Para isso, foram criadas funções intermediárias onde são analisados apenas dois valores de cada vez.

Para iniciar a síntese da expressão algébrica de C(A,B) temos que criar uma função intermediária, chamada C_0(A,B), mostrada na tabela 1.6, definida apenas pelos termos 0 e 1, sendo que os 0´s e 1´s são conservados, e no lugar dos valores 2 e 3 aparece o valor indiferente X, que pode ser substituído pelo valor 0 ou 1, dependendo da forma mais conveniente para a simplificação da expressão. Como o objetivo principal deste trabalho não é a simplificação de expressões lógicas não entraremos em maiores detalhes sobre a substituição do valor indiferente.

C_0		B			
		0	**1**	**2**	**3**
A	**0**	0	0	0	0
	1	0	0	1	1
	2	0	X	X	X
	3	X	X	X	X

Tabela 1.6 Função intermediária C_0

Esta estrutura faz com que a função C_0 possa ter a sua representação por uma fórmula em que apareçam somente as operações α e β, juntamente com as de complementação. Na expressão foram selecionados os pontos em que a função vale 0, e usando as funções A e B de maneira que seus valores se tornem 0 nessa posição, operados por β. Os produtos parciais são combinados por meio da operação α, de

forma que todos os valores 0 e 1 da função inicial sejam selecionados. A expressão para C_0 é a dada pela equação 1.5.

$$C_0 = A\alpha(\underline{A}\beta B)\alpha(\underline{A}\beta\underline{B})\alpha(\overline{\overline{A}}\beta B)\alpha 1 \qquad (1.5)$$

O termo $\alpha 1$ no final da expressão é utilizado para que só tenham valores 0´s e 1´s na tabela de C_0, já que o valor 1 é o valor secundário da operação α.

Agora temos que gerar uma segunda função intermediária C_1, dada pela tabela 1.7, conservando os valores originais de 1´s e 2´s de C(A,B), onde tínhamos os valores 0´s colocamos o valor 1 e aonde tínhamos os valores 3´s colocamos o termo X (indiferente).

C_1		B			
		0	**1**	**2**	**3**
A	**0**	1	1	1	1
	1	1	1	1	1
	2	1	2	2	2
	3	X	X	X	X

Tabela 1.7 Função intermediária C_1

Usando o mesmo processo de síntese para C_0, com as operações β conectando os produtos parciais e a operação γ nas combinações de A e B nos pontos em que C_1 tem o valor 1. Assim temos a seguinte expressão, dada por 1.6, para C_1.

$$C_1 = A\beta\overline{A}\beta(\underline{A}\gamma\overline{B})\beta 2 \qquad (1.6)$$

O termo β2 no final da expressão serve para confirmar que os demais termos da tabela terão o valor 2.

Uma terceira função intermediária é agora gerada, C_2, mostrada na tabela 1.8, onde analisaremos apenas os valores 2´s e 3´s da função C(A,B). Os valores originais de 2´s e 3´s são conservados e aonde tínhamos os valores 0´s e 1´s temos o valor 2.

C_2		B			
		0	**1**	**2**	**3**
A	**0**	2	2	2	2
	1	2	2	2	2
	2	2	2	2	2
	3	3	3	3	3

Tabela 1.8 Função intermediária C_2

Utilizamos a operação δ para combinarmos as funções de A e B onde seus valores são 2 e a operação γ para conectar esses produtos parciais. Então temos a expressão dada por 1.7 para C_2.

$$C_2 = A\gamma\bar{A}\gamma\bar{\bar{A}}\gamma 3 \tag{1.7}$$

Finalmente encontramos as três funções intermediárias para C(A,B). A função final de C, vista em 1.8, será a união das funções intermediárias C_0 e C_1 pela operação γ, onde obtemos os valores 0, 1 e 2 da função inicial e pela operação δ com a função C_2, onde obtemos os valores 3.

$$C = C_0\gamma C_1\delta C_2 \tag{1.8}$$

1.4 Simplificações de funções lógicas multi-valores

Na lógica proposta, algumas simplificações foram deduzidas e podem ser observadas no trabalho de Serran [16]. Neste trabalho duas novas simplificações são deduzidas.

1.4.1 Lei de Absorção

A Lei de Absorção pode ser deduzida através da verificação mostrada em 1.9. Utilizando os teoremas de Post, conseguimos demonstrar que a lei de absorção é válida para a lógica multi-valores.

$$A\delta(A\gamma(A\beta(A\alpha X))) = A\delta(A\gamma((A\alpha 2)\beta(A\alpha X))) = A\delta(A\gamma(A\alpha(2\beta X)))$$

$$= \mathbf{A\delta(A\gamma A)\alpha(A\gamma(2\beta X)) = A\delta(A\alpha(A\gamma(2\beta X))) = A\delta(A\gamma 1)\alpha(A\gamma(2\beta X))}$$

$$= \mathbf{A\delta(A\gamma(1\alpha(2\beta X))) = A\delta(A\gamma((1\alpha 2)\beta(1\alpha X)) = A\delta(A\gamma(1\beta(1\alpha X)))}$$

$$= \mathbf{A\delta(A\gamma 1) = A\delta A = A} \qquad (1.9)$$

Para três valores lógicos temos a expressão dada em 1.10.

$$\mathbf{A\alpha(A\beta(A\gamma X) = A} \qquad (1.10)$$

E para quatro valores lógicos temos a expressão dada em 1.11.

$$\mathbf{A\alpha(A\beta(A\gamma(A\delta X) = A} \qquad (1.11)$$

1.4.2 Teorema

No processo de síntese para uma determinada função, foi confirmado que a expressão vista em 1.12 é sempre verdadeira. Neste trabalho é mostrado o porquê dessa veracidade.

$$\mathbf{A\alpha(A\beta\overline{A}) = A} \qquad (1.12)$$

Quando fazemos $\mathbf{A\beta\overline{A}}$ temos sempre como resultado os mesmos elementos de A, eliminando apenas o elemento indiferente da operação β, o valor 0. Quando efetuamos a operação Aα a esse resultado obtemos os mesmos valores para A, pois a operação α coloca o valor 0, seu valor principal, quando o próprio A vale 0.

1.5 Síntese utilizando o método Clássico de Epstein

Para fazer a síntese de uma função utilizando as regras de Post, Epstein [8] utilizou as regras usuais para as operações de Mínimo (AND), denotadas por $\wedge$ e Máximo (OR), denotadas por $\vee$, incluindo a distributividade de uma operação através de outra e os seguintes postulados para duas operações unárias C_0 e C_2, com três constantes e_0, e_1 e e_2, para nível ternário:

(H1) $\mathbf{e_0 \vee x = x} \qquad \mathbf{y \wedge e_2 = y}$

(H2)

(a) $\mathbf{C_0(x) \wedge C_2(x) = e_0, C_0(x) \wedge C_0(C_0(x)) = e_0}$

(b) $\mathbf{C_0(x) \vee [C_0(C_0(x)) \wedge C_0(C_2(x))] \vee C_2(x) = e_2}$

(H3)

(a) $\mathbf{C_2(x \vee y) = C_2(x) \vee C_2(y)}$

(b) $\mathbf{C_0(x \wedge y) = C_0(x) \vee C_0(y)}$

(H4) $\mathbf{C_0(e_0) = e_2, C_0(e_1) = e_0, C_2(e_1) = e_0, C_2(e_2) = e_2}$

(H5) $\mathbf{x = [e_1 \wedge C_0(C_0(x))] \vee C_2(x).}$

O postulado (H1) mostra que e_0 é o menor elemento e que e_2 é o maior elemento dos elementos dessa álgebra. Postulado (H2a) pode ser visto como uma generalização da lei de contradição. Postulado (H2b) pode ser visto como uma generalização da lei de exclusão mútua. Postulado (H3) mostra as regras de transmissão para C_2 através do operador $\vee$ e para C_0 através do operador $\wedge$. Postulado (H4) mostra a variação das regras. Postulado (H5) expressa cada elemento x da álgebra.

X	$C_0(x)$	$C_1(x)$	$C_2(x)$
e_0	e_2	e_0	e_0
e_1	e_0	e_2	e_0
e_2	e_0	e_0	e_2

Tabela 1.9 Regras de variação para operações quando n=3

Usando as operações $C_0(x)$, $C_1(x)$ e $C_2(x)$, mostradas na tabela 1.9, para n=3, a forma normal para qualquer função ternária é obtida utilizando-se os operadores $\wedge$ e $\vee$. Para duas variáveis u e v, temos a função generalizada f(u,v), definida em 1.13.

$$\mathbf{f(u,v) = [f(e_0,e_0) \wedge C_0(u) \wedge C_0(v)]}$$

$$\mathbf{\vee [f(e_0,e_1) \wedge C_0(u) \wedge C_1(v)]}$$

$$\vee[\mathbf{f}(\mathbf{e_0},\mathbf{e_2}) \wedge \mathbf{C_0}(\mathbf{u}) \wedge \mathbf{C_2}(\mathbf{v})]$$

$$\vee[\mathbf{f}(\mathbf{e_1},\mathbf{e_0}) \wedge \mathbf{C_1}(\mathbf{u}) \wedge \mathbf{C_0}(\mathbf{v})]$$

$$\vee[\mathbf{f}(\mathbf{e_1},\mathbf{e_1}) \wedge \mathbf{C_1}(\mathbf{u}) \wedge \mathbf{C_1}(\mathbf{v})]$$

$$\vee[\mathbf{f}(\mathbf{e_1},\mathbf{e_2}) \wedge \mathbf{C_1}(\mathbf{u}) \wedge \mathbf{C_2}(\mathbf{v})]$$

$$\vee[\mathbf{f}(\mathbf{e_2},\mathbf{e_0}) \wedge \mathbf{C_2}(\mathbf{u}) \wedge \mathbf{C_0}(\mathbf{v})]$$

$$\vee[\mathbf{f}(\mathbf{e_2},\mathbf{e_1}) \wedge \mathbf{C_2}(\mathbf{u}) \wedge \mathbf{C_1}(\mathbf{v})]$$

$$\vee[\mathbf{f}(\mathbf{e_2},\mathbf{e_2}) \wedge \mathbf{C_2}(\mathbf{u}) \wedge \mathbf{C_2}(\mathbf{v})] \qquad (1.13)$$

Utilizando o método clássico de Epstein, podemos trabalhar com apenas dois operadores, e com o método dos operadores teremos o mesmo número de operadores que n, nível no qual estamos trabalhando. A utilização de um número maior de operadores permite uma síntese mais simples, obtendo como expressão final, um resultado menor e mais compacto.

Como qualquer função na álgebra multi-valores pode ser dedutível a apenas duas operações básicas, nos projetos de circuitos integrados, temos a possibilidade de usar blocos para uma única operação como por exemplo αTopo ou δTopo, com a qual podemos projetar qualquer circuito. No método clássico de Epstein, ao projetar um circuito, devemos usar blocos para o operador de mínimo $\wedge$ e de máximo $\vee$, além de um circuito para as funções auxiliares de C_0, C_1 e C_2, para nível ternário e C_0, C_1, C_2 e C_3 para nível quaternário.

CAPÍTULO 2

DEFINIÇÕES E MOTIVAÇÕES PARA O PROJETO

2.1 Definições do Projeto

Diante do estudo da lógica da álgebra Multi-valores, percebemos a necessidade de uma ferramenta automatizada que auxiliasse no processo de síntese de funções lógicas multi-valores. Uma ferramenta na qual o usuário pudesse entrar com uma expressão lógica multi-valores e que retornasse o resultado dessa expressão na forma de tabela.

Geralmente no processo de síntese nos deparamos com expressões complexas e extensas e desejamos saber se essa expressão possui uma simplificação. Como o processo de simplificação para funções multi-valores é ainda muito complexo, uma forma de obtermos expressões simplificadas é através do resultado da expressão. Assim o usuário poderá analisar a tabela e obter uma expressão mais simples.

Nos projetos de circuitos digitais multi-valores, a ferramenta proposta pode ser útil, pois algumas vezes é necessário saber qual será a saída em umas das portas internas ou externas do circuito. A saída dessas portas pode ser expressa por funções demasiadamente complexas, e o resultado obtido, calculado manualmente, pode conter alguns erros. Utilizando a ferramenta automatizada, o usuário poderá se sentir mais seguro, contando com a credibilidade do resultado esperado para projetar o seu circuito.

CAPÍTULO 2 – DESCRIÇÃO E MOTIVAÇÕES PARA O PROJETO

Para realizar tal ferramenta um estudo aprofundado de cálculos de expressões multi-valores foi feito. O princípio básico para calcular uma tabela seria a verificação do operador a ser utilizado. Para cada operador (α, β, γ ou δ) temos um valor principal, o qual preenche o maior número de células dentro de uma tabela, um valor secundário, um valor ternário (se nível quaternário) e um valor indiferente.

Um primeiro algoritmo foi implementado, onde uma expressão de entrada simples, com apenas uma operação e duas variáveis, poderia ser digitada e na tela uma tabela com o resultado aparecia. As principais linguagens de programação utilizadas atualmente, C, C++, Turbo Pascal foram testadas para verificar qual apresentaria um desenvolvimento mais eficiente do algoritmo. Como o processo do algoritmo não requer problemas matemáticos muito complexos, não houve diferenças em termos de velocidade e simplicidade do código nas linguagens testadas. Portanto a linguagem de programação escolhida foi o Turbo Pascal, pois havia maior experiência na linguagem.

Para gerar a interface gráfica seria necessário um software mais complexo, não poderia ser feito em Turbo Pascal. Um ambiente gráfico disponível, o qual oferece muitas facilidades de programação e que suporta códigos em Pascal, é o Delphi, que foi o software de ambiente gráfico escolhido e que efetivamente foi muito satisfatório.

Alguns problemas foram encontrados na elaboração do código ainda em Pascal. Os diversos problemas encontrados e as soluções geradas podem ser verificados no Anexo A.

2.2 Motivações para o desenvolvimento da ferramenta

Para termos credibilidade nos processos de síntese de funções lógicas multi-valores, após obtermos as expressões intermediárias para C_0, C_1, ..., C_{n-1}, onde n indica o nível que estamos trabalhando, devemos colocar um termo a mais, para que na tabela intermediária obtenhamos apenas dois valores. Em C_0, apenas os valores 0 e 1, em C_1, apenas os valores 1 e 2, fazendo o mesmo até C_{n-1}. Para termos essa consistência colocamos os termos $\alpha 1$ no final da expressão para C_0, o termo $\beta 2$ para a expressão C_1, o termo $\gamma 3$ para a expressão C_2 (se nível quaternário) ou $\gamma 0$ (se o nível for ternário) e finalmente, o termo $\delta 0$ para a expressão de C_3 quando o nível for quaternário. Mas, nem sempre o aparecimento desses termos auxiliares é necessário. Uma aplicação do software proposto é a verificação da necessidade de se utilizar esses termos. Como podemos observar no exemplo abaixo, na síntese da tabela 2.1.

C		**A**			
		0	**1**	**2**	**3**
B	**0**	0	1	2	3
	1	0	1	2	3
	2	0	1	2	3
	3	0	1	2	3

Tabela 2.1. Função exemplo C

Para sintetizar a tabela 2.1 utilizando o processo convencional, o primeiro passo é a obtenção da expressão para C_0. A tabela 2.2 mostra os valores de C_0.

C_0		A			
		0	**1**	**2**	**3**
B	**0**	0	1	1	1
	1	0	1	1	1
	2	0	1	1	1
	3	0	1	1	1

Tabela 2.2. Função auxiliar exemplo C_0

De onde obtemos a expressão para C_0, dada por 2.1.

$$C_0 = A\alpha 1 \tag{2.1}$$

A expressão para C_1 é obtida através da tabela 2.3.

C_1		A			
		0	**1**	**2**	**3**
B	**0**	1	1	2	2
	1	1	1	2	2
	2	1	1	2	2
	3	1	1	2	2

Tabela 2.3. Função auxiliar exemplo C_1

Onde podemos obter a expressão para C_1, mostrada em 2.2. Para verificarmos a necessidade do termo $\beta 2$, podemos utilizar o software e verificar que realmente há necessidade de colocarmos o termo.

$$C_1 = A\beta\overline{A}\beta 2 \tag{2.2}$$

Como estamos trabalhando com quatro níveis, temos ainda a função C_2, onde analisamos os valores 2´s e 3´s da tabela. A tabela 2.4 mostra a função auxiliar C_2.

C_2		A			
		0	**1**	**2**	**3**
B	**0**	2	2	2	3
	1	2	2	2	3
	2	2	2	2	3
	3	2	2	2	3

Tabela 2.4. Função auxiliar exemplo C_2

Para a função auxiliar C_2 chegamos a expressão dada por 2.3.

$$C_2 = A\gamma\overline{A}\gamma\overline{\overline{A}}\gamma 3 \tag{2.3}$$

Para terminarmos a síntese da tabela devemos obter a função para C, mostrada em 2.3.

$$C = C_0\gamma C_1\delta C_2 \tag{2.4}$$

Substituindo C_0 (2.1), C_1 (2.2) e C_2 (2.4) em C (2.3), temos como função final para C, mostrada em 2.5.

$$C = (A\alpha 1)\gamma(A\beta\overline{A}\beta 2)\delta(A\gamma\overline{A}\gamma\overline{\overline{A}}\gamma 3) \tag{2.5}$$

Pelo método convencional adotado para síntese de funções, a expressão obtida para a tabela dada é a expressão encontrada para C, dada em 2.5 Não é possível através de simplificações obtermos uma expressão onde C= A, que através de uma inspeção é a expressão mais simplificada para a tabela. O software auxilia na veracidade das expressões obtidas através da síntese.

Um outro exemplo no qual verificamos a necessidade de utilizar uma ferramenta que gere tabelas dada a expressão é a síntese de circuitos lógicos como Somador Completo e Multiplicador. Como mostraremos, existem duas maneiras de chegarmos a uma expressão para esses circuitos. Seguindo o processo comum de síntese mostrado no capítulo anterior, obtemos expressões muito complexas. Através da inspeção da tabela, com alguma experiência, podemos conseguir uma expressão mais simples. E depois, verificarmos a igualdade dessas expressões. Manualmente, esse processo de verificação tomaria um tempo desnecessário ao projetista. O software pode calcular rapidamente as duas expressões e assim verificar a igualdade das mesmas.

A tabela 2.5 define a operação de Soma (S) e transporte anterior (V) para um Somador Completo, onde temos dois operandos a serem somados e acréscimo de um dígito de transporte. O dígito de transporte anterior (U) pode apresentar apenas dois valores, pois estamos efetuando uma soma com duas variáveis.

		0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**U**
S,V		**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**A**
B	**0**	00	01	02	03	01	02	03	10	
	1	01	02	03	10	02	03	10	11	
	2	02	03	10	11	03	10	11	12	
	3	03	10	11	12	10	11	12	13	

Tabela 2.5. Função Soma e Transporte

Pelo método clássico de Epstein [8] chegamos a expressão para S (Soma), mostrada em 2.6.

$$S = [e_0(B) \cdot e_1(A) \cdot e_0(U) \cdot 1] + [e_0(B) \cdot e_2(A) \cdot e_0(U) \cdot 2] +$$
$$[e_0(B) \cdot e_3(A) \cdot e_0(U)] + [e_0(B) \cdot e_0(A) \cdot e_1(U) \cdot 1] +$$
$$[e_0(B) \cdot e_1(A) \cdot e_1(U) \cdot 2] + [e_0(B) \cdot e_2(A) \cdot e_1(U)] +$$
$$[e_1(B) \cdot e_0(A) \cdot e_0(U) \cdot 1] + [e_1(B) \cdot e_1(A) \cdot e_0(U) \cdot 2] +$$
$$[e_1(B) \cdot e_2(A) \cdot e_0(U)] + [e_1(B) \cdot e_0(A) \cdot e_1(U) \cdot 2] +$$
$$[e_1(B) \cdot e_1(A) \cdot e_1(U)] + [e_1(B) \cdot e_3(A) \cdot e_1(U) \cdot 1] +$$
$$[e_2(B) \cdot e_0(A) \cdot e_0(U) \cdot 2] + [e_2(B) \cdot e_1(A) \cdot e_0(U)] +$$
$$[e_2(B) \cdot e_3(A) \cdot e_0(U) \cdot 1] + [e_2(B) \cdot e_0(A) \cdot e_1(U)] +$$
$$[e_2(B) \cdot e_2(A) \cdot e_1(U) \cdot 1] + [e_2(B) \cdot e_3(A) \cdot e_1(U) \cdot 2] +$$
$$[e_3(B) \cdot e_0(A) \cdot e_0(U)] + [e_3(B) \cdot e_2(A) \cdot e_0(U) \cdot 1] +$$
$$[e_3(B) \cdot e_3(A) \cdot e_0(U) \cdot 2] + [e_3(B) \cdot e_1(A) \cdot e_1(U) \cdot 1] +$$
$$[e_3(B) \cdot e_2(A) \cdot e_1(U)] + [e_3(B) \cdot e_3(A) \cdot e_1(U)] \qquad (2.6)$$

A expressão para o Somador obtida pelo método dos Operadores, descrito no capítulo 1, com algumas simplificações é dada pela expressão 2.7,

$$C = C_0 \gamma C_1 \delta C_2 \qquad (2.7)$$

onde, C_0, C_1 e C_2 são dadas pelas expressões 2.8, 2.9 e 2.10, respectivamente.

$$C_0 = (U\beta((A\beta B)\alpha(\overline{A}\beta\underline{B})\alpha(\overline{\overline{A}}\beta\overline{B})\alpha(\underline{A}\beta\overline{B})))\alpha(\underline{U}\beta((\overline{A}\beta B)\alpha(\overline{\overline{A}}\beta\underline{B})\alpha(\underline{A}\beta\overline{\overline{B}})\alpha(A\beta\overline{B})))\alpha 1 \qquad (2.8)$$

$$C_1 = (\overline{U}\gamma((\overline{A}\gamma\overline{B})\beta(A\gamma\overline{B})\beta(\overline{A}\gamma B)\beta(\overline{\overline{A}}\gamma B)\beta(\underline{A}\gamma\underline{B})\beta(\overline{\overline{A}}\gamma\underline{B})\beta(A\gamma\overline{\overline{B}})\beta(\underline{A}\gamma\overline{\overline{B}})))\beta$$
$$(U\gamma((\overline{A}\gamma\overline{B})\beta(\overline{\overline{A}}\gamma\overline{B})\beta(\underline{A}\gamma B)\beta(\overline{\overline{A}}\gamma B)\beta(A\gamma\underline{B})\beta(\underline{A}\gamma\underline{B})\beta(\overline{A}\gamma\overline{\overline{B}})))\beta 2 \qquad (2.9)$$

$$C_2 = (\overline{\overline{U}}\delta((\overline{\overline{A}}\delta\overline{\overline{B}})\gamma(\overline{A}\delta\overline{\overline{B}})\gamma(A\delta\overline{\overline{B}})\gamma(\overline{\overline{A}}\delta\overline{B})\gamma(\overline{A}\delta\overline{B})\gamma(\underline{A}\delta\overline{B})\gamma(\overline{\overline{A}}\delta B)\gamma(A\delta B)\gamma(\underline{A}\delta B)\gamma$$

$$(\overline{A}\delta\underline{B})\gamma(A\delta\underline{B})\gamma(\underline{A}\delta\underline{B})))\gamma(\overline{U}\delta((\overline{\overline{A}}\delta\overline{\overline{B}})\gamma(\overline{A}\delta\overline{\overline{B}})\gamma(A\delta\overline{\overline{B}})\gamma(\overline{\overline{A}}\delta\overline{B})\gamma(A\delta\overline{B})\gamma(\underline{A}\delta\overline{B})\gamma$$

$$(\overline{A}\delta B)\gamma(A\delta B)\gamma(\underline{A}\delta B)\gamma(\overline{\overline{A}}\delta\underline{B})\delta(\overline{A}\delta\underline{B})\gamma(A\delta\overline{B})))\gamma 3 \quad (2.10)$$

Através de uma inspeção na tabela do Somador (2.5) podemos concluir que a expressão para S (Soma) pode ser dividida em expressões mais simples (S_{00}, S_{01}, S_{02} e S_{03}), mostradas em 2.11, 2.12, 2.13 e 2.14, as quais são facilmente deduzidas.

$$S_{00} = A\beta(B\alpha 1) \quad (2.11)$$

$$S_{01} = \overline{A}\beta(\underline{B}\alpha 1) \quad (2.12)$$

$$S_{02} = \overline{\overline{A}}\beta(\overline{\overline{B}}\alpha 1) \quad (2.13)$$

$$S_{03} = \underline{A}\beta(\overline{B}\alpha 1) \quad (2.14)$$

Para chegarmos na expressão de S_0, devemos unir as expressões para S_{00}, S_{01}, S_{02} e S_{03} pelo operador γ. Então temos uma expressão simplificada para S_0, dada por 2.15.

$$S_0 = S_{00}\gamma S_{01}\gamma S_{02}\gamma S_{03} \quad (2.15)$$

Para chegarmos na expressão efetiva de S, devemos adicionar um termo onde a variável U (transporte) seja analisada. Então temos para S a expressão simplificada, dada por 2.16.

$$S = (S_0\beta U)\gamma(\overline{\overline{S_0}}\beta(\overline{U}\gamma 0)) \quad (2.16)$$

Utilizando o software pudemos verificar que a expressão completa obtida pelo método dos operadores tem como saída a mesma tabela da expressão obtida por inspeção. Esta é uma das principais aplicações do software.

Um segundo exemplo é a síntese do Multiplicador de 1 bit para 4 níveis. A tabela 2.6 define a multiplicação de dois fatores (A e B), com acréscimo de 1 bit de transporte

(U). Para a multiplicação com dois fatores, o transporte pode assumir três valores apenas (0, 1 ou 2). Suas saídas são o Produto (P) e o transporte (V).

		0	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**U**
V,P		**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	
B	**0**	0	0	0	0	1	1	1	1	2	2	2	2	
	1	0	1	2	3	1	2	3	10	2	3	10	11	
	2	0	2	10	12	1	3	11	13	2	10	12	20	
	3	0	3	12	21	1	10	13	22	2	11	20	23	

Tabela 2.6. Função Produto e Transporte

Pelo método clássico de Epstein [8] obtemos a expressão para P (produto), mostrada em 2.17.

$$P = [e_1(U)\cdot e_0(A)\cdot e_0(B)\cdot 1] + [e_1(U)\cdot e_1(A)\cdot e_0(B)\cdot 1] + [e_1(U)\cdot e_2(A)\cdot e_0(B)\cdot 1] +$$

$$[e_1(U)\cdot e_3(A)\cdot e_0(B)\cdot 1] + [e_2(U)\cdot e_0(A)\cdot e_0(B)\cdot 2] + [e_2(U)\cdot e_1(A)\cdot e_0(B)\cdot 2] +$$

$$[e_2(U)\cdot e_2(A)\cdot e_0(B)\cdot 2] + [e_2(U)\cdot e_3(A)\cdot e_0(B)\cdot 2] + [e_0(U)\cdot e_1(A)\cdot e_1(B)\cdot 1] +$$

$$[e_0(U)\cdot e_2(A)\cdot e_1(B)\cdot 2] + [e_0(U)\cdot e_3(A)\cdot e_1(B)] + [e_1(U)\cdot e_0(A)\cdot e_1(B)\cdot 1] +$$

$$[e_1(U)\cdot e_1(A)\cdot e_1(B)\cdot 2] + [e_1(U)\cdot e_2(A)\cdot e_1(B)] + [e_2(U)\cdot e_0(A)\cdot e_1(B)\cdot 2] +$$

$$[e_2(U)\cdot e_1(A)\cdot e_1(B)] + [e_2(U)\cdot e_3(A)\cdot e_1(B)\cdot 1] + [e_0(U)\cdot e_1(A)\cdot e_2(B)\cdot 2] +$$

$$[e_0(U)\cdot e_3(A)\cdot e_2(B)\cdot 2] + [e_1(U)\cdot e_0(A)\cdot e_2(B)\cdot 1] + [e_1(U)\cdot e_1(A)\cdot e_2(B)] +$$

$$[e_1(U)\cdot e_2(A)\cdot e_2(B)\cdot 1] + [e_1(U)\cdot e_3(A)\cdot e_2(B)] + [e_2(U)\cdot e_0(A)\cdot e_2(B)\cdot 2] +$$

$$[e_2(U)\cdot e_2(A)\cdot e_2(B)\cdot 2] + [e_0(U)\cdot e_1(A)\cdot e_3(B)] + [e_0(U)\cdot e_2(A)\cdot e_3(B)\cdot 2] +$$

$$[e_0(U)\cdot e_3(A)\cdot e_3(B)\cdot 1] + [e_1(U)\cdot e_0(A)\cdot e_3(B)\cdot 1] + [e_1(U)\cdot e_2(A)\cdot e_3(B)] +$$

$$[e_1(U)\cdot e_3(A)\cdot e_3(B)\cdot 2]+[e_2(U)\cdot e_0(A)\cdot e_3(B)\cdot 2]+[e_2(U)\cdot e_1(A)\cdot e_3(B)\cdot 1]+$$

$$[e_2(U)\cdot e_3(A)\cdot e_3(B)] \quad (2.17)$$

Se fizermos a síntese do Multiplicador de 1 bit para 4 níveis utilizando o método dos operadores, obteremos 3 funções intermediárias P_0, P_1 e P_2. Para chegarmos a expressão final de P (produto) teremos que unir as três funções pelos operadores γ e δ.

Para P_0 temos a tabela 2.7 com apenas os valores 0 e 1.

P_0		**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**U**
		0	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	
B	**0**	0	0	0	0	1	1	1	1	1	1	1	1	
	1	0	1	1	1	1	1	1	0	1	1	0	1	
	2	0	1	0	1	1	1	1	1	1	0	1	0	
	3	0	1	1	1	1	0	1	1	1	1	0	1	

Tabela 2.7. Função Auxiliar para Produto P_0

A expressão para P_0 é dada pela expressão 2.18.

$$P_0 = U\beta(A\alpha B)\alpha(U\beta\overline{\overline{A}}\beta\overline{\overline{B}})\alpha(\underline{U}\beta\underline{A}\beta\overline{B})\alpha(\underline{U}\beta\overline{A}\beta\underline{B})\alpha(\overline{\overline{U}}\beta\overline{\overline{A}}\beta\underline{B})\alpha(\overline{\overline{U}}\beta\overline{\overline{A}}\beta\overline{B})\alpha(\overline{\overline{U}}\beta\underline{\overline{A}}\beta\overline{\overline{B}})\alpha 1 \quad (2.18)$$

A tabela 2.8 define P_1, apenas com os valores 1 e 2.

P_1		0	0	0	0	1	1	1	1	2	2	2	2	U
		0	1	2	3	0	1	2	3	0	1	2	3	
B	0	1	1	1	1	1	1	1	1	2	2	2	2	
	1	1	1	2	2	1	2	2	1	2	2	1	1	
	2	1	2	1	2	1	2	1	2	2	1	2	1	
	3	1	2	2	1	1	1	2	2	2	1	1	2	

Tabela 2.8. Função Auxiliar para Produto P_1

Analisando os valores iguais a 1, chegamos a uma expressão para P_1, mostrada em 2.19.

$$P_1 = \overline{U}\gamma(\overline{A}\beta\overline{B})\beta U\gamma(\overline{A}\beta\overline{B})\beta(\overline{U}\gamma A\gamma B)\beta(\overline{U}\gamma\underline{A}\gamma\underline{B})\beta$$

$$(\overline{U}\gamma\overline{\overline{A}}\gamma\overline{\overline{B}})\beta(U\gamma A\gamma\overline{\overline{B}})\beta(U\gamma\underline{A}\gamma\underline{B})\beta(U\gamma\overline{\overline{A}}\gamma B)\beta$$

$$(\underline{U}\gamma A\gamma\underline{B})\beta(\underline{U}\gamma A\gamma\overline{\overline{B}})\beta(\underline{U}\gamma\underline{A}\gamma B)\beta(\underline{U}\gamma\underline{A}\gamma\overline{\overline{B}})\beta$$

$$(\underline{U}\gamma\overline{\overline{A}}\gamma B)\beta(\underline{U}\gamma\overline{\overline{A}}\gamma\underline{B})\beta 2 \qquad (2.19)$$

Finalmente, na tabela 2.9, os valores para P_2 são definidos, apenas os valores 2´s e 3´s da tabela inicial.

P_2		0	0	0	0	1	1	1	1	2	2	2	2	U
		0	1	2	3	0	1	2	3	0	1	2	3	
B	0	2	2	2	2	2	2	2	2	2	2	2	2	
	1	2	2	2	3	2	2	3	2	2	3	2	2	
	2	2	2	2	2	2	3	2	3	2	2	2	2	
	3	2	3	2	2	2	2	3	2	2	2	2	3	

Tabela 2.9. Função Auxiliar para Produto P_2

A expressão obtida para P_2 é dada pela expressão 2.20.

$$P_2 = (\overline{\overline{A}}\gamma\overline{\overline{B}})\gamma(\overline{\overline{U}}\gamma B)\gamma(U\gamma B)\gamma(U\gamma A)\gamma$$

$$(\overline{\overline{U}}\delta\overline{A}\delta\overline{B})\gamma(\overline{\overline{U}}\delta A\delta\overline{B})\gamma(\overline{\overline{U}}\delta A\delta\underline{B})\gamma(\overline{\overline{U}}\delta\underline{A}\delta\underline{B})\gamma$$

$$(\overline{U}\delta\overline{A}\delta\overline{B})\gamma(\overline{U}\delta\overline{A}\delta\underline{B})\gamma(\overline{U}\delta\underline{A}\delta\overline{B})\gamma(\overline{U}\delta\underline{A}\delta\underline{B})\gamma$$

$$(U\delta\overline{A}\delta\underline{B})\gamma(U\delta\underline{A}\delta\overline{B})\gamma 3 \qquad (2.20)$$

Finalmente a expressão para P é dada por 2.21.

$$P = P_0\gamma P_1\delta P_2 \qquad (2.21)$$

Através de uma inspeção na tabela do Multiplicador (2.6), podemos observar que se utilizarmos expressões intermediárias quando U=0, U=1 e U=2, chegamos a expressões um pouco mais simples.

Para U=0, chegamos a expressão dada por 2.22.

$$P_0 = (P_{00}\gamma P_{01})\delta P_{02} \qquad (2.22)$$

P_{00}, P_{01} e P_{02} são expressões mostradas por 2.23, 2.24 e 2.25, respectivamente.

$$P_{00} = A\,\alpha\,B\,\alpha\,(\overline{\overline{A}}\beta\,\overline{\overline{B}})\,\alpha\,1 \qquad (2.23)$$

$$P_{01} = \overline{P_{00}}\beta(A\gamma B)\beta(\overline{\overline{A}}\gamma\overline{\overline{B}}) \qquad (2.24)$$

$$P_{02} = \overline{P_{01}}\gamma(A\delta\overline{B})\gamma(\overline{A}\delta B)\gamma(\underline{A}\delta B)\gamma(A\delta\underline{B}) \qquad (2.25)$$

Para U=1 e U=2, as expressões seguem o mesmo raciocínio.

Utilizando o software concluímos que as expressões finais encontradas, tanto pelo método dos operadores quanto por inspeção, geram a tabela do Produto (P) do Multiplicador.

CAPÍTULO 3

DESCRIÇÃO DO PROGRAMA

3.1 Descrição Funcional do Programa ELOmv

O programa recebeu o nome de ELOmv, Equações Lógicas Multi-valores, pois seu objetivo é o cálculo das equações lógicas multi-valores, retornando uma tabela verdade para todas as opções possíveis. Neste capítulo será mostrado, em detalhes, todo o funcionamento do programa, desde a entrada dos dados até a saída do resultado na tela.

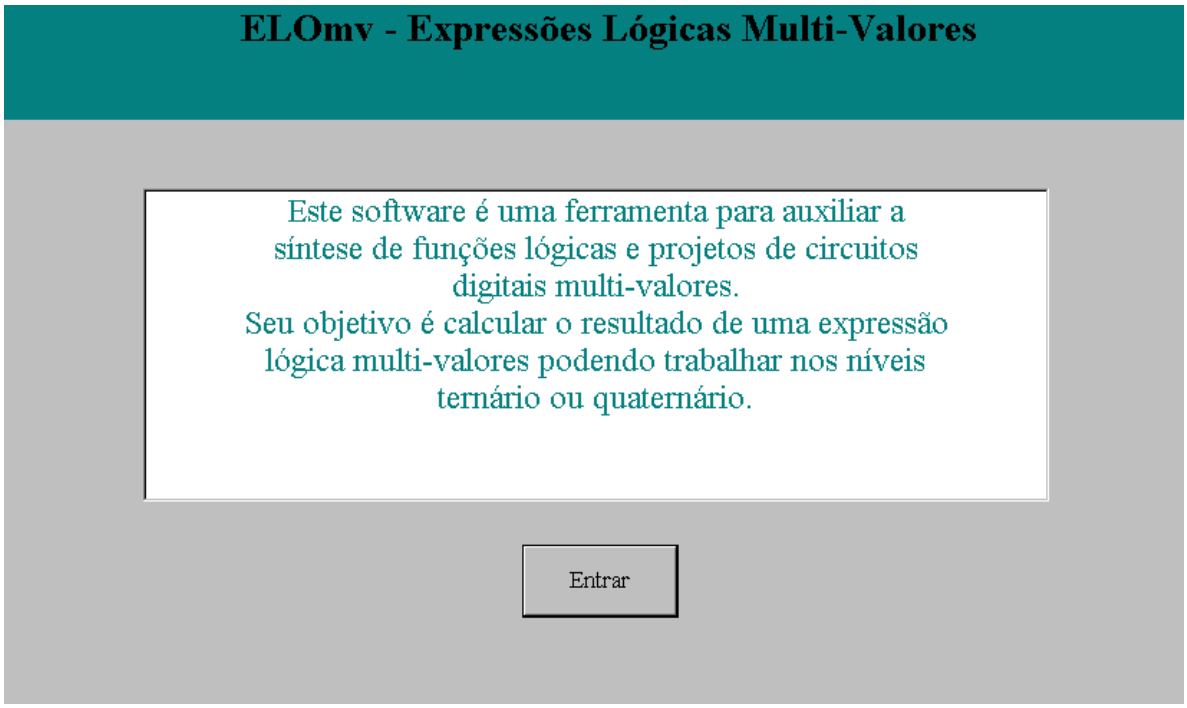


Figura 3.1 Tela de apresentação

Ao iniciar o programa, uma tela de apresentação (figura 3.1) aparece na tela em tamanho maximizado. Nesta primeira tela, com uma pequena explicação do objetivo do

programa, existe apenas um botão, **ENTRA**, que ao ser clicado, faz com que essa tela de abertura seja minimizada e a tela de entrada mostrada pela figura 3.2 fique visível.

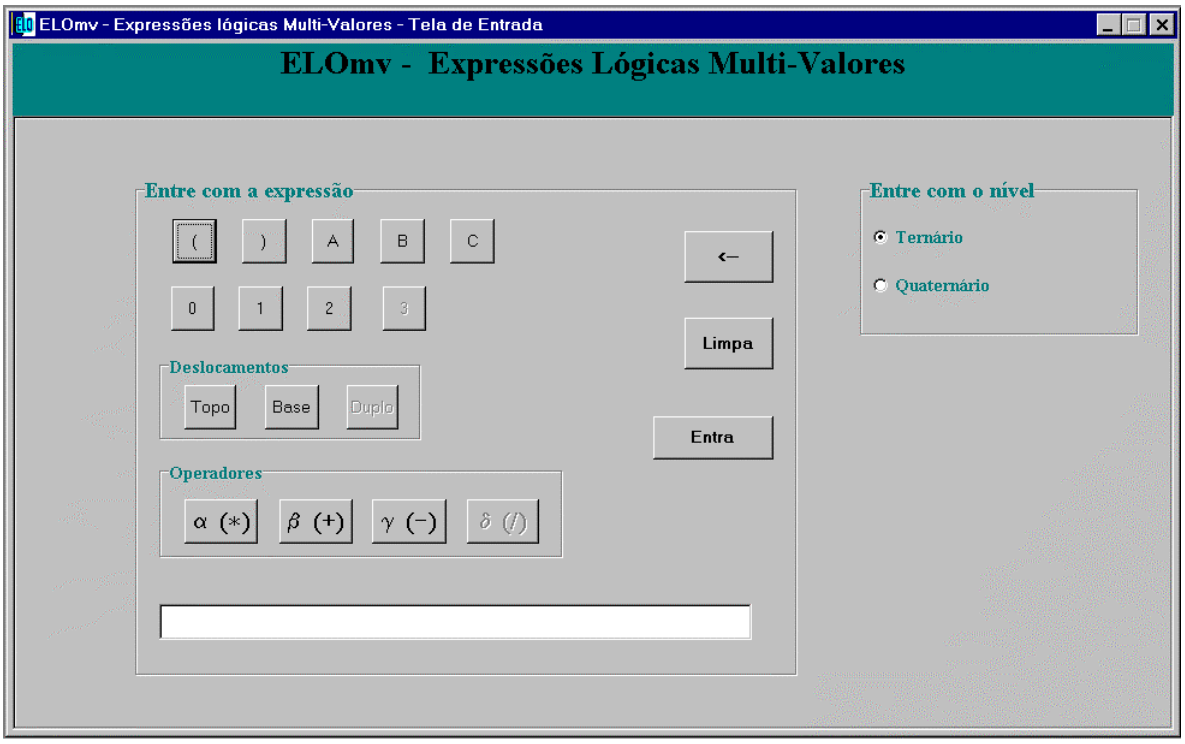


Figura 3.2 Tela de entrada para nível ternário

Na tela de entrada (figura 3.2), o usuário deverá entrar com a expressão desejada. Para definir as prioridades das operações, deve-se colocar os parênteses através dos botões ou do teclado. Os botões com as letras maiúsculas **A**, **B** e **C**, definem as variáveis que podem ser utilizadas para as expressões. Outros nomes de variáveis podem ser usados, para isso, o nome da variável deve ser digitado diretamente na caixa de entrada em letra maiúscula. Os botões com os valores **0**, **1**, **2** e **3**, podem ser usados para definir termos tais como $\alpha 1$, por exemplo.

Os botões de deslocamento, **Topo**, **Base** e **Duplo** definem os deslocamentos das variáveis, ou dos operadores, como podemos observar nos exemplo: (At α Bb)t. Os

deslocamentos aparecem em letras minúsculas, ‘t’ para topo, ‘b’ para base e ‘d’ para duplo. Dessa forma a expressão indica uma operação alfa topo entre A topo e B base.

O usuário também deve definir o nível em que a expressão deverá ser calculada, ternário ou quaternário, sendo o nível ternário definido como "default". Se o nível for alterado para quaternário, os botões **3**, **Duplo** e **δ(/)** tornam-se visíveis, como pode ser visto na figura 3.3. Como existe a possibilidade do usuário digitar diretamente na caixa de entrada um desses caracteres e não estar trabalhando em nível quaternário, uma caixa de diálogo de erro (figura 3.4) foi criada, e, torna-se visível se o nível for igual a 3 e o caracter digitado for igual a ´3´, ´d´ou ´/´.

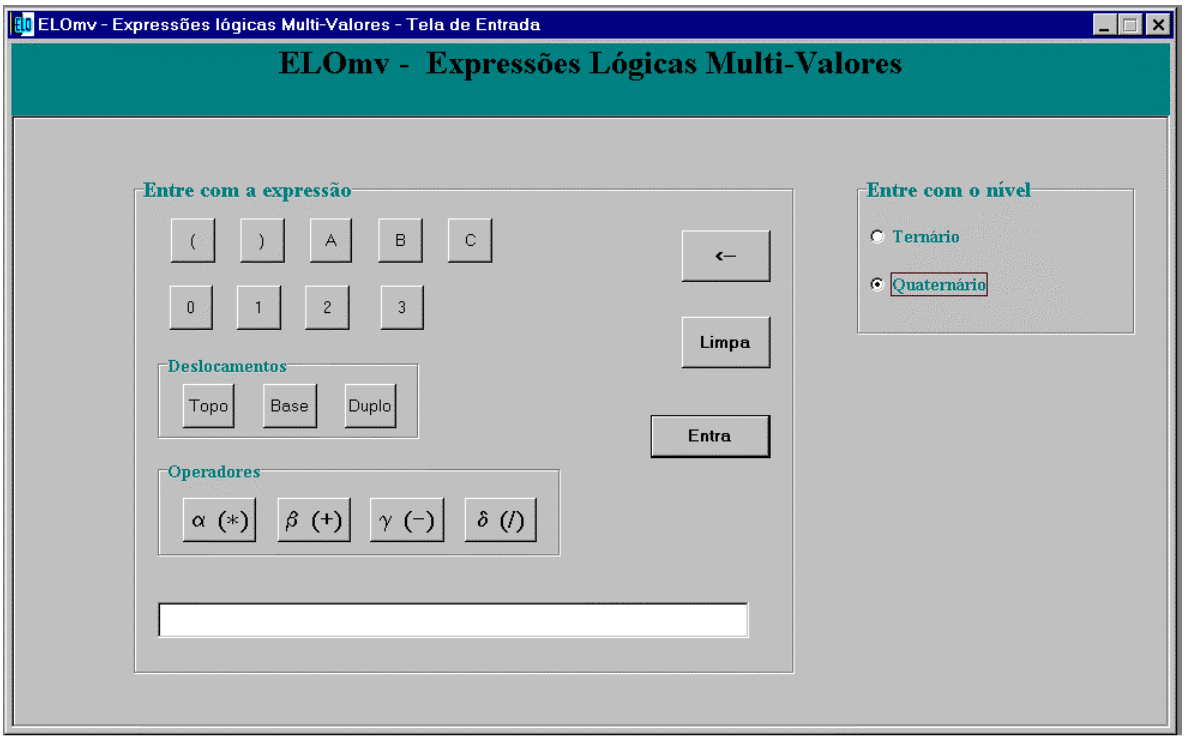


Figura 3.3 Tela de entrada para nível quaternário

Ao clicar no botão **Entra** da tela de entrada ou pressionar a tecla Enter, a expressão de entrada é lida e armazenada em uma variável do tipo **string** e o nível escolhido em uma variável do tipo **integer** chamada **nivel**.

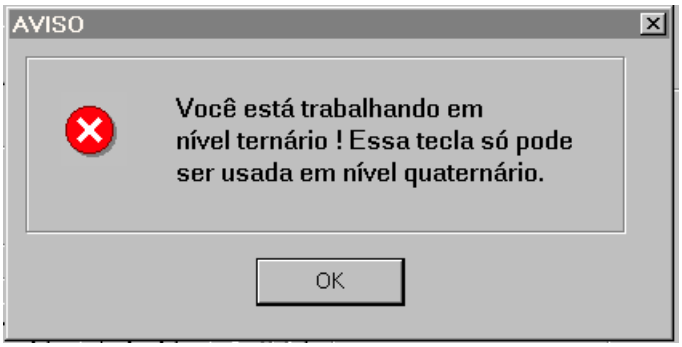


Figura 3.4 Tela de aviso

Após o botão **Entra** ser clicado, algumas verificações dentro da expressão de entrada são feitas. Para cada um dos erros encontrados uma caixa de diálogo é aberta colocando o problema para o usuário.

A primeira verificação é se a expressão contém caracteres repetidos por erro de digitação, como por exemplo AAαB. A caixa de diálogo para esse caso pode ser vista na figura 3.5.

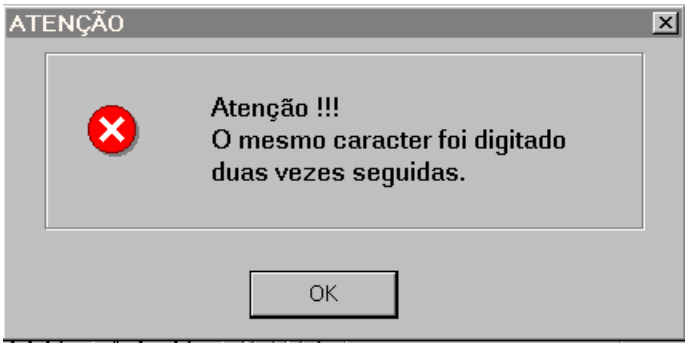


Figura 3.5 Tela de aviso caracteres repetidos

A segunda verificação é se existe dentro da expressão algum caracter que não pertence ao conjunto dos caracteres utilizados no programa, como podemos observar

no exemplo Aα4, o caracter ‘4’ não pode ser utilizado. Para esse caso a caixa de diálogo mostrada na figura 3.6 é aberta, indicando qual é o caracter não permitido.

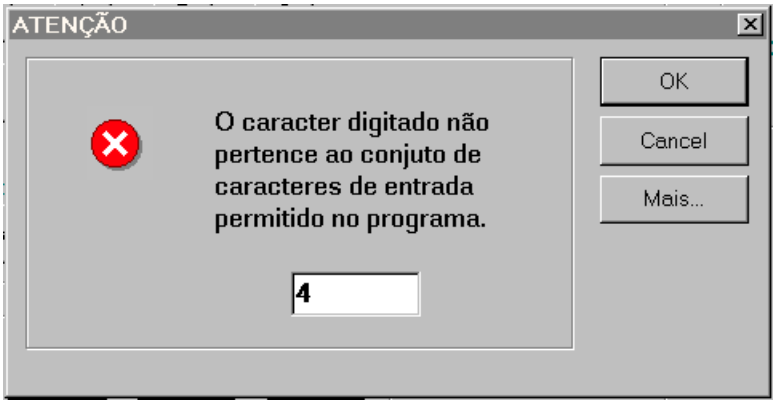


Figura 3.6 Tela de aviso para caracteres não permitidos

A terceira verificação é a quantidade de abre parênteses e fecha parênteses que deve ser igual; caso contrário, a caixa de diálogo mostrada na figura 3.7 aparece.

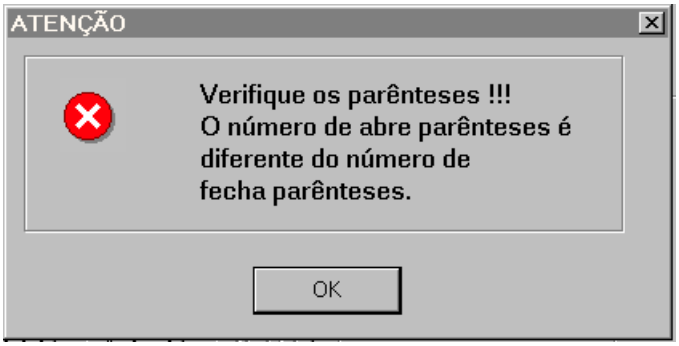


Figura 3.7 Tela de aviso para número de parênteses diferentes

A última verificação observa a presença de um operador entre duas variáveis e vice-versa. Um exemplo de expressão com esse erro é ABβC, deve haver um operador entre as variáveis A e B. Nesse caso a caixa de diálogo mostrada na figura 3.8 é aberta.

Figura 3.8 Tela de aviso para caracteres não permitidos

Para um melhor acompanhamento do algoritmo utilizado no programa, um diagrama de blocos foi criado com as principais "Units" e "procedures" do programa. Os diagramas de blocos para o programa ELOmv podem ser vistos nas figuras 3.9 3.10, 3.11 e 3.12.

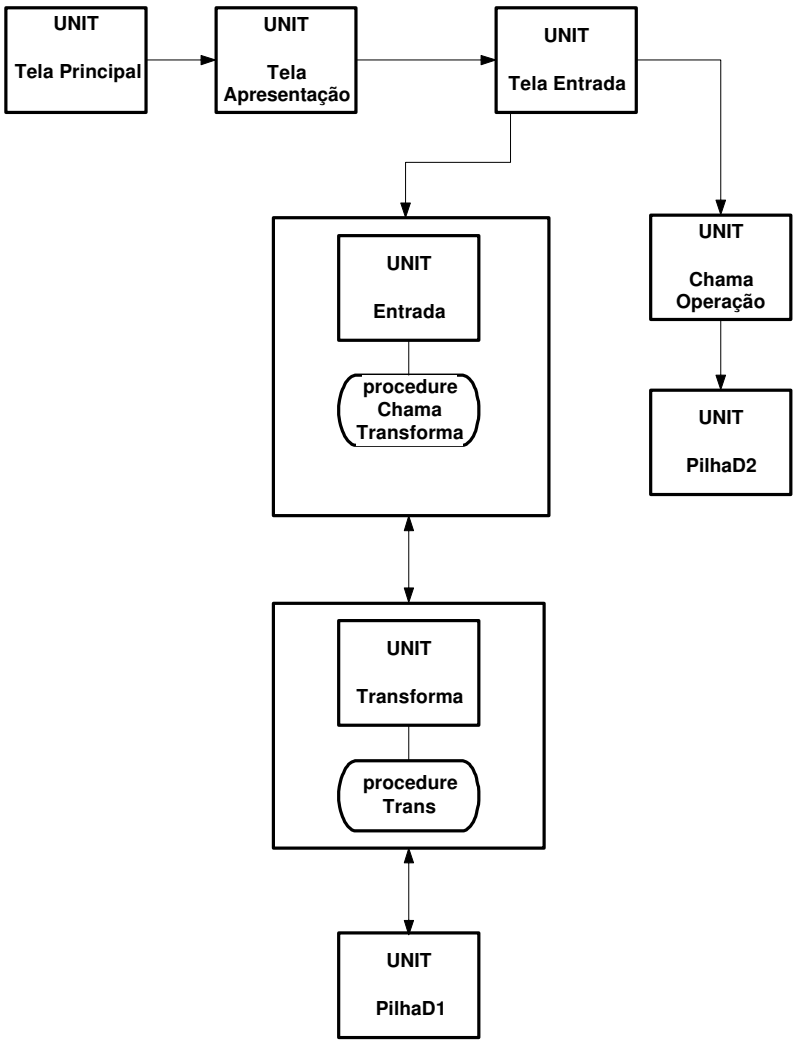


Figura 3.9 Diagrama de blocos

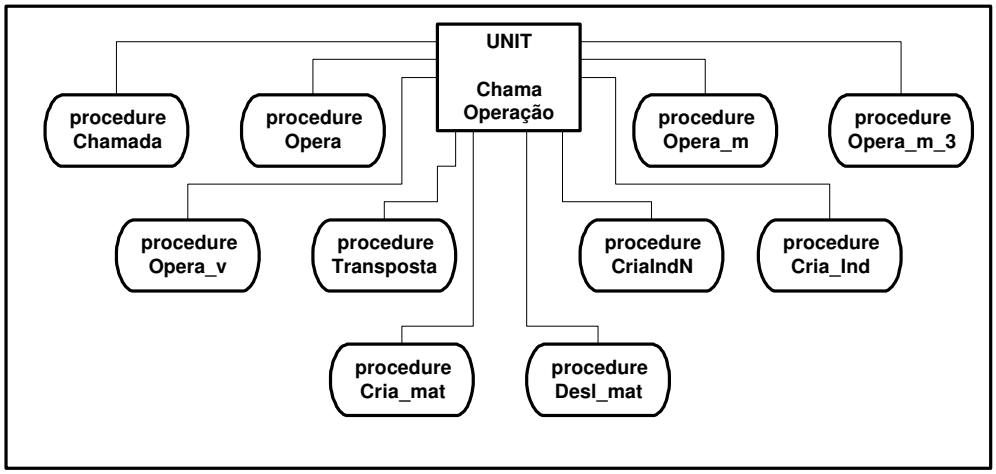


Figura 3.10. Diagrama de blocos para Unit Chama Operação

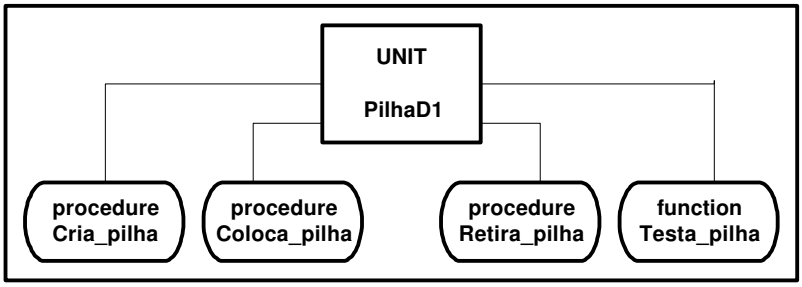


Figura 3.11. Diagrama de blocos para Unit PilhaD1

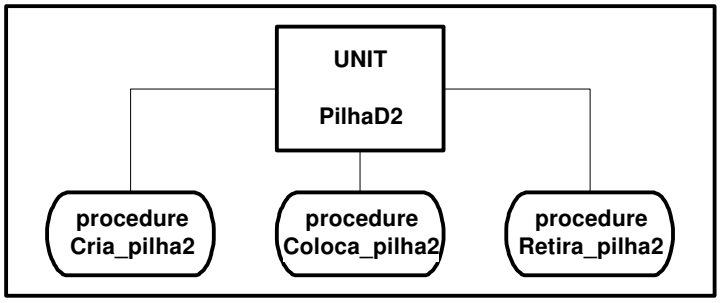


Figura 3.12. Diagrama de blocos para Unit PilhaD2

Através dos fluxogramas apresentados nas figuras 3.13, 3.14 e 3.15 pode ser feito um melhor acompanhamento das decisões feitas pelos algoritmos.

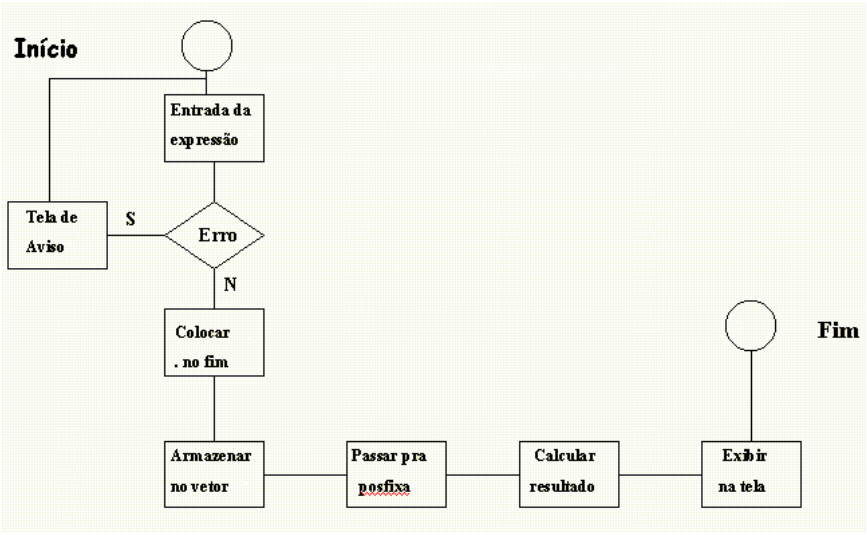


Figura 3.13 Fluxograma Geral

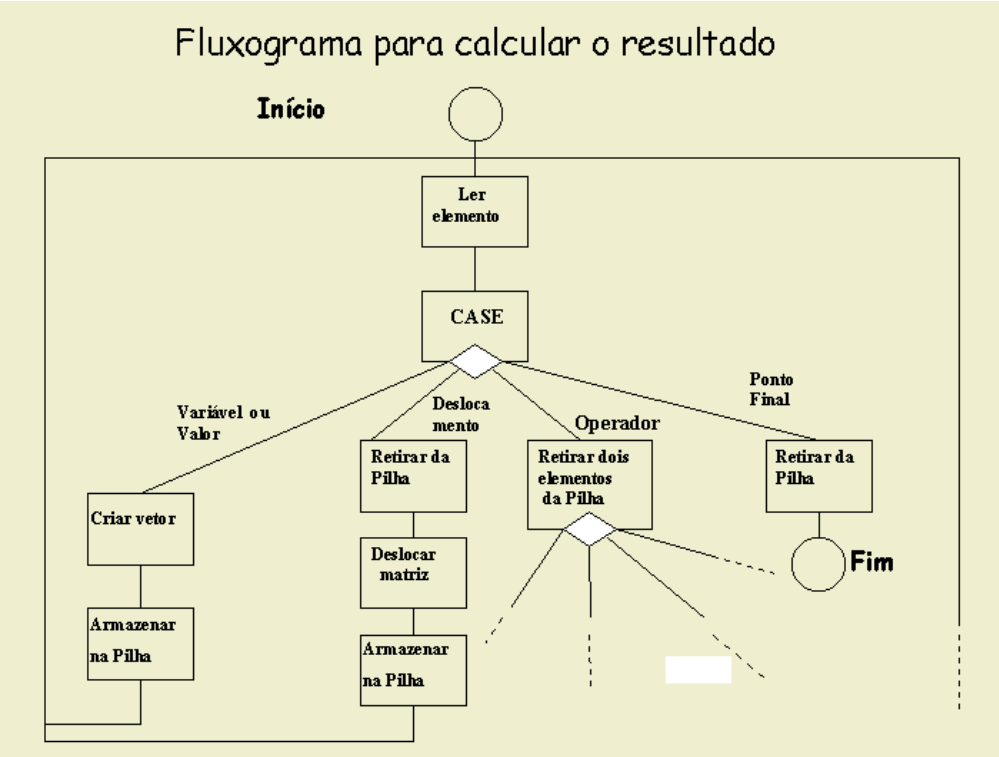


Figura 3.14 Fluxograma para calcular resultado

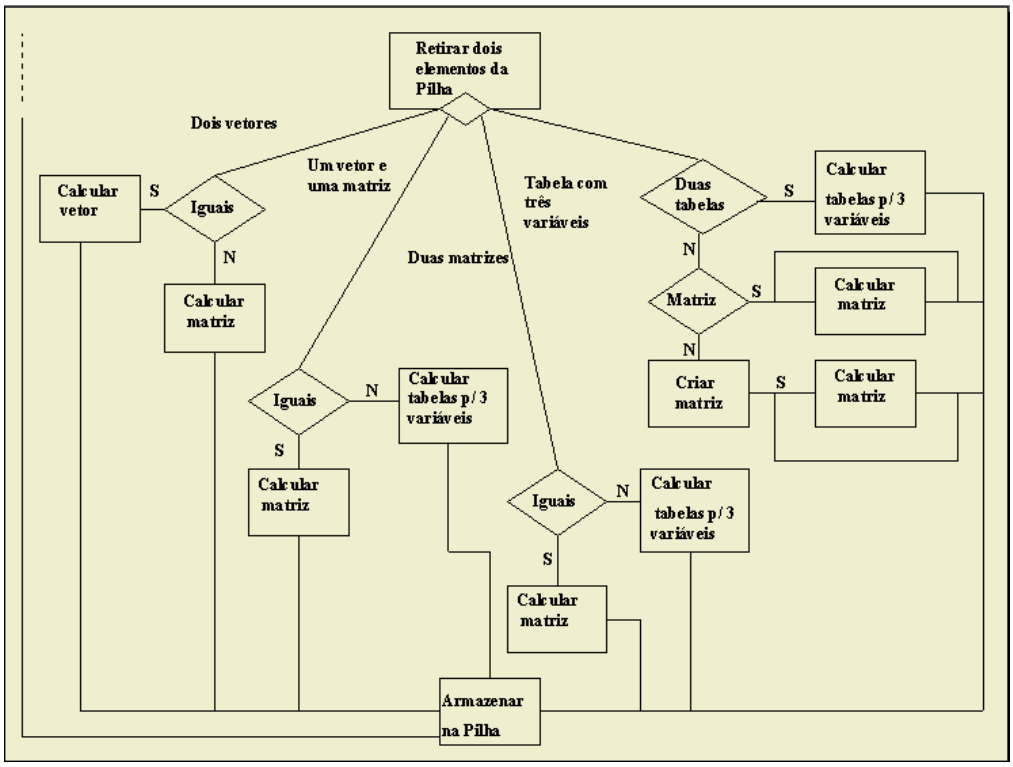


Figura 3.15 Continuação do fluxograma para Calcular Resultado

É feita uma chamada ao procedimento **Chamatra** da **Unit Entrada**. No procedimento **Chamatra** a **string** é transformada em uma estrutura do tipo **array,** definida na **Unit Transforma**, onde cada elemento possui a seguinte forma:

```
type
    expressa = record
                valor: char;
                tipo: char;
               end ;
    vet = array[0..300] of expressa;
```

É definida uma variável do tipo **vet**, com o nome de **infix**, e é responsável por armazenar toda a expressão de entrada na forma infixa, a forma que normalmente utilizamos para uma expressão numérica (A+B), ou no caso da lógica multi-valores (AαB).

Para cada valor lido da **string** é verificado se este é uma variável ou uma operação através da comparação com as letras maiúsculas de 'A' a 'Z' ou com os caracteres '*', '+', '-' e '/', armazenando esse caracter no campo **valor** de um elemento de **infix;** se o valor lido for o deslocamento (tipo) da variável ou da operação (base, duplo ou topo), comparado aos caracteres 't', 'b' ou 'd', então esse valor é armazenado no campo **tipo** do elemento de **infix** correspondente. Os parênteses que determinam prioridade dentro da expressão também são armazenados na variável **infix** no campo **valor**.

Após ler todos os caracteres da **string** inicial o procedimento **Chamatra** faz uma chamada ao procedimento **Trans** da **Unit Transforma**, enviando como parâmetros as variáveis **infix** e **posfix**. **Infix** e **posfix** possuem o mesmo tipo (**vet**). Sendo que a

variável **infix** armazena a expressão na forma infixa e **posfix** armazena na forma posfixa.

No procedimento **Trans**, a expressão de entrada, já na forma de array, é passada para sua forma posfixa, para poder ser calculada.

Na forma posfixa de uma expressão os operandos vêm primeiro e depois o operador, como podemos observar no seguinte exemplo: AB+. Seguindo da mesma forma para expressões algébricas multi-valores, se temos uma expressão como BβC, sua forma posfixa será BCβ. A transformação para forma posfixa é um dos métodos utilizados para resoluções de expressões numéricas, visto que em sua forma infixa é difícil termos um algoritmo que realize o cálculo. Na forma posfixa os parênteses são retirados e a prioridade das operações é definida pela posição do operador na forma posfixa. Exemplo: Dada a expressão na forma infixa Aα(BβC) temos como prioridade a operação BβC, na forma posfixa a expressão tem a seguinte forma ABCβα, podemos perceber que a operação β tem prioridade maior que a operação α. Essa transformação segue o algoritmo apresentado por Kruse [13] com algumas modificações necessárias.

Todos os caracteres armazenados na variável **infix** (forma infixa) são passados para a variável **posfix** (forma posfixa), exceto os parênteses. A variável chamada **posfix** possui a mesma estrutura da variável **infix** definida acima. Para fazer a transformação da forma infixa para a forma posfixa, o algoritmo utiliza uma estrutura auxiliar denominada **pilha**, definida na **Unit PilhaD1**, como mostrado a seguir:

```
type proximo=     ^expressa;
     expressa=    record
                  valor: char;
                  tipo: char;
```

prox: proximo;

end;

Para alocação dos elementos da pilha, foi utilizada a alocação dinâmica, que gera espaços na memória apenas quando for necessário, e, após a retirada do elemento, o espaço alocado é liberado da memória.

Ao terminar o procedimento **Trans** da **Unit Transforma**, retorna-se ao procedimento **Chamatra** da **Unit Entrada** e finalmente retorna-se ao procedimento do botão **Entra**.

Nesse procedimento é feita uma chamada ao procedimento **Chamada** da **Unit Chamaope**, no qual as tabelas serão calculadas. São passados como parâmetros **posfix**, **retorno**, **n** e **p**, definidos como **Transform.vet**, **PilhaD2.y**, **integer** e **PilhaD2.proximo**, respectivamente.

O tipo **vet** da **Unit Transform** foi declarado acima, serve para armazenar o array com valores na forma posfixa. A variável **n**, do tipo **integer**, armazena o nível em que se está trabalhando. O tipo **y** e **proximo** de **PilhaD2** será definido abaixo.

Para calcularmos uma expressão na forma posfixa, onde temos primeiro os operandos e depois os operadores, temos que ler cada elemento dessa expressão, verificando se o elemento é um operando que pode ser uma variável ('A'..'Z') ou um valor de '0' a '3' ou se o valor lido é um operador (α, β, γ ou δ).

Se o valor lido for um operando este deve ser armazenado em uma estrutura auxiliar denominada **pilha**, do tipo **PilhaD2.proximo**, assim como todos os outros valores lidos que forem do tipo operando. Ao ler um valor do tipo operador, devemos retirar dois valores da pilha (dois operandos) e realizarmos o cálculo entre os dois operandos pelo operador lido. O resultado dessa operação deve ser armazenado

novamente na pilha. Assim ao chegarmos ao final da expressão na forma posfixa, lendo o último caracter igual a '.', teremos como último elemento na nossa pilha o resultado final da expressão.

Quando obtemos a expressão na forma posfixa, que também é um array como apresentado acima, cada elemento desse array é lido sequencialmente.

A pilha serve para armazenar todas as variáveis e resultados de operações. É um ponteiro para o **tipo y**. Toda a estrutura é definida da seguinte forma:

```
type  proximo = ^y;
      mat = array[0..3,0..3]of integer;
      y =   record
            valor : mat;
            tipo  : char;
             oper1:char;
             oper2:char;
             oper3: char;
             prox: proximo;
             end;
```

No campo **valor**, o qual é uma matriz, são armazenadas as tabelas, que podem ser variáveis simples como A, matrizes que contêm resultados de uma operação com duas variáveis como AαB ou resultados de operações com três variáveis como AαBβC. No campo **tipo** podemos fazer essa identificação: se a variável é um vetor (variável de entrada) ou operação de (AαAt) estamos trabalhando com apenas uma variável e deve ser atribuído o valor 'v' para **tipo;** se for uma matriz (tabela resultado de uma operação), temos duas variáveis e o **tipo** tem valor 'm'; ou se for uma das tabelas resultantes de

operações com três variáveis, nesse caso **tipo** pode ser atribuído com os valores '1', '2', '3' ou '4'. Quando trabalhamos com três valores e temos três variáveis a tabela resultante será composta por três tabelas onde cada tabela é uma matriz 3x3, e se estivermos trabalhando com nível quaternário a tabela resultante será composta por quatro tabelas, onde cada tabela é uma matriz 4x4. Ao obtermos uma tabela, é importante guardarmos quais as variáveis que originaram a tabela, para isso os campos **oper1, oper2** e **oper3** foram criados. Exemplo: para uma tabela resultado da operação AαB, o campo **oper1** recebe o valor 'A' e o campo **oper2** recebe o valor 'B'. Se tivermos uma expressão AαBβC, nos campos **oper1** teremos o valor 'A', no campo **oper2** o valor 'B' e no campo **oper3**, o valor 'C'.

O procedimento para calcularmos a expressão na forma posfixa segue os seguintes passos:

- cada elemento do array posfix lido é verificado, através de uma estrutura de escolha **CASE** (estrutura de escolha) entre os casos a, b, c e d. Exemplos dos casos a, b, c e d podem ser observados em 3.2.
- a) Se o elemento for uma variável ('A'..'Z') ou caso b um valor de '0' a '3', este é transformado em um elemento do tipo **y**, através do procedimento **Cria_ind** ou **Cria_ind_num**, retornando um elemento com **tipo** igual a 'v' indicando um vetor. O elemento retornado é armazenado na pilha.
- c) Se o elemento lido for um deslocamento igual a 't', 'b' ou 'd' é retirada uma variável da pilha, e é verificado se o **tipo** da variável é igual a '3', então mais duas variáveis são retiradas da pilha ou se o **tipo** for igual a '4', mais três variáveis são retiradas da **Pilha**. Para cada uma das variáveis retiradas da pilha é feita uma chamada ao procedimento **Desl_mat**, que faz um deslocamento (topo, base ou

duplo) em todos os elementos do campo **valor** da variável. A variável ou as variáveis que retornam do procedimento **desl_mat** são armazenadas novamente na pilha.

- d) Se o elemento lido for um operador (α, β, γ, δ), dois dados da pilha são removidos, armazenados em variáveis do tipo **y** (elemento1 e elemento2). É verificado através do campo **tipo** das variáveis **elemento1** e **elemento2**, se os dados são resultados de uma operação com três variáveis, onde **tipo** pode ser igual a '3' (se o nível for ternário) ou igual a '4' (se nível quaternário) . Se forem tabelas resultantes de operação com três variáveis, as demais tabelas resultados da operação também são retiradas da pilha e armazenadas em outras variáveis do tipo **y**.
- Para obter o resultado da operação entre os dados lidos é utilizada uma estrutura de " if's aninhados " onde é verificado se é verdadeiro um dos casos entre e, f, g, h, i, j, k ou l. Exemplos de situações para cada um dos casos citados podem ser vistos em 3.2.
    - e) Os dois dados são do **tipo** vetor e têm no campo **oper1** o mesmo nome de variável (estamos trabalhando com apenas uma variável), o procedimento **opera_v** é chamado e retorna uma variável do tipo **y** (**retorno**) com campo **tipo** igual a 'v' indicando que o resultado obtido também é um vetor. Ex.: (AαA topo).
    - f) Os dois dados são do **tipo** vetor mas têm no campo **oper1** nomes diferentes (duas variáveis), o procedimento **opera** é chamado e retorna para a variável retorno com campo **tipo** igual a 'm', indicando que é uma matriz. Ex.: AαB.
    - g) Um dos dados é um vetor e o outro dado é uma matriz, sendo que um dos campos **oper1** ou **oper2** da variável com tipo = 'm' é igual ao campo **oper1** do

da variável com tipo = ´v´ (estamos trabalhando com duas variáveis). Ex.: primeiro elemento A e segundo elemento AαB. O elemento que for um vetor é transformado através do procedimento **Cria_mat** em uma matriz, é verificado se o campo **oper1** do vetor é igual ao campo **oper1** ou **oper2** da matriz. Se for igual ao **oper2**, a matriz gerada para o vetor deve ser modificada através do procedimento **transposta**, o qual retorna a matriz transposta para a matriz de entrada. Depois o procedimento **opera_m** é chamado tendo como parâmetro a matriz gerada e a outra matriz já existente. Retorna uma nova variável que têm nos campos **oper1** e **oper2** os nomes das duas variáveis e no campo **tipo** um identificador para matriz 'm'.

- h) Um dos dados é um vetor e o outro é uma matriz, sendo que o campo **oper1** e **oper2** da matriz diferem do campo **oper1** do vetor (estamos trabalhando com três variáveis). Ex.: primeiro elemento A e segundo elemento BαC. O elemento com campo **tipo** igual a vetor é transformado em uma matriz e depois é feita uma chamada ao procedimento **opera_m_3**. Nesse procedimento são calculadas três tabelas (se nível ternário) ou quatro tabelas (se nível quaternário). Se o nível for ternário para cada uma das três tabelas calculadas, temos no campo **tipo** os seguintes identificadores: '1', '2' e '3'. Se nível for quaternário são geradas quatro tabelas que recebem nos campos **tipo** os seguintes identificadores: '1', '2', '3' e '4'. Nos campos **oper1** e **oper2** das tabelas resultantes temos os mesmos valores dos campos **oper1** e **oper2** da matriz e no campo **oper3** o valor do campo **oper1** do vetor.
- i) Os dois dados são do tipo matriz, e temos apenas duas variáveis (os campos **oper1** e **oper2** das duas matrizes são iguais). Ex.: primeiro elemento

AαB e segundo elemento AβB, é feita uma chamada ao procedimento **opera_m** que retorna apenas uma variável que recebe no campo **tipo** o valor 'm'.

- j) Os dois dados são do tipo matriz e temos três variáveis (apenas um dos campos **oper1** ou **oper2** de uma variável corresponde a um dos campos **oper1** ou **oper2** da segunda variável). Ex.: primeiro elemento AαB e segundo elemento AδC. Para calcularmos as três ou quatro tabelas resultantes dessa operação, foi criada uma variável do tipo **y** chamada **temp**. Dentro de um "loop" que percorre toda a matriz do campo valor do segundo elemento, os valores das células de cada uma das linhas dessa matriz são armazenadas na variável **temp**. Primeiro, a variável **temp** recebe os valores da primeira linha do segundo elemento. É feita uma chamada ao procedimento **cria_mat** (o qual retorna uma matriz), depois é feita uma chamada ao procedimento **opera_m**, com os seguintes parâmetros: a matriz do primeiro elemento e a matriz **temp** gerada. O campo **tipo** do elemento resultante tem valor ´1´, indicando a primeira tabela de uma operação com três variáveis. A variável com o resultado é armazenada na pilha. O mesmo procedimento é feito para as demais linhas da matriz do segundo elemento até que todas as linhas tenham sido passadas para a variável **temp**, transformadas em matrizes e calculadas as tabelas que são armazenadas na pilha.
- k) Um dos dados é uma variável com **tipo** igual a '3' ou '4' indicando que restam mais duas ou três variáveis a serem retiradas da pilha. As demais variáveis são retiradas da pilha e armazenadas em variáveis do tipo **y**. É verificado se o segundo elemento retirado da pilha é um vetor ou matriz. Se o

elemento for um vetor, este é transformado em uma matriz e é feita uma chamada ao procedimento **opera_m** para cada uma das variáveis (3 para ternário e 4 para quaternário) tendo como parâmetros uma das variáveis e o outro elemento transformado em matriz. Para cada tabela resultante o campo **tipo** será definido com o mesmo campo **tipo** das três ou quatro tabelas. Se o elemento for uma matriz, serão feitas três ou quatro chamadas ao procedimento **opera_m** tendo como parâmetro a matriz e cada uma das variáveis com **tipo** iguais a '1', '2', '3' e '4' (se nível quaternário), retornando variáveis que têm em seu campo **tipo** os mesmos valores das variáveis de entrada. Cada uma das variáveis resultantes é armazenada na pilha.

- l) Os dois elementos retirados da pilha são tabelas resultantes de operações com três variáveis com campo **tipo** igual a '3' ou '4'. Para cada um dos elementos são retiradas as outras tabelas e armazenadas em variáveis do tipo **y**. Para nível ternário são feitas três chamadas ao procedimento **opera_m**, tendo como parâmetros as tabelas com **tipo** igual a '3', ´2´e ´1´ retornando três tabelas com **tipo** iguais a ´3´, ´2´e ´1´. Para nível quaternário, o procedimento é o mesmo, fazendo quatro chamadas a **opera_m**.

- Todas as variáveis resultantes dos procedimentos de cálculo são armazenadas na pilha para todos os casos descritos de e a l.
- Este processo de leitura dos elementos do array **posfix** continua até que seja encontrado no campo **valor** do elemento lido o caracter '.' que indica o final da expressão.

Ao terminar o procedimento **Chamada**, retorna-se ao procedimento do botão **Entra**. As saídas são armazenadas nas tabelas das telas de saída.

Primeiro verifica-se se o **tipo** da variável **retorno**, a qual é o parâmetro passado no procedimento **chamatra** que retorna o resultado final da expressão, é igual a 'v', indicando um vetor. Para essa condição apenas a primeira linha do campo **valor** da variável **retorno**, contém os valores de saída. As demais linhas da tabela recebem o valor da primeira linha. Esta condição pode ser mais bem observada através do exemplo mostrado pelas tabelas 3.1 e 3.2.

A		
0	**1**	**2**
0	0	1
0	0	0
0	0	0

Tabela 3.1 Exemplo de tabela do tipo vetor

Somente os elementos da linha, onde temos os valores 0 0 1, correspondem ao valor de saída, pois estamos trabalhando com apenas uma variável. Para exibir na tela de saída, os elementos da primeira linha são copiados para as demais linhas. Assim temos na tela os valores vistos na tabela 3.2.

A		
0	**1**	**2**
0	0	1
0	0	1
0	0	1

Tabela 3.2 Exemplo de saída na tela de tipo vetor

A segunda condição verificada é se o **tipo** de retorno é igual a ‘3’, dois elementos da pilha são retirados e armazenados em **retorno1** e **retorno2**. O mesmo acontece se o **tipo** de retorno é igual a ‘4’. Retiram-se três outros elementos da pilha, que são armazenados em **retorno1**, **retorno2** e **retorno3**.

A tela de entrada, **Form4**, é minimizada, e verifica-se se o nível é igual a 3 ou 4. Se nível igual a 3, a tela de saída para três níveis, **Form5** torna-se visível, e, se o nível for 4, a tela de saída para quatro níveis, **Form6** torna-se visível.

Após verificar se o nível é 3 ou 4, é colocado na tela de saída correspondente, **Form5** ou **Form6**, a expressão de entrada em uma variável do tipo **Edit** na forma de caixa de texto e o nome da variável contida no campo **oper2** da variável **retorno**.

Figura 3.16 Tela de saída para três variáveis nível ternário

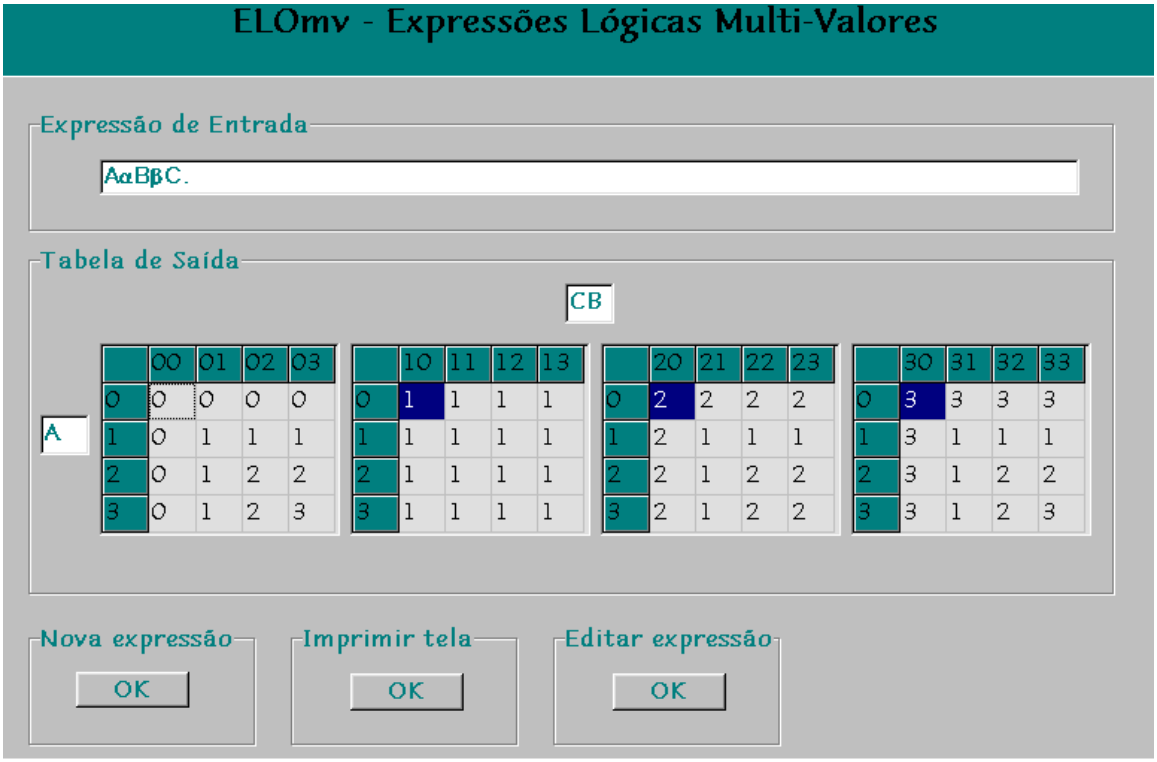


Figura 3.17 Tela de saída para três variáveis nível quaternário

Verificando se o campo **tipo** de **retorno** é igual a '3' ou '4', as demais tabelas das telas de saída tornam-se visíveis. Para tipo igual a '3', as variáveis **StringGrid2** e **StringGrid3** da **Form5** tornam-se visíveis, como podemos observar na figura 3.16. E, para tipo igual a '4', as variáveis **StringGrid2**, **StringGrid3** e **StringGrid4** da **Form6** tornam-se visíveis, como observamos na figura 3.17. Os nomes das variáveis armazenados em **oper3** e **oper1** são colocadas em uma pequena caixa de texto centralizada acima das tabelas nas telas de saída.

Se a condição de **tipo** ainda for '3' ou '4', os seguintes passos são feitos:

- preenchimento das células fixas das tabelas, indicando as linhas, nas telas de saída com os valores 0, 1, 2 e 3 (se nível for quaternário);
- preenchimento da caixa de texto localizada à esquerda da tabela na tela com o nome da variável em **oper2**;

- preenchimento das células fixas das tabelas, indicando as colunas, com os valores das duas variáveis armazenadas em **oper1** e **oper3** nas caixas de texto centralizadas acima das tabelas. Os valores para as células seguem a seguinte ordem: 00, 01, 02, 03 (se nível for quaternário), 10, 11, 12, 13 (apenas para nível quaternário), 20, 21, 22, 23 (nível quaternário) e finalmente apenas para nível quaternário, 30, 31, 32 e 33;
- preenchimento dos valores das tabelas. Testando se nível é igual a 3 ou 4, optando-se pela **Form5** ou **Form6**, respectivamente;
- primeira tabela, armazenada na tela pela variável **StringGrid1**, tanto para a **Form5** quanto para a **Form6**;
- segunda tabela, armazenada em **StringGrid2**;
- terceira tabela, armazenada em **StringGrid3** e, apenas se o nível for quaternário, para a quarta tabela, armazenada em **StringGrid4** na **Form6**.

Se o **tipo** de **retorno** não for igual a '3' ou '4', estamos trabalhando com uma matriz ou um vetor, ou seja, apenas uma ou duas variáveis. Sendo assim apenas uma tabela deverá estar visível nas telas de saída. O primeiro passo é o preenchimento das células fixas com os valores 0, 1, 2 e 3, se nível for quaternário. O segundo passo é o preenchimento dos valores da tabela, que são armazenados na variável **StringGrid1** tanto para **Form5** quanto para **Form6**.

A tela de saída para nível ternário com uma ou duas variáveis, pode ser observada através da figura 3.18.

Figura 3.18 Tela de saída para nível ternário

Para nível igual a quatro, temos outra tela de saída, que apenas difere da tela para três valores, no formato da tabela. Para três valores, as tabelas possuem formato 3x3, e para quatro as tabelas possuem formato 4x4.

Uma saída para nível quaternário com uma ou duas variáveis pode ser vista na figura 3.19.

Podemos observar nas telas de saída os seguintes botões: **Nova Expressão**, **Imprimir Tela** e **Editar Expressão**.

Ao clicar no botão **Nova Expressão**, a tela de saída ativa é minimizada e a tela de entrada torna-se visível novamente para a entrada de uma nova expressão. A caixa de texto para a entrada da expressão na tela de entrada (figura 3.2) fica vazia.

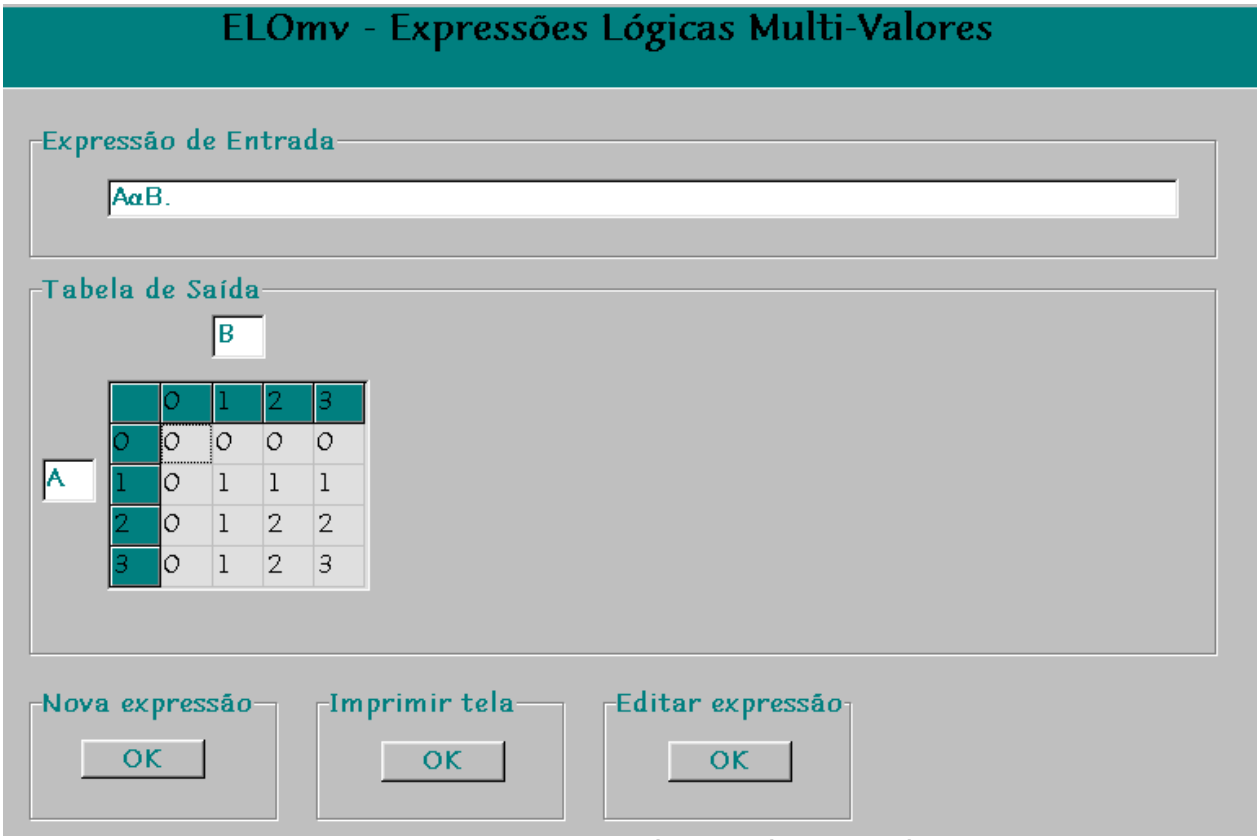


Figura 3.19 Tela de saída para nível quaternário.

Uma opção do usuário é a impressão da tela de saída, incluindo todos os itens visualizados na tela ativa.

Se o usuário necessitar apenas modificar a expressão de entrada, alterando algum operador, variável ou nível, por exemplo, poderá clicar no botão **Editar Expressão**, que retornará a tela de entrada (figura 3.2), conservando na caixa de texto da tela de entrada a expressão digitada pelo usuário inicialmente.

Outras opções do usuário podem ser vistas na tela principal (**MainForm**), onde temos a barra de menus e as barras de ferramentas. A tela principal sempre está presente, e todas as demais janelas são abertas dentro de sua área de trabalho. A tela principal pode ser vista na figura 3.20.

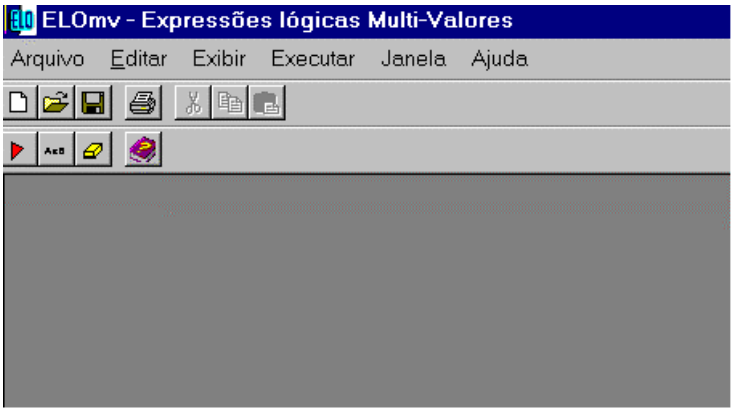


Figura 3.20 Tela Principal

Utilizando o menu principal, o usuário pode escolher entre salvar o resultado em um arquivo do tipo .txt, através do menu **Arquivo|Salvar** ou **Arquivo|Salvar Como**, ou ainda utilizando o botão **Salvar** da barra de ferramentas. Ao salvar uma saída pela primeira vez, sempre a opção **Salvar Como** é aberta, pois o usuário deve entrar com o nome do arquivo. Ao salvar uma segunda saída, o usuário pode escolher entre gravar no mesmo arquivo, usando a opção **Salvar**, ou gravar em um arquivo novo, usando a opção **Salvar Como**.

Existe a possibilidade de imprimir o arquivo .txt com as saídas ou de imprimir a tela de saída com as tabelas, da mesma forma que aparecem na tela. Para imprimir o arquivo .txt, deve ser usada a opção do menu **Arquivo|Imprimir** ou ainda o botão **Imprimir** na barra de ferramentas, quando o arquivo a ser impresso estiver ativo na área de trabalho.

Se for necessário, o usuário poderá alterar as configurações da impressora, como tamanho do papel, orientação do tipo retrato ou paisagem, e outras propriedades comuns em **Arquivo|Configurar Impressora**.

Uma outra possibilidade para o usuário é criar um arquivo de texto (.txt) novo, usando o menu **Arquivo**|**Novo**, ou o botão **Novo** da barra de ferramentas, ou ainda abrir um arquivo já existente, através do comando **Arquivo**|**Abrir** ou do botão **Abrir** na barra de ferramentas. Para fechar um arquivo ativo na tela o usuário pode usar o comando **Arquivo**|**Fechar**.

3.2 Exemplos dos casos

A) O elemento lido da variável **posfix** possui em seu campo **valor** o caracter 'A' e em seu campo **tipo** o caracter 't'. É feita uma chamada ao procedimento **Cria_ind** que retorna uma variável chamada **retorno**. A variável 'A' com deslocamento 't' (Topo) deverá receber os valores apresentados na tabela 3.3.

1	2	3	0

Tabela 3.3 Valores da variável do caso a

O campo **valor,** que é a matriz que armazena os valores das tabelas, ficará da forma apresentada pela tabela 3.4, onde podemos verificar que apenas a primeira coluna da tabela está preenchida, indicando que o elemento é um vetor.

1			
2			
3			
0			

Tabela 3.4 Campo valor para caso a

Os demais campos da variável retorno devem ser atribuídos com os seguintes valores: **tipo** = 'v', **oper1** = 'A', **oper2** e **oper3** = '' (vazio), gerando uma variável com os seguintes campos: variável **retorno** = (1,'','','',2,'','','',3,'','','',0,'','','','v','A','','') .

B) O elemento lido possui em seu campo **valor** o caracter '0', indicando um único valor, e não pode ter deslocamento. Para esse caso é feita uma chamada ao procedimento **Cria_IndN**, o qual retorna uma variável do tipo **y** chamada **retorno**. No campo **valor** da variável retorno temos a tabela como apresentada na tabela 3.5.

0			
0			
0			
0			

Tabela 3.5 Campo valor para caso b

Os demais campos da variável retorno recebem os seguintes valores: campo **tipo** = 'v', campo **oper1**='0', campo **oper2** e **oper3** = ' '.

C) O elemento lido é um deslocamento (topo, base ou duplo) de um operador, pois o deslocamento para operadores está no campo **valor** e não no campo **tipo**. Como exemplo colocaremos o caracter 'b', que indica um deslocamento Base. É retirado um elemento da pilha e armazenado em **elemento1**, verificado se o **tipo** do elemento é igual a '3' ou '4'. Para esses casos são retirados outros elementos da pilha. Como exemplo, o **elemento1** retirado da pilha possui em seus campos os valores : campo **tipo**= 'm', indicando que é uma matriz; campo **oper1**= 'A' e campo **oper2**= 'B', indicando que a matriz é um resultado de uma operação entre as variáveis A e B; campo **valor** da variável **elemento1** pode ser visto na tabela 3.6.

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.6 Campo valor para caso c

Após retirar o **elemento1** da pilha é feita uma chamada ao procedimento **desl_mat** passando como parâmetro a variável **elemento1** e o elemento lido, igual a 'b'. Como resultado do procedimento **desl_mat** temos a mesma variável **elemento1** com o campo **valor** modificado como apresentado na tabela 3.7.

3	**3**	**3**	**3**
3	**0**	**0**	**0**
3	**0**	**1**	**1**
3	**0**	**1**	**2**

Tabela 3.7 Campo valor calculado para caso c

Comparando as tabelas 3.6 e 3.7 observamos que os valores do campo **valor** sofreram um deslocamento para a esquerda, base. Os outros campos da variável **elemento1** não tiveram nenhuma modificação.

D) O elemento lido é um operador, como exemplo o caracter '*' no campo **valor**, o qual indica o operador α, e no campo **tipo** não tem nenhum caracter, pois para os operadores não há um deslocamento vinculado no campo **tipo**.

Para calcular a tabela resultante entre os elementos a serem retirados da pilha e que devem ser operador por ´*´, passamos para a descrição dos casos e ao caso l.

E) Os dois elementos retirados da pilha possuem no campo **tipo** o caracter 'v', indicando dois vetores, e seus campos **oper1** são iguais. Como exemplo temos a variável **elemento1** que possui em seu campo **oper1** o caracter 'A' e no seu campo **valor** os valores apresentados na tabela 3.8. Para o elemento2, o campo **oper1** também é igual a 'A' e no seu campo **valor** os valores apresentados na tabela 3.9.

0	1	2	3

Tabela 3.8 Valores do campo valor para elemento1 do caso e

1	2	3	0

Tabela 3.9 Valores do campo valor para elemento2 do caso e

0	1	2	0

Tabela 3.10 Valores do campo valor para retorno do caso e

No procedimento **opera_v** temos como resultado um vetor, armazenado na variável **retorno**, que no exemplo, recebe em seu campo **valor** os dados mostrados na tabela 3.10. A tabela mostra apenas uma linha pois o resultado também é um vetor, embora esteja armazenado em uma tabela, ocupando somente a primeira coluna da matriz do campo **valor**. O campo **tipo** da variável retorno recebe 'v', o campo **oper1** 'A' e os campos **oper2** e **oper3** ficam vazios.

F) Dois vetores são retirados da pilha, mas representam variáveis diferentes. Usaremos como exemplo as variáveis A e B operadas por *.

O campo **valor** da variável elemento1, com **oper1** igual a 'A' e **tipo** igual a 't', pode ser visto pela tabela 3.11 e o campo **valor** do elemento2 com **oper1** igual a 'B' e **tipo** igual a ' ', pode ser visto na tabela 3.12. Depois de fazer uma chamada ao

procedimento **opera** o qual retorna a variável **retorno**, encontramos no campo **valor** da variável **retorno** os valores vistos na tabela 3.13 (apenas os valores das células sombreadas). Nos campos **tipo** temos agora o caracter 'm', indicando que a tabela é uma operação entre duas variáveis. Em **oper1** temos o valor de **oper1** do **elemento1** igual a 'A' e em **oper2** o valor de **oper1** do **elemento2** igual a 'B'.

1	2	3	0

Tabela 3. 11 Valores do campo valor para elemento1 do caso f

0	1	2	3

Tabela 3.12 Valores do campo valor para elemento2 do caso f

$\overline{A}$ α B		B			
		0	1	2	3
A	1	0	1	1	1
	2	0	1	2	2
	3	0	1	2	3
	0	0	0	0	0

Tabela 3.13 Valores do campo valor para retorno do caso f

G) Neste caso temos como **elemento1** um vetor, como exemplo a variável A e como **elemento2** a matriz como resultado de operação entre as variáveis A e B. Inicialmente os campos **valor** para **elemento1** e para **elemento2** podem ser vistos nas tabelas 3.14 e 3.15 (apenas os valores sombreados). No campo **oper1** do **elemento1**

temos o caracter 'A' e nos campos **oper1** e **oper2** do **elemento2** temos os caracteres 'A' e 'B', respectivamente.

O primeiro passo é uma chamada ao procedimento **Cria_mat** para o **elemento1** o qual retorna uma variável onde o campo **valor** possui os valores vistos na tabela 3.16.

0	1	2	3

Tabela 3.14 Valores do campo valor para elemento1 (A) do caso g

AαB		B			
		0	1	2	3
A	1	0	1	1	1
	2	0	1	2	2
	3	0	1	2	3
	0	0	0	0	0

Tabela 3.15 Valores do campo valor para elemento2 do caso g

0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3

Tabela 3.16 Valores do campo valor para A após cria_mat do caso g

Como o **oper1** do **elemento1** é igual ao **oper1** do **elemento2** é feita uma chamada ao procedimento **opera_m**, o qual retorna uma variável que é armazenada em **retorno**. Esta variável possui em seu campo **valor** os valores vistos na tabela 3.17; no campo **tipo** o caracter 'm', indicando uma matriz, e nos campos **oper1** e **oper2**, os caracteres 'A' e 'B'.

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**0**	**0**	**0**

Tabela 3.17 Valores do campo valor retorno do caso g

H) O **elemento1** é um vetor referente a C topo, ou seja, possui no campo **valor** os valores da tabela 3.18, no campo **tipo**, o caracter 'v' e no campo **oper1** 'C'. O **elemento2** é uma matriz, possui no campo **valor** os valores contidos na tabela 3.20, em **tipo** 'm', nos campos **oper1** e **oper2** os caracteres 'A 'e 'B'.

1	**2**	**3**	**0**

Tabela 3.18 Valores do campo valor para elemento1 (C topo) do caso h

		B			
		0	**1**	**2**	**3**
A	**1**	**0**	**0**	**0**	**0**
	2	**0**	**1**	**1**	**1**
	3	**0**	**1**	**2**	**2**
	0	**0**	**0**	**0**	**0**

Tabela 3.19 Valores do campo valor para elemento2 do caso h

É feita uma chamada ao procedimento **cria_mat** para o **elemento1** retornando uma matriz onde os valores podem ser vistos na tabela 3.20.

Como estamos trabalhando com três variáveis A, B e C é feita uma chamada ao procedimento **opera_m_3**. Este procedimento retorna quatro tabelas (nível quaternário), que são armazenadas na pilha. É criada uma matriz auxiliar chamada

temp para cada linha do campo valor do **elemento1** transformado em matriz. Para exemplificar o conteúdo do campo **valor** da matriz **temp** podemos observar a tabela 3.21.

1	**1**	**1**	**1**
2	**2**	**2**	**2**
3	**3**	**3**	**3**
0	**0**	**0**	**0**

Tabela 3.20 Valores do campo valor para elemento1 (C topo) do caso h

1	**1**	**1**	**1**
1	**1**	**1**	**1**
1	**1**	**1**	**1**
1	**1**	**1**	**1**

Tabela 3.21 Valores do campo valor para temp da primeira linha do elemento1 do caso h

Dentro do procedimento **opera_m_3** são feitas quatro chamadas ao procedimento **opera_m** para cada uma das tabelas auxiliares e o **elemento1**. Na primeira tabela resultante de **elemento2** e a matriz gerada para o **elemento1** temos no campo **valor** os valores vistos na tabela 3.22. Essa primeira tabela será o resultado do **elemento2** e a tabela de C valendo. O campo **tipo** tem o caracter '1', indicando que é a primeira tabela do resultado de uma operação com três variáveis. Nos campos **oper1**, **oper2** e **oper3** temos os valores 'A', 'B' e 'C', respectivamente. Para as demais tabelas o procedimento é o mesmo, e somente os campos **valor** e **tipo** são diferentes. Os

campos **valor** para a segunda, terceira e quarta tabelas com **tipos** '2', '3' e '4', respectivamente, podem ser vistos nas tabelas 3.23, 3.24 e 3.25.

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**1**	**1**
0	**0**	**0**	**0**

Tabela 3.22 Valores do campo valor para retorno tipo '1' do caso h)

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**0**	**0**	**0**

Tabela 3.23 Valores do campo valor para retorno tipo '2' do caso h

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**0**	**0**	**0**

Tabela 3.24 Valores do campo valor para retorno tipo '3'do caso h

0	**0**	**0**	**0**
0	**0**	**0**	**0**
0	**0**	**0**	**0**
0	**0**	**0**	**0**

Tabela 3.25 Valores do campo valor para retorno tipo '4' do caso h

Na segunda tabela (tabela 3.22) temos o resultado da matriz do **elemento2** com uma matriz de C valendo 2. Assim como para as tabelas 3.24 e 3.25, temos o resultado da matriz do **elemento2** com as matrizes de C valendo 3 e 0.

I) O **elemento1** e o **elemento2** possuem em seus campos **tipos** 'm', estamos trabalhando com duas matrizes. Nos campos **oper1** e **oper2** do **elemento1** temos A e B, e nos campos **oper1** e **oper2** do **elemento2** também. Estamos trabalhando com duas variáveis. Podemos verificar os valores dos campos **valor** para **elemento1** pela tabela 3.26 (AαB) e para o **elemento2** pela tabela 3.27 (At δ B).

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.26 Campo valor para elemento1 do caso i

0	**1**	**1**	**3**
0	**1**	**2**	**3**
3	**3**	**3**	**3**
0	**0**	**0**	**3**

Tabela 3.27 Campo valor para elemento2 do caso i

É feita uma chamada ao procedimento **opera_m** que retorna uma variável que será armazenada em **retorno**. No campo **tipo** de retorno temos o caracter 'm', indicando uma matriz. Nos campos **oper1** e **oper2** temos as variáveis 'A' e 'B'; e, no

campo **valor** (tabela 3.28) a tabela resultante da operação entre **elemento1** e **elemento2** pelo operador α.

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**0**	**0**	**3**

Tabela 3.28 Campo valor para retorno do caso i

J) Como **elemento1** temos a matriz referente a AαB e como **elemento2** temos a matriz referente a AδC que devem ser operadas por α. No campo **valor** do **elemento1** temos os dados apresentados na tabela 3.29 e no campo **valor** do **elemento2** da tabela 3.30 Nos campos **tipos** dos dois elementos temos 'm'. No campos **oper1** do **elemento1** temos 'A' e no campo **oper2** temos 'B'. Para o **elemento2** temos nos campos **oper1** 'A' e **oper2** 'C'.

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.29 Campo valor para elemento1 do caso j

0	**0**	**0**	**3**
0	**1**	**1**	**3**
0	**1**	**2**	**3**
3	**3**	**3**	**3**

Tabela 3.30 Campo valor para elemento2 do caso j

É feita uma chamada ao procedimento **opera_m_3**, dentro do procedimento são feitas chamada ao procedimento **cria_mat** para cada uma das linhas do campo **valor** do **elemento2** gerando para cada uma das linhas uma tabela auxiliar chamada **temp**. Os valores das tabelas geradas podem ser vistos nas tabelas 3.31, 3.32, 3.33 e 3.34.

0	**0**	**0**	**0**
0	**0**	**0**	**0**
0	**0**	**0**	**0**
3	**3**	**3**	**3**

Tabela 3.31 Campo valor para temp - primeira linha do elemento2 do caso j

0	**0**	**0**	**0**
1	**1**	**1**	**1**
1	**1**	**1**	**1**
3	**3**	**3**	**3**

Tabela 3.32 Campo valor para temp - segunda linha do elemento2 do caso j

0	**0**	**0**	**0**
1	**1**	**1**	**1**
2	**2**	**2**	**2**
3	**3**	**3**	**3**

Tabela 3.33 Campo valor para temp - terceira linha do elemento2 do caso j

3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3

Tabela 3.34 Campo valor para temp - quarta linha do elemento2 do caso j

Para cada uma das tabelas auxiliares é feita uma chamada ao procedimento **opera_m** tendo como parâmetros o **elemento1** e **temp**. Como resultados dessas operações temos elementos com campo **tipo** iguais a '1', '2', '3' e '4'. Nos campos **oper1**, **oper2** e **oper3** temos as variáveis 'A', 'B' e 'C'. Os valores dos campos **valor** para as quatro tabelas geradas podem ser vistos nas tabelas 3.35, 3.36, 3.37 e 3.38.

0	0	0	0
0	0	0	0
0	0	0	0
0	1	2	3

Tabela 3.35 Campo valor para retorno com tipo '1' do caso j

0	0	0	0
0	1	1	1
0	1	1	1
0	1	2	3

Tabela 3. 36 Campo valor para retorno com tipo '2' do caso j)

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.37 Campo valor para retorno com tipo '3' do caso j

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.38 Campo valor para retorno com tipo '4' do caso j

K) Temos como **elemento1** uma variável com **tipo** igual a '4', então são retiradas da pilha mais três variáveis, que são armazenadas em **elemento3**, **elemento4** e **elemento7**, com **tipo** iguais a '3', '2' e '1', respectivamente. Se temos em **elemento2** um vetor, com **tipo** igual a 'v' e **oper1** igual a 'A' o procedimento a ser tomado é idêntico ao caso j. São feitas as tabelas auxiliares e chamadas ao procedimento **opera_m** para cada uma das tabelas de cada linha do campo **valor** do **elemento2**.

Se, o **elemento2** for uma matriz, com **tipo** igual a 'm' e campos **oper1** e **oper2** iguais a 'A' e 'B'. É feita uma comparação entre os campos **oper1** e **oper2** do **elemento2** com os campos **oper1**, **oper2** e **oper3** do **elemento1** para verificar se a matriz do **elemento2** deve ser transposta ou não. Caso seja necessário, é feita uma chamada ao procedimento transposta. O resto do procedimento é idêntico ao caso k com chamadas ao procedimento **opera_m** tendo como parâmetros o **elemento1** e cada uma das tabelas auxiliares do **elemento2**.

L) O **elemento1** retirado da pilha tem **tipo** igual a '4' , então mais três tabelas devem ser retiradas da pilha e armazenadas em **elemento3**, **elemento4** e **elemento7**, respectivamente. O **elemento2** também possui **tipo** igual a '4', então mais três tabelas devem ser retiradas da pilha e armazenadas em **elemento5**, **elemento6** e **elemento8**. Nas tabelas 3.39, 3.40, 3.41 e 3.42 podemos observar os valores para o campo **valor** das variáveis **elemento7** (**tipo** '1'), elemento4 (**tipo** '2'), elemento3 (**tipo** '3') e **elemento1** (**tipo** '4'), respectivamente.

0	**0**	**0**	**0**
0	**0**	**0**	**0**
0	**0**	**0**	**0**
0	**1**	**2**	**3**

Tabela 3.39 Campo valor para elemento7 com tipo '1' do caso I

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**1**	**1**
0	**1**	**2**	**3**

Tabela 3. 40 Campo valor para elemento4 com tipo '2' do caso I

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.41 Campo valor para elemento3 com tipo '3' do caso I

0	**0**	**0**	**0**
0	**1**	**1**	**1**
0	**1**	**2**	**2**
0	**1**	**2**	**3**

Tabela 3.42 Campo valor para elemento1 com tipo '4' do caso I

O conteúdo dos campos **valor** para as variáveis **elemento8** (tipo '1'), **elemento6** (tipo '2'), **elemento5** (tipo '3') e **elemento2** (tipo '4') podem ser vistas nas tabelas 3.43, 3.44, 3.45 e 3.46.

1	**1**	**1**	**1**
1	**1**	**1**	**1**
1	**1**	**1**	**1**
1	**2**	**3**	**0**

Tabela 3.43 Campo valor para elemento8 com tipo '1' do caso I)

1	**1**	**1**	**1**
1	**2**	**2**	**2**
1	**2**	**2**	**2**
1	**2**	**3**	**0**

Tabela 3. 44 Campo valor para elemento6 com tipo '2' do caso I

1	1	1	1
1	2	2	2
1	2	3	3
1	2	3	0

Tabela 3.45 Campo valor para elemento5 com tipo '3' do caso I

1	1	1	1
1	2	2	2
1	2	3	3
1	2	3	0

Tabela 3.46 Campo valor para elemento2 com tipo '4' do caso I

Para as variáveis **elemento7** e **elemento8**, as duas tabelas com **tipo** iguais a '1', é feita uma chamada ao procedimento **opera_m**, retornando uma variável **retorno** com **tipo** também igual a '1', e com **oper1**, **oper2** e **oper3** iguais aos valores dos campos referentes ao **elemento7**. A chamada ao procedimento **opera_m** é feita para as variáveis **elemento4** e **elemento6**, retornando uma variável **retorno** com **tipo** igual a '2' e **oper1**, **oper2** e **oper3** iguais aos do **elemento4**. Outra chamada é feita para as variáveis **elemento3** e **elemento5** , retornando uma variável **retorno** com **tipo** igual a '3' e **oper1**, **oper2** e **oper3** iguais ao do **elemento3**; e, finalmente, uma chamada para **elemento1** e **elemento2** retornando a variável **retorno** com **tipo** igual a '4' e **oper1**, **oper2** e **oper3** iguais ao do **elemento1**.

Os valores para os campos **valor** das variáveis de **retorno** com tipos iguais a '1', '2', '3' e '4' podem ser vistas nas tabelas 3.47, 3.48, 3.49 e 3.50, respectivamente.

0	0	0	0
0	0	0	0
0	0	0	0
0	1	2	0

Tabela 3.47 Campo valor para retorno com tipo '1' do caso l

0	0	0	0
0	1	1	1
0	1	1	1
0	1	2	0

Tabela 3. 48 Campo valor para retorno com tipo '2' do caso l

0	0	0	0
0	1	1	1
0	1	2	2
0	1	2	0

Tabela 3.49 Campo valor para retorno com tipo '3' do caso l

0	0	0	0
0	1	1	1
0	1	2	2
0	1	2	0

Tabela 3.50 Campo valor para retorno com tipo '4' do caso l

CAPÍTULO 4

MANUAL DE UTILIZAÇÃO DO PROGRAMA

4.1 Instalação do software ELOmv

A criação dos discos de instalação foi feita através do programa InstallShield Express, ferramenta criada especificamente para gerar discos de instalação, viabilizada no CD do Delphi 4.0.

A utilização dessa ferramenta é bem simples; apenas devem-se completar os itens pedidos no processo, tais como, nome do arquivo executável, ícone representativo do programa, arquivos adicionais ao funcionamento do programa (arquivo de Ajuda, figuras, ícones), tipo de disco para instalação, etc.

Completado todos os itens necessários do CheckList, o programa gera os discos de instalação.

Para o programa ELOmv, o software InstallShield Express, gerou dois discos de instalação, com capacidade de 1.44 Mb cada um.

Para a instalação do programa, deve ser feita uma cópia dos discos e ao abrir o arquivo setup.exe, o processo de instalação é iniciado. Uma tela de apresentação aparece mostrando o logotipo criado para o programa, como podemos ver na figura 4.1.

CAPÍTULO 4 – MANUAL DE UTILIZAÇÃO DO PROGRAMA

Figura 4.1 Logotipo do programa ELOmv

Para uma instalação sem problemas, o usuário deve colocar corretamente os dados onde o programa será copiado (unidade e diretório corretos), assim como o local no menu iniciar onde o nome do programa será mostrado. O programa executável é copiado para uma pasta ELOmv, juntamente com os demais arquivos necessários para a execução do programa. O tamanho do arquivo executável é de 764 Kb e da pasta ELOmv completa de 1,21 Mb.

4.2 Como usar o programa

Para abrir o programa o usuário pode usar o ícone do programa ELOmv, visto na figura 4.2, ou utilizar o menu Iniciar, Arquivos de Programas, ELOmv, como podemos observar na figura 4.3.

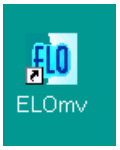


Figura 4.2. Ícone do programa ELOmv

Ao iniciar o programa, a tela de apresentação aparece e o usuário deve clicar no botão **Entra**, mostrado na figura 4.4. Ao clicar no botão **Entra** da tela de apresentação,

a tela de entrada (figura 4.5) aparece, com os botões de variáveis, parênteses, valores 0, 1, 2 ou 3, deslocamentos e operadores.

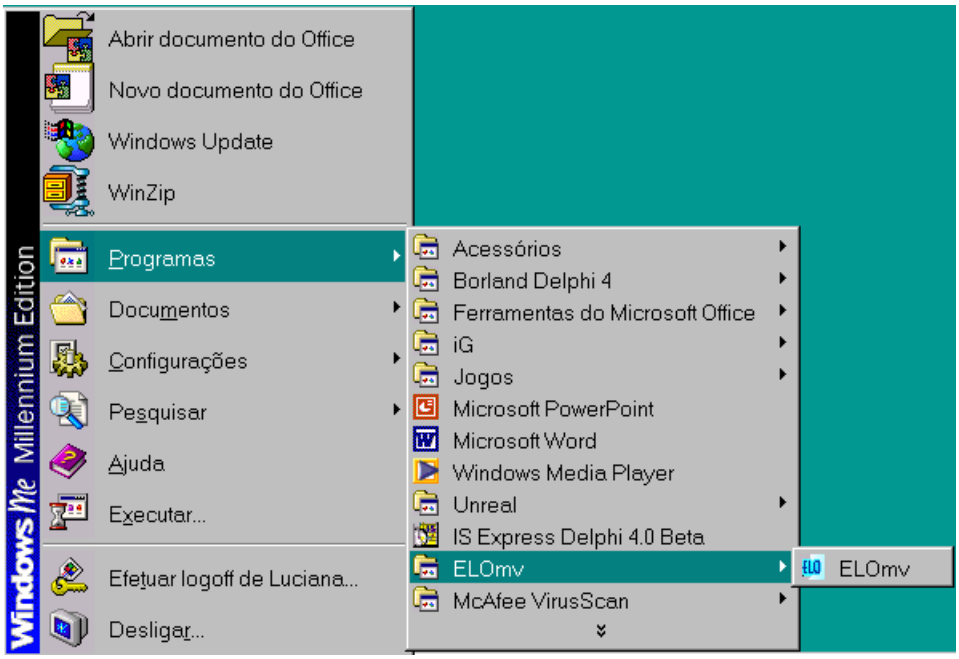


Figura 4.3. Menu Iniciar

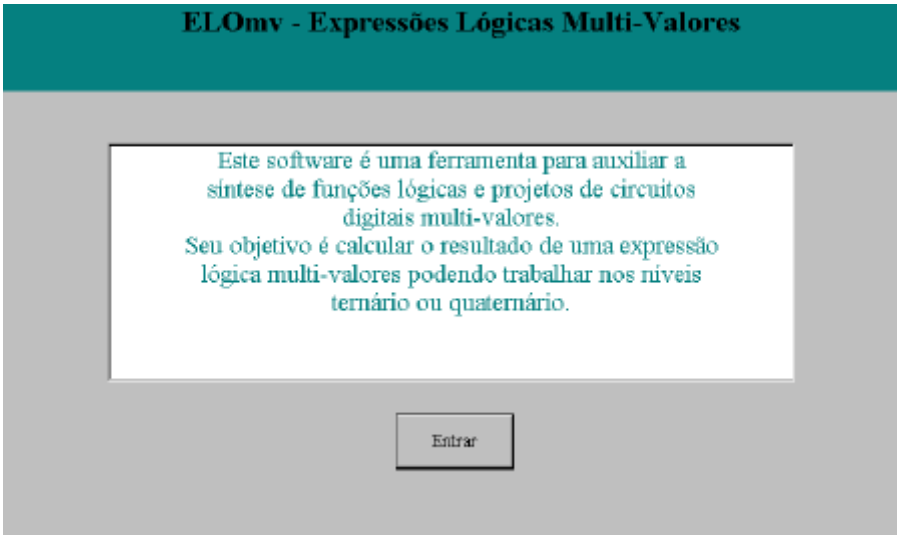


Figura 4.4. Tela de apresentação

O usuário pode utilizar os botões para entrar com uma expressão ou ainda usar diretamente o teclado.

Os parênteses servem para dar prioridades às operações, no exemplo mostrado por 4.1, os parênteses dão prioridade à operação β.

$$(A\alpha(A\beta B)) \tag{4.1}$$

Se for necessário entrar com outro nome para alguma variável o usuário pode digitar na caixa de texto um nome para essa variável desde que esta esteja em letra maiúscula.

Uma observação importante para o usuário é que os caracteres α β, γ e δ são substituídos na caixa de texto pelos caracteres *,+,- e /, respectivamente, para o usuário poder utilizar o teclado. Os deslocamentos são substituídos na caixa de texto por t (Topo), b (Base) e d (Duplo).

Para entrar com um deslocamento de uma variável o usuário deve colocar o deslocamento logo após o nome da variável como podemos observar em 4.2., onde o caracter 't' indica um deslocamento Topo para a variável A.

$$(B\alpha At) \tag{4.2}$$

Se o deslocamento for referente a uma operação, como por exemplo, β Topo, então o usuário pode colocar o deslocamento logo após o caracter 'β', ou ainda colocar o deslocamento após o fecha parênteses que indica a operação que sofrerá o deslocamento. O exemplo visto em 4.3, indica que a operação α sofre um deslocamento duplo, indicado pelo caracter 'd' após o fecha parênteses.

$$(A\alpha Ct)d \tag{4.3}$$

O programa foi feito para calcular expressões lógicas multi-valores nos níveis ternário e quaternário, podendo-se entrar com até três variáveis diferentes.

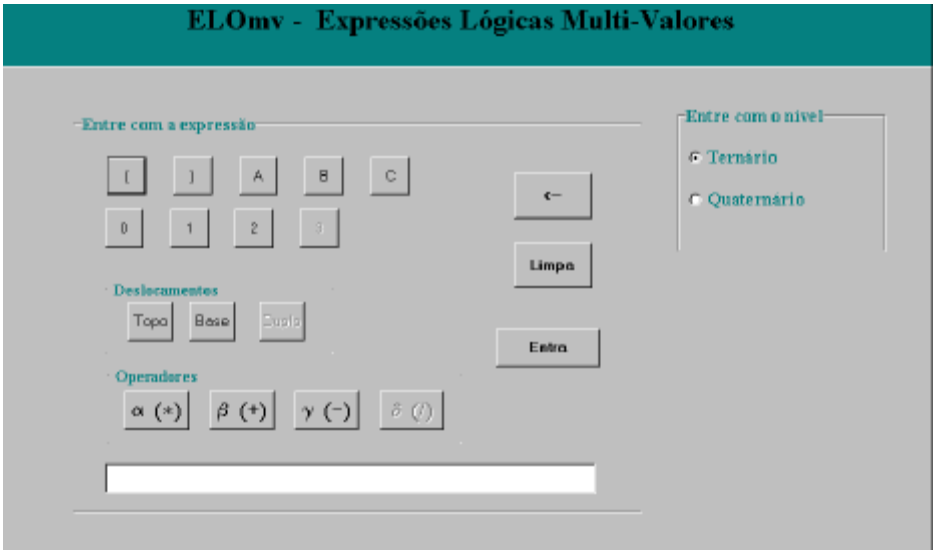


Figura 4.5. Tela de entrada

Após entrar com a expressão, o usuário pode clicar no botão **Entra** da tela de entrada (figura 4.5), pressionar a tecla Enter do teclado, ou utilizar a opção **Executar** do Menu Executar (figura 4.6), ou ainda, o botão **Executar** na barra de ferramentas Executar (figura 4.7).

Figura 4.6. Menu Executar

Figura 4.7. Barra de ferramentas Executar

Na tela de entrada, o usuário pode utilizar o botão **Limpa** (figura 4.5), para apagar todo o conteúdo da caixa de texto, ou usar a opção **Executar|Limpar** ou ainda o botão **Limpar** da barra Executar (figura 4.7).

Para apagar apenas um caracter da caixa de texto o usuário pode usar o botão implementado ←, que pode também ser visto na figura 4.5.

Na tela de saída tanto para três (figura 4.8) quanto quatro (figura 4.9) valores, temos três botões principais. O botão **Nova Expressão**, que volta para a tela de entrada para que o usuário possa entrar com uma nova expressão. O botão **Editar Expressão**, que também retorna para a tela de entrada conservando na caixa de texto a expressão digitada. E para poder imprimir o resultado da expressão na forma que é apresentada na tela, o usuário pode utilizar o botão **Imprimir Tela**.

Após obter o resultado da expressão, o usuário pode gravar o resultado da expressão em um arquivo do tipo texto (.txt), utilizando a opção **Arquivo|Salvar**. E para gravar novos resultados no mesmo arquivo deve continuar utilizando a opção **Salvar** ou o botão **Salvar** na barra de ferramentas Padrão. Para gravar o resultado de uma nova expressão em outro arquivo deve ser utilizada a opção **Arquivo|Salvar Como**.

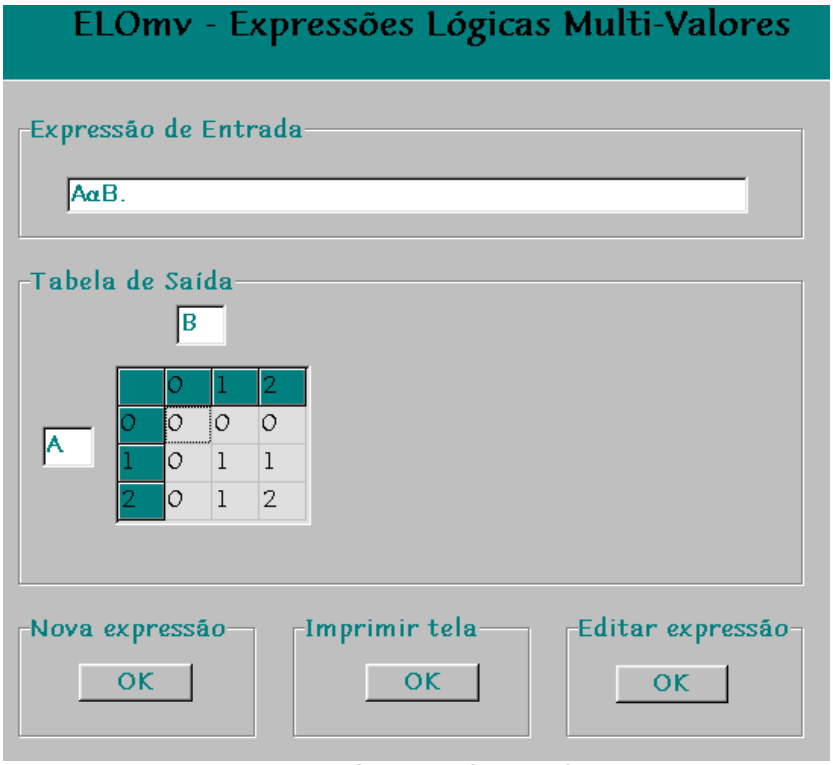


Figura 4.8. Tela de saída para nível ternário

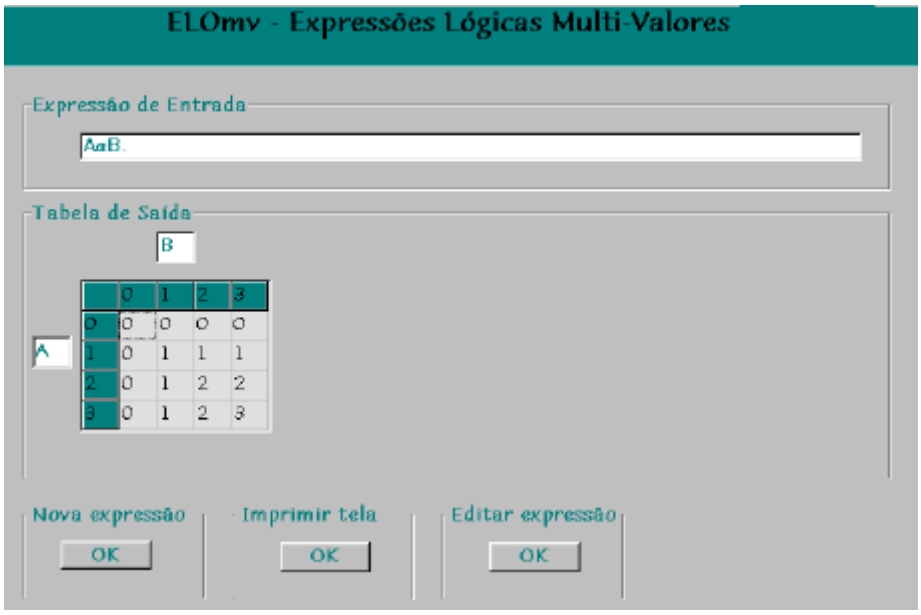


Figura 4.9. Tela de saída para nível quaternário

Para imprimir o arquivo texto, o usuário pode utilizar a opção **Arquivo|Imprimir**, que abre a caixa de diálogo para impressão (figura 4.10), definindo qual impressora será usada para a impressão. Os detalhes de configuração para a impressão, tais como tipo de papel e orientação (retrato ou paisagem), podem ser alterados no menu **Arquivo|Configurar Página**.

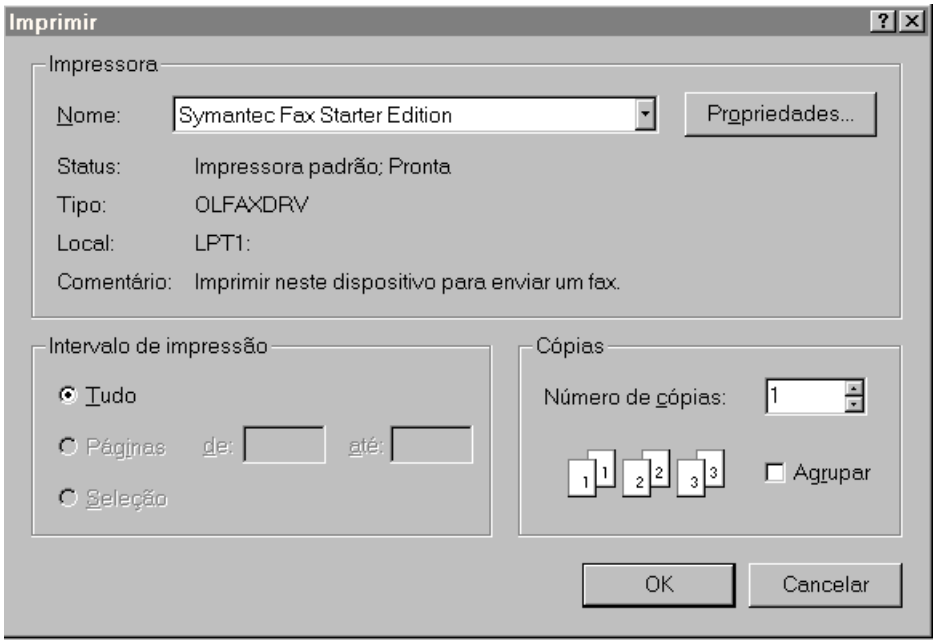


Figura 4.10. Caixa de diálogo para impressão

Outras opções do Menu Arquivo são **Arquivo|Novo**, onde é criado um novo arquivo do tipo texto (txt), **Arquivo|Abrir**, onde a caixa de diálogo Abrir (figura 4.11), permite que um arquivo texto seja aberto, **Arquivo|Fechar**, que fecha o arquivo ou a tela que estiver ativa no momento.

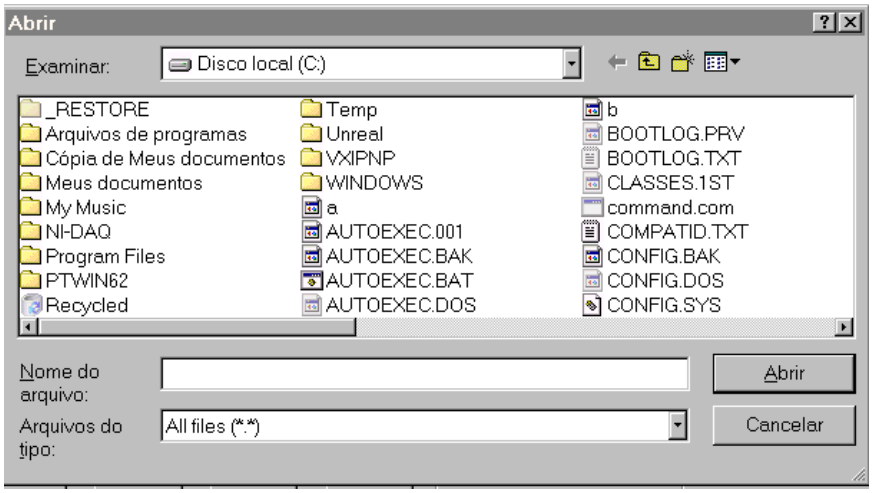


Figura 4.11. Caixa de diálogo Abrir

No menu Editar temos os comandos **Cortar**, **Copiar** e **Colar**, que podem ser usados em qualquer momento da execução do programa.

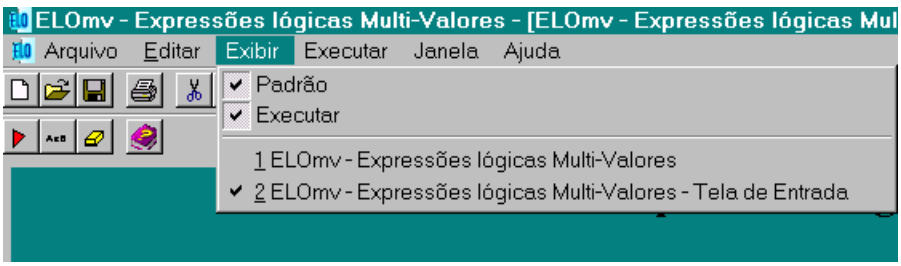


Figura 4.12. Menu Exibir

No menu Exibir (figura 4.12), temos duas caixas de verificação: **Padrão** e **Executar**, referentes às duas barras de ferramentas já comentadas. Essas barras de ferramentas podem estar visíveis ou não, dependendo da escolha do usuário. Todas as telas utilizadas na execução do programa aparecem no menu Exibir. Para escolher entre uma delas, o usuário pode apenas clicar sobre o seu nome no menu. Uma opção disponível, que pode ser útil em alguns casos, quando muitas janelas estão abertas, é a opção do menu **Executar**|**Reinicializar**, que fecha todas as janelas ativas, sem fechar o programa.

4.3 Usando o Arquivo de Ajuda do programa

Para fazer o arquivo de ajuda do programa, foi utilizado o software Helpgen, que cria um arquivo executável.

Dentro do programa ELOmv, o usuário pode acessar o arquivo de Ajuda através do menu **Ajuda|Conteúdo** e **Índice** (figura 4.13) ou o botão **Ajuda** na barra de ferramentas Executar.

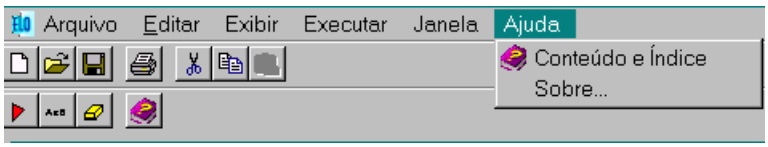


Figura 4.13. Menu Ajuda

A caixa Sobre pode ser vista através do menu **Ajuda|Sobre** e é mostrada na figura 4.14.

A tela inicial do arquivo de ajuda pode ser vista na figura 4.15. O usuário tem a opção de escolher entre um dos índices da **Tabela de Assuntos** ou utilizar a caixa de **Tópicos** (figura 4.16) onde poderá digitar uma palavra e verificar se existe um texto explicativo para ela.

Para cada tópico da **Tabela de Assuntos**, um arquivo descreve procedimentos de como o usuário deve utilizar o programa.

Os arquivos para cada um dos itens são mostrados no Anexo C.

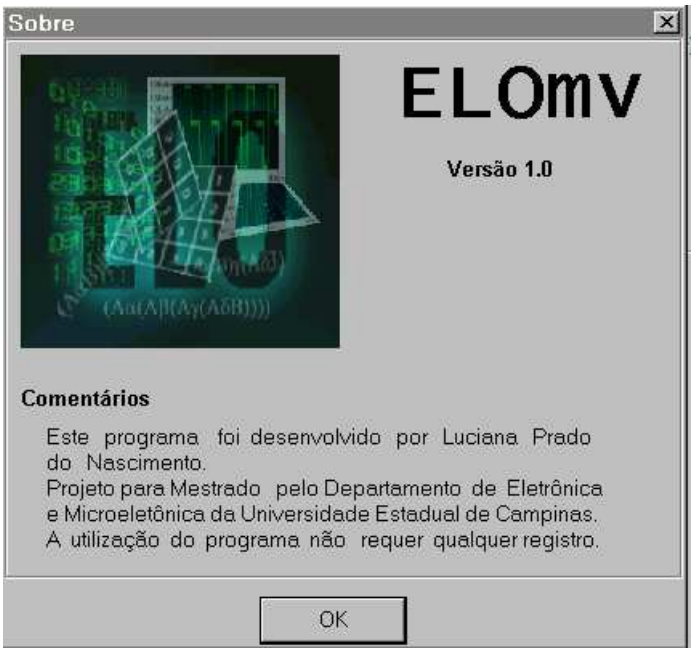


Figura 4.14 Caixa Sobre

Figura 4.15. Tela inicial de Ajuda

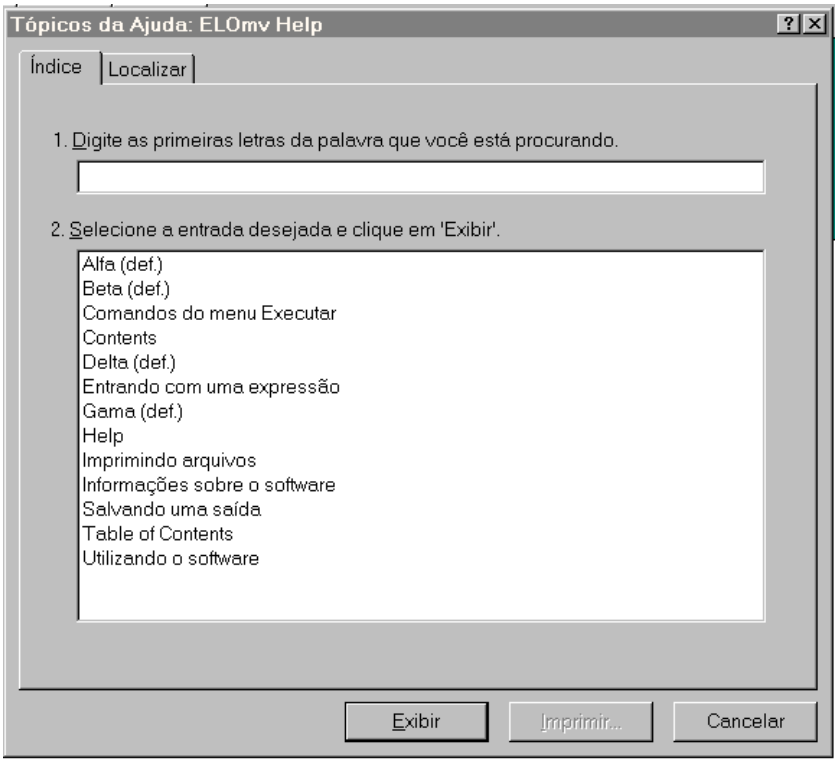


Figura 4.16. Tela de Tópicos de Ajuda

4.4 Resultados e discussões

O resultado esperado do programa foi satisfatório, levando-se em consideração o objetivo inicial. Algumas modificações ainda podem ser feitas para uma melhoria do projeto atual. Entre essas alterações podemos ampliar o programa para trabalhar com mais de três variáveis de entrada, e em outros níveis além dos ternário e quaternário, tornando a ferramenta ainda mais completa.

Qualquer programa desenvolvido necessita de um período de experiência para que observações a respeito de sua funcionalidade sejam feitas. Assim, com a utilização do programa por outros usuários em projetos de circuitos digitais multi-valores podem ser sugeridas modificações, que podem ser alteradas no projeto original, caso se comprove sua necessidade.

CONCLUSÕES

O objetivo inicial do projeto era obter uma ferramenta confiável que pudesse auxiliar no projeto de circuitos digitais multi-valores. Tal objetivo foi atingido, pois o software foi utilizado no projeto do circuito digital ternário para um Somador Completo, no qual foi necessária a comparação entre funções lógicas multi-valores obtidas por métodos diferentes.

O programa se mostrou uma ferramenta muito útil para analisar os termos auxiliares que estão presentes no processo de síntese utilizando o método dos operadores. Testando várias funções lógicas verificou-se que a presença de tais termos é em alguns casos irrelevante, e em outros, necessária.

O programa é capaz de obter as tabelas para as funções lógicas mais complexas utilizadas no estudo de circuitos digitais multi-valores, obtendo uma velocidade mínima de processamento, praticamente igual a zero.

Os usuários que utilizaram o software não encontraram muitas dificuldades. A interface do ambiente de trabalho do programa mantém o padrão Windows®, tão comum à maioria dos usuários. O programa se mostrou como uma ferramenta viável, superando as expectativas em termos de velocidade e facilidade.

É nossa intenção mostrar, em um trabalho futuro, a união com outro software em desenvolvimento, no qual podemos entrar com valores para até três variáveis. Esse software retornará na saída os valores correspondentes em formas de ondas, levando-se em consideração os atrasos correspondentes a cada operação. A saída do programa desenvolvido neste trabalho, em formato de um arquivo texto, servirá como

entrada para as variáveis do software que está sendo desenvolvido. A união dos dois programas será uma ferramenta ainda mais completa.

Em projetos futuros, o programa pode realizar a transformação da função lógica de entrada em uma função que use apenas uma porta básica, como exemplo α Topo. Essa transformação simplifica o projeto de circuitos digitais multi-valores, uma vez que no projeto haverá apenas um tempo de atraso a ser calculado.

Outro trabalho futuro, com fortes motivações a ser desenvolvido é a síntese automatizada de funções lógicas multi-valores. Para realizar tal projeto é necessário um estudo mais aprofundado da álgebra multi-valores e de possíveis simplificações para a lógica mostrada neste trabalho.

RERÊNCIAS BIBLIOGRÁFICAS

[1] Abo, Andy, “CMOS Current Mode Adders”, Andy Abo and Srenik Mehta, EE211.

[2] Biazon, A. J. Filho, “Multi-Valued Half Adder circuit using a CMOS Universal gate” - XV International Conference on Microelectronics and Packaging, Manaus, Brazil, pp. 125-129 - Set. 2000.

[3] Biazon, A J. Filho, Projeto e Construção de uma Porta Universal Cmos em Lógica Ternária, Tese de Mestrado, Universidade Estadual de Campinas, Brasil, 2000.

[4] Demarco, Tom, Structured analysis and system specification, Prentice-Hall software series, New Jersey, 1979.

[5] D. D. Olmsted, “The History and Principles of Multivalued Logic”, 1998.

[6] E. L. Post, “Introduction to a general theory of elementary propositions”, *American Journal of Mathematics*, Vol. 43, pp. 163-185, 1920.

[7] Gane, Chris, Análise estruturada de sistemas, Chris Gane e Trish Sarson; tradução de Gerry Edward Tompkins, Rio de Janeiro: LTC – Livros Técnicos e Científicos, Editora S.A, 1983.

[8] G. Epstein, G. Frieder and D. C. Rine, “The development of Multiple-valued logic as related to computer science”, *Computer,* Vol. 7 (9), pp. 20-32, 1974.

[9] I. Thoidis, D. soudris, I. Karafyllidis, S. Christoforidis and A. Thanailakis, “Quaternary voltage-mode CMOS circuits for multiple-valued logic”, *IEE Proceedings – Circuits Devices Systems,* Vol. 145, No. 2, pp. 71-77, 1998.

[10] J. Lukasiewicz, “O logice trójwartósciowej*”,* Ruch Filozoficzny, english translation: On three valued-logic, in L. Borkowski (ed), *Select Works,* North-Holland, Amsterdam, Vol. 5, pp. 169-171, 1920.

REFERÊNCIAS BIBLIOGRÁFICAS

[11] Jorge, A. Martins, “A Universal CMOS gate for Multi-Valued logic (MVL) circuits” - XV International Conference on Microelectronics and Packaging, Manaus, Brazil, pp. 119-124, Set. 2000.

[12] Jones, Jacqueline A., Problem solving using Turbo Pascal / Jacqueline A. Jones, Keith Harrow, Englewood Cliffs :Prentice-Hall, 1986.

[13] Kruse, Robert L., Data Structures and Program Design, Prentice-Hall, New Jersey, 1987.

[14] M. Cardoli and M. Schaerf, “On the Complexity of Entailment in Propositional Multivalued Logics”, *Annals of Mathematics and Artificial Intelligence”,* Roma, Italy, 1996.

[15] Serran, N. V., A. Martins Jorge and J. A. Siqueira Dias, “A proposal for the implementation of ternary digital circuits”, *Microelectronics Journal*, Vol.28, pp. 533-541, 1997.

[16] Serran, N. V., Ternary digital circuits based on the Post algebra , Tese de Doutorado, Universidade Estadual de Campinas, SP, Brasil, 1996

[17] Turbo Pascal Programmer’s Guide Version 6.0, Borland International, 1990.

[18] Yacoub, M. N. R. D., “Proposta de Implementação de uma Lógica Ternária em tecnologia CMOS” - Tese de Doutorado, Universidade de Campinas, S.P., Brasil, 2000.

ANEXO A

O primeiro problema apresentado foi como armazenar a expressão de entrada digitada pelo usuário, armazenada inicialmente em uma variável do tipo **string**. Para poder armazenar o tipo de deslocamento vinculado a uma determinada variável ou operação foi necessária a criação de um novo tipo de variável. Definida abaixo:

```
type expressa=  valor: char;
                tipo: char;
                end;
```

Para entrar com uma expressão as operações α, β, γ e δ foram primeiramente substituídas pelos caracteres *, +, - e /, respectivamente, pois não havia a possibilidade de se trabalhar em um mesmo objeto com dois tipos de fonte (“times new roman” e “symbol”).

Posteriormente, foi criada uma nova fonte ELO, na qual os caracteres *, +, - e / foram substituídos pelos símbolos α, β, γ e δ. Assim, o usuário pode visualizar melhor a expressão na tela, mas, para depuração do programa, os caracteres substituem os símbolos.

O campo **valor** armazena o nome da variável de entrada (A, B, C...) ou o símbolo correspondente à operação (*, + ,- ,/). E **tipo** armazena o tipo de deslocamento permitido (t, b, d). Onde ‘t’ equivale ao deslocamento unitário (topo), ‘b’ ao deslocamento para a esquerda (base) e ‘d’ ao deslocamento triplo (duplo) se o nível for quaternário.

ANEXO A

Um pequeno algoritmo foi desenvolvido para que a expressão na forma de **string** fosse transformada em um array (vetor) de variáveis do novo tipo definido, **expressa**.

Um segundo problema foi como resolver uma expressão lógica multi-valores. Para expressões algébricas convencionais existem alguns processos eficientes tais como árvore binária, árvore binária balanceada, passagem da forma infixa para posfixa. Analisando os processos básicos foi possível observar que a passagem da expressão na forma infixa para posfixa poderia ser aplicada a expressões lógicas multi-valores.

Com algumas modificações no algoritmo apresentado por Kruse [13] foi possível obter um algoritmo eficiente que pudesse transformar expressões lógicas multi-valores em sua forma infixa (AαB) para sua forma posfixa (ABα).

Um próximo passo foi a elaboração de um algoritmo que através da expressão na forma posfixa pudesse calcular uma tabela simples. Um programa em Pascal foi elaborado independentemente tendo como entrada uma expressão na forma posfixa. Esse programa calculava a tabela para expressões simples, com apenas duas variáveis e um operador (Ex: ABα).

Para armazenar a tabela foi criado um novo tipo de variável. Uma matriz 3x3 de elementos do tipo inteiro.

Ao ler a expressão digitada, o operador é passado como parâmetro para o algoritmo implementado. As variáveis de entrada são transformadas em vetores de até 4 elementos. Para nível ternário um vetor de 0 a 2 e para nível quaternário um vetor de 0 a 3. Esse vetor serve como um índice para o cálculo da tabela.

Dentro do algoritmo uma estrutura do tipo **CASE** (escolha) verifica qual é o operador: α, β, γ ou δ.

ANEXO A

Para cada operador, o algoritmo percorre toda a matriz analisando se nos índices (vetores) correspondentes há um valor principal, se houver, o valor armazenado na célula atual da tabela é o próprio valor principal. Após preencher todas as células com o valor principal, o algoritmo percorre a matriz novamente analisando onde estão os valores secundários nos índices e armazenando o valor secundário nas demais células (é feita uma observação para que as células onde o valor principal já foi armazenado não sejam alteradas). Novamente o algoritmo percorre as células verificando nos índices a presença de um valor ternário (se nível igual a 4) e armazenando nas células correspondentes o valor ternário.

Completando o processo são verificados os índices com valores indiferentes, então as células ainda não preenchidas recebem os valores do segundo índice.

Ao tentar desenvolver o algoritmo um problema foi identificado. Na mesma PILHA utilizada para resolução da expressão deveriam ser armazenados tipos de dados diferentes. Uma simples variável como o vetor A ou uma tabela resultante de uma operação, como AαB. Para solucionar esse problema foi criado um tipo de estrutura que armazena a tabela e um campo tipo que auxilia na identificação do elemento inserido na PILHA.

```
type
    y = record
          valor: array[0..3,0..3] of integer;
          tipo: char;
        end;
```

O campo **tipo** foi definido como **char**, aceita apenas um caracter. Assim se estivermos trabalhando com uma variável, A, o elemento é armazenado na PILHA,

tendo em seu campo **tipo** o caracter 'v', que serve para identificar um vetor. Quando calculamos uma tabela ela é armazenada em um elemento do tipo y, no campo **valor** são armazenados seus dados e em seu campo **tipo** o caracter 'm', indicando uma tabela.

Com o programa para expressões simples, com apenas duas variáveis e um operador, funcionando corretamente, o próximo passo foi integrar o programa transforma (transforma uma expressão de entrada em sua forma infixa para forma posfixa) com o programa que calcula as expressões simples. Gerando um programa simples, que ainda possuía uma interface pobre (**DOS**) e que calculava apenas expressões com um único operador.

Um próximo passo foi implementar o programa para calcular expressões mais complexas. Foi necessário criar um algoritmo que fizesse chamadas aos procedimentos para calcular os diferentes tipos de operações.

Esse algoritmo, denominado **chama_operacao**, tem como princípio básico a retirada de dois elementos da PILHA, e através de várias verificações, descobrir qual tipo de operação deverá ser feita.

Depois de resolver o problema dos tipos de operações que poderiam ser feitas, foi iniciada uma pesquisa de como poderíamos trabalhar com três variáveis. Como seriam calculadas as tabelas e novos algoritmos para geração dessas tabelas. Uma modificação no tipo **y** foi necessária. Observou-se a necessidade de armazenar junto a tabela o nome ou nomes das variáveis que foram operadas. Assim foi necessário a criação de outros três campos : **oper1**, **oper2** e **oper3**. O tipo **y** foi redefinido da seguinte forma:

type

ANEXO A

```
Y =  record
        valor: array[0..3,0..3] of integer;
        tipo: char;
        oper1: char;
        oper2: char;
        oper3: char;
     end;
```

Dessa forma ao calcularmos uma operação como AαB, ou como seria identificado no programa A*B, nos campos **oper1** atribuímos o valor A, e no campo **oper2** o valor da segunda variável B. Como estamos trabalhando com apenas duas variáveis o campo **oper3** fica vazio.

Quando estamos trabalhando com três variáveis o resultado de uma operação nos leva a mais de uma tabela. Se o nível for ternário temos como resultado três tabelas, e se o nível for quaternário temos quatro tabelas. Para podemos identificar esse tipo de resultado, nos campos **tipo** atribuímos os caracteres '1', '2', '3' e '4'. Novos procedimentos para cálculo das tabelas também foram criados e serão descritos posteriormente.

Existem algumas expressões nas quais temos um deslocamento para operações como ($\overline{A\alpha B}$), para implementar esse tipo de deslocamento primeiramente foi definido que a expressão de entrada deveria ser digitada da seguinte forma: (A*tB), o indicador para deslocamento unitário Topo (t) deveria ser colocado logo após a operação. Para calcular o resultado desse tipo de operação, ao final de todo procedimento para calcular as tabelas era feita uma verificação se no campo **tipo** do operador havia um 't', 'b' ou 'd',

então o procedimento **desl_mat** era chamado. Esse procedimento altera os valores da tabela de acordo com o deslocamento encontrado.

Em um exemplo demonstrado podemos observar claramente o que o programa faz:

Dada a expressão ($\overline{A\alpha B}$), através do procedimento opera obtemos a tabela para $A\alpha B$ mostrada em A1.

	A		
B	0	0	0
	0	1	1
	0	1	2

Tabela A1: Operação alfa

Após a chamada ao procedimento **desl_mat** para a tabela $A\alpha B$ com deslocamento unitário 't', obtemos a tabela vista em A2.

	A		
B	1	1	1
	1	2	2
	1	2	0

Tabela A2: Operação com deslocamento

Ao utilizarmos o programa para calcularmos expressões um pouco mais complexas, observamos o freqüente aparecimento de expressões como $\overline{A\alpha B\beta C\delta 2}$. Do modo como estava implementado o usuário deveria transformar essa expressão,

retirando o deslocamento topo, seguindo as normas da lógica multi-valores antes de entrar com a expressão. Uma tarefa que não se enquadra em uma ferramenta utilizada para “automatizar” um processo. Uma solução encontrada foi o modo como o usuário entra com o deslocamento referente às operações. Para definir um deslocamento para uma operação simples ou composta, a expressão de entrada deve estar entre parênteses e o deslocamento desejado deve ser colocado logo após o fecha-parênteses. Podemos observar como a entrada ficou mais clara e intuitiva para o usuário, observando o seguinte exemplo: Dada a seguinte expressão (AαBβCγ0) sua forma de entrada seria (AγBαCβ2)

E sua forma dentro do programa Ab-Bb*Cb+2. Utilizando a solução obtida a entrada no programa fica da seguinte forma (A*B+C-0)b. Essa proposta traz uma maior facilidade para que uma expressão com deslocamento seja dada como entrada no programa e não requer a transformação que antes era necessária.

Todo esse novo processo causou algumas modificações no programa. Primeiro no procedimento para transformar a expressão na forma infixa para a forma posfixa. Quando encontramos um deslocamento ('t', 'b' ou 'd') após um fecha parênteses ')', esse deslocamento será armazenado no campo **tipo** para a variável que tem em seu campo valor ')'. No procedimento para calcular a expressão uma pequena modificação foi feita. Dentro do **CASE** que verifica que tipo de valor foi retirado da PILHA, uma nova opção foi inserida: 't', 'b' ou 'd’, fazendo uma chamada ao procedimento **desl_mat** para o último valor armazenado na PILHA e colocando novamente esse elemento na PILHA.

ANEXO B

O nome do projeto do programa ELOmv é “mdiapp” e ele possui **Units** (arquivos) e **Forms** (arquivos de tela). Estão listados os principais arquivos que formam o projeto:

1. **Main**: **Unit** principal
2. **Form1** (**Unit1**): Tela de abertura
3. **Form4** (**Unit4**): Tela de entrada
4. **Form5** (**Unit5**): Tela de saída para três níveis
5. **Form6** (**Unit6**): Tela de saída para quatro níveis

Dentro desses arquivos existem os procedimentos (**procedures**), e aqui serão mostrados os principais procedimentos de cada **Unit** e uma breve explicação de seus algoritmos.

1. **Main**:
   - **procedure FileNew1Execute(Sender: TObject);**

     Abre caixa de diálogo para criar um novo arquivo texto.
   - **procedure FileOpen1Execute(Sender: TObject);**

     Abre caixa de diálogo para abrir um arquivo texto.
   - **procedure FileCloseItemClick(Sender: TObject);**

     Fecha o arquivo texto ou **Form** (tela) ativa
   - **procedure FileSaveItemClick(Sender: TObject);**

     Salva o resultado da expressão no arquivo previamente gravado. Se não houve uma gravação anterior, então a caixa de diálogo para salvar arquivo é aberta.

ANEXO B

- **procedure FileSaveAsItemClick(Sender: TObject);**

  Abre uma caixa de diálogo para salvar o resultado da expressão em um arquivo do tipo texto.

- **procedure Print1Click(Sender: TObject);**

  Abre caixa de diálogo para impressão e imprime o arquivo texto desejado.

- **procedure PrintScreenclick(Sender: TObject);**

  Imprime a tela de saída ativa.

- **procedure PrintSetup1Click(Sender: TObject);**

  Abre a caixa de diálogo para configurar impressora

- **procedure Exit1Click(Sender: TObject);**

  Fecha todas as janelas abertas e o aplicativo

- **procedure Run1Click(Sender: TObject);**

  Só estará disponível quando **Form4** (Tela de entrada) estiver aberta. Executa o cálculo da expressão dada e chama a tela de saída.

- **procedure Newexpression1Click(Sender: TObject);**

  Abre a tela de entrada para calcular nova expressão

- **procedure Clear1Click(Sender: TObject);**

  Só estará disponível quando **Form4** (tela de entrada) estiver ativa. Limpa o campo onde a expressão de entrada deve ser digitada.

- **procedure Reset1Click(Sender: TObject);**

  Fecha todas as telas ativas, deixando o aplicativo aberto.

- **procedure HelpAbout1Execute(Sender: TObject);**

  Abre caixa de diálogo Sobre.

2. **Unit1**: Esta tela é apenas uma tela de abertura, onde o usuário recebe uma pequena introdução sobre o objetivo do software.

3. **Unit4**: Nesta tela o usuário poderá entrar com a expressão da qual deseja obter a tabela verdade.

   Procedures:

   - **procedure TForm4.Button14Click(Sender: TObject);**

O botão **Button14** é o botão **Entra**. Serve para calcular a tabela verdade para a expressão dada. Dentro desse procedimento é feita uma chamada para **Entrada.chamatra** (procedimento **chamatra** do arquivo **Entrada.pas**; procedimento que recebe uma string (expressão de entrada) e retorna um array no qual estão colocados todos os caracteres de entrada com os seus respectivos deslocamentos (tipos). Dentro desse procedimento é feita uma chamada para **Transfor.trans**: procedimento **trans** do arquivo **transfor.pas**; neste procedimento é feita a transformação da expressão na forma infixa para a forma posfixa, retornando para a **Unit4**.

Logo após a chamada para **Entrada.chamatra,** é feita uma chamada para **Chamaope.chamada**: procedimento **chamada** do arquivo **Chamaope.** Este procedimento é formado por um CASE, e através desse CASE são feitas chamadas aos procedimentos que calculam as tabelas para cada caso. Esses procedimentos estão dentro do arquivo **Chamaope** e são listados abaixo:

- **procedure CriaIndN(var op:Transfor.expressa; var oper:Pilha2.y);**

  Cria um vetor [n] para o valor lido (0, 1, 2 ou 3).

- **procedure Cria_Ind(var op: Transfor.expressa; var oper:Pilha2.y);**

Cria um vetor [n] para a variável de entrada levando em consideração o seu deslocamento (topo, base, duplo topo). O vetor é armazenado em uma matriz nxn onde os elementos serão armazenados nas células que contêm índice de coluna igual a 1.

- **procedure Cria_mat(var r:Pilha2.y);**

  Cria uma matriz nxn para o parâmetro de entrada. A matriz terá os mesmos valores do vetor, para cada elemento da célula com índice i1, os elementos das células com índices ij, onde 1<j<=n, receberão os valores da célula com índice i1.

- **procedure opera(var op1vet:Pilha2.y; op2vet:Pilha2.y ; op:Transfor.expressa; var r:Pilha2.y);**

  Gera uma matriz nxn, calculando os valores entre op1vet e op2vet, com o operador passado por op. Esse procedimento é chamado quando temos duas tabelas com as mesmas variáveis em op1vet e op2vet.

- **procedure desl_mat(var result:Pilha2.y; t:char);**

  Faz um deslocamento em todos os elementos do campo **mat** da variável **result**, levando em consideração o tipo do deslocamento dado por t.

- **procedure opera_m(a:Pilha2.y; b:Pilha2.y; operador:Transfor.expressa; var result:Pilha2.y);**

  Gera uma matriz com o resultado da operação entre a e b pelo operador, este procedimento é usado quando temos duas variáveis diferentes do tipo vetor.

- **procedure opera_m_3(a : Pilha2.y ; b : Pilha2.y ; operador : Transfor.expressa; var result : Pilha2.y);**

Gera três ou quatro matrizes como resultado da variável a e b pelo operador. Neste caso, as variáveis de entrada são matrizes e possuem em seus campos oper1 e oper2 nomes diferentes. O procedimento é chamado apenas quando estamos trabalhando com três variáveis.

- **procedure opera_v(var op1vet : Pilha2.y ; op2vet : Pilha2.y ; op : Transfor.expressa;var r:Pilha2.y);**

  Gera como resultado entre op1 e op2 operados por op um vetor. Este procedimento é chamado quando estamos trabalhando com dois vetores que possuem em seu campo oper1 o nome da mesma variável.

4. **Unit5** e **Unit6** (**Form5** e **Form6**): São as telas de saída para três e quatro valores, dentro desses arquivos temos os procedimentos dos botões que aparecem na tela: **Nova Expressão**, **Imprimir Tela** e **Editar Expressão**.

   - **procedure Nova:**

     Minimiza a tela de saída e faz com que a tela de entrada apareça novamente limpando a caixa de texto de entrada.

   - **procedure Imprimir:**

     Imprime a **form** que estiver ativa da forma com que se apresenta na tela.

   - **procedure Editar:**

     Minimiza a tela de saída fazendo com que a tela de entrada apareça novamente, conservando na caixa de texto da tela de entrada a expressão digitada.

ANEXO B

ANEXO C

TEXTOS DOS TÓPICOS DO ARQUIVO DE AJUDA DO PROGRAMA ELOMV

1. Utilizando o software

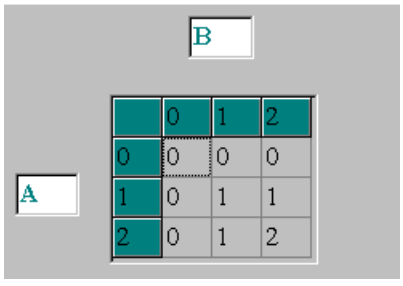
Este software foi desenvolvido com o objetivo de calcular o resultado de uma expressão lógica multi-valores, podendo trabalhar com até três variáveis de entrada, obtendo na saída sua tabela-verdade.

A lógica não clássica utilizada apresenta novos operadores Alfa, Beta, Gama e Delta.

Os operadores criados seguem a seguinte regra, para cada operador têm-se um valor principal, um valor secundário, um valor ternário (se o nível for quaternário) e um valor indiferente.

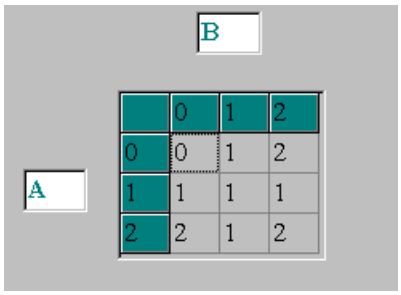
Os exemplos abaixo mostram tabelas paras as operações simples.

Operação Alfa

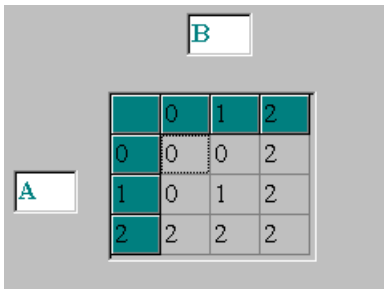


ANEXO C

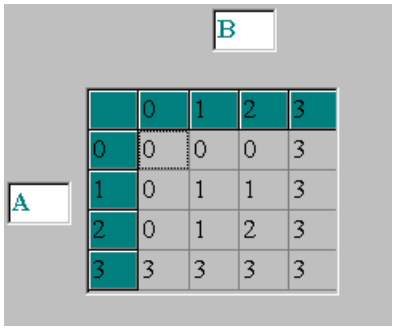
Operação Beta



Operação Gama



Operação Delta

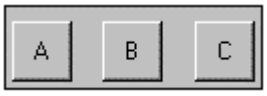


2. Entrando com uma expressão

Ao clicar no botão ENTRA, na tela de abertura, a tela de entrada aparecerá.Para entrar com uma expressão desejada na tela de entrada apresentada digitar a expressão diretamente na caixa de texto ou utilizar os botões apresentados.Para dar prioridade às operações, utilizar os parênteses.

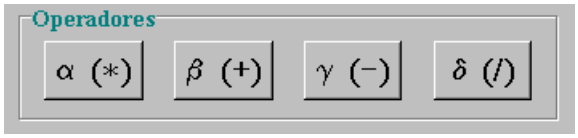
ANEXO C

Para entrar com uma variável pode-se utilizar as variáveis definidas nos botões (A, B ou C).Outro nome de variável pode ser digitado na caixa de texto desde que esteja em letra maiúscula.



Para entrar com um valor numérico utilizar os botões 0, 1, 2 ou 3.O botão 3 é permitido apenas quando nível for quaternário.É utilizado quando o termo Alfa 1, Beta 2 são necessários.

As operações definidas são: Alfa (botão *), Beta (botão +), Gama (botão -) e Delta (botão /). A operação Delta (/) somente estará disponível se o nível for setado para quaternário.



Os deslocamentos definidos para as variáveis ou operações são: Topo (botão t), Base (botão b), ou Duplo Topo (botão d), disponível apenas se nível quaternário.

Os deslocamentos aparecem na caixa de texto apenas com a primeira letra em minúsculo.

O botão Entra calcula o resultado da expressão contida na caixa de entrada, chamando a tela de saída.

ANEXO C

Alguns exemplos de expressões de entrada:

AαB - representa a operação Alfa (*) entre as variáveis A e B.

(AβBt) - representa a operação Beta (+) entre as variáveis A e B Topo (Bt).

(AαBb)t - representa a operação Alfa (*) Topo (t logo após o parênteses) entre as variáveis A e B Base (Bb).

((Aα(BγCd)) - representa a operação Gama (-) entre as variáveis B e C Duplo (Cd) e depois a operação Alfa (*) entre o resultado da primeira operação e a variável A.

3. Comandos do menu executar

O botão Entra calcula o resultado da expressão contida na caixa de entrada, chamando a tela de saída.

O comando Executar somente estará disponível se a tela de entrada estiver ativa. Após entrar com a expressão clicar em Executar|Executar para obter o resultado da expressão contida na caixa de texto.

O comando Limpar somente estará disponível se a tela de entrada estiver ativa.

Limpa todo o conteúdo da caixa de texto onde a expressão está contida.

O comando Nova Expressão abre a tela de entrada para que uma expressão possa ser calculada. Se a tela de saída estiver ativa este comando retornará para a tela de entrada para que um novo cálculo seja feito.

ANEXO C

O comando Reinicializar fecha todas as janelas abertas, sem fechar o programa. Utilizado quando se deseja reiniciar as tarefas dentro do software.

4. Salvando uma saída

Após calcular o resultado de uma expressão a saída obtida pode ser armazenada em um arquivo do tipo texto (.txt). Para salvar em um arquivo utilizar o comando Arquivo|Salvar ou Arquivo|Salvar Como. O comando Arquivo|Salvar e Arquivo|Salvar Como estão disponíveis somente quando a tela de saída estiver ativa.

A saída poderá ser gravada em um novo arquivo, ou vários resultados podem ser gravados em um único arquivo.

5. Imprimindo arquivos

Para imprimir um arquivo de saída utilizar o comando Arquivo|Imprimir.

As opções para configurar página estão disponíveis em Arquivo|Configurar Página.

Uma outra opção disponível é Imprimir Tela, na qual o usuário poderá imprimir a saída no formato apresentado na tela. Para obter essa impressão, o usuário deverá utilizar o comando Arquivo|Imprimir Tela.