A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming

Neucimar J. Leite
Marcelo A. de Barros

Relatório Técnico DCC–95-10

Agosto de 1995

A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming

Neucimar J. Leite*
Marcelo A. de Barros†

August 11, 1995

Abstract

A special purpose cellular processor, based on the Functional Programming approach, is proposed for image processing applications. The basic data flow graph of the processor is defined according to the data structure of a class of nonlinear filters very useful in image processing. The cells of the processor are quite simple and support a certain degree of programmability that allows, for instance, a logical reconfiguration of the network. This flexibility contributes to the execution of a multitude of standard low-level image processing algorithms on the same basic structure.

1 Introduction

Digital image processing is characterized by the bulk of image points (pixels) which leads to a large amount of computational load per image pixel. Advances in VLSI microelectronic technology have provided designers with a powerful tool for creating image processing machines. These machines are really efficient and flexible but tend to be expensive for many applications, due to the amount of hardware required. Another way to face the problem related to the great amount of data in image processing is to define special purpose processors. These processors, concerned with different kinds of applications, can be assembled in a pipeline network to create a complete vision system. One way to reduce the costs of such a system is to consider *flexible processors* which can execute a set of algorithms on the same structure.

We will use the Functional Programming (FP) [1] concept to define a flexible cellular processor. As we will see later, the FP style can be used as a practical mean to express image processing algorithms, and as an efficient tool to map algorithms onto architectures. Thanks to the programming flexibility embedded on its cells, the processor can be logically reconfigured to map a large number of standard low-level image processing algorithms.

*Department of Computer Science, University of Campinas, Cx. Postal 6065, 13081 Campinas - SP, Brazil, e-mail: neucimar@dcc.unicamp.br

†Lab. de Meteorologia, Recursos Hídricos, e Sensoriamento Remoto, Universidade Federal da Paraíba, 58100 Campina Grande, PB - Brazil

This work is organized as follows. Section 2 introduces briefly the interaction between FP and image processing. Section 3 presents the model of the processor cells and the general data structure of the algorithms defining its data flow graph. Section 4 gives some examples of the reconfiguration capabilities of the processor and Section 5 discusses its hardware implementation. Conclusions are drawn in Section 6.

2 Functional programming and data flow graph

As introduced by Backus in [1], the Functional Programming (FP) style is based on the idea of combining simple operations or primitives, for creating more complex ones. Associated with the FP approach, there are some algebraic laws defined to operate on the variables of the program. A program can be seen as a set of functions that map *objects* to *objects* without the introduction of any control mechanism. Moreover, in the FP style, a program can be naturally expressed in an equivalent data flow graph which leads to a dual representation of programs and graphs (architectures) [2].

The most interesting feature of the functional approach is represented by its ability to express complex programs thanks to the simple functions and the rules to associate them [2]. On the other hand, low-level image processing algorithms are characterized by a constant repetition of a set of elementary operations, such as addition, multiplication, maximum, exponentiation, and so on. Then, the functional decomposition aspect inherent to the FP approach can represent a practical and efficient mean to define image processing algorithms. Indeed, by combining the set of elementary operations in a suitable form, we can define more complex functions that can express image-to-image transformations in a very flexible manner.

Based on these concepts we define a modular neighborhood image processor which can implement various low-level image processing algorithms on the same basic data flow graph.

3 The cellular processor

The data flow graph of an algorithm in the FP style [2] is defined by a set of primitive functions represented by *nodes*, a set of *objects* (the pixels of an image, for example), which constitute the arguments of the primitive functions, and dependences between functions indicated by *arcs* "transporting" the objects.

The next sections present the model of the cells (the nodes) that constitute the processor. They also present the general structure of the algorithms defining the basic data flow graph of the network.

3.1 The processor cells

The simplicity of the cells of the processor is an important feature concerned with the costs of the physical implementation of the structure. Considering this aspect and the functional decomposition feature inherent to the low-level image processing algorithms, we define the cells of the processor such that they can execute only two basic operations: a representative

function of the non-linear operations, the function **MAX** (maximum), and a representative function of the linear operations, the function **ADD** (addition). To improve the logical capabilities of the cells we interconnect them in an iconic way by means of weights w_i.

The propagation of information on the cell can be easily represented by a directed graph as in Fig. 1.

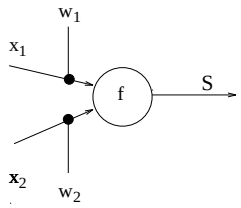

Figure 1: The model of the cells of the processor.

The node in the graph corresponds to the function executed by the cell, and the edges express dependences between functions. The inputs x_i are the arguments or objects of the function **f**. S is the output object corresponding to f(ε, $x_1 w_1$, $x_2 w_2$), ε is the pre-defined state of the cell which indicates the function to be executed (ADD or MAX). The weights w_i are used, for example, to reconfigure the processor and to improve the logical capabilities of the cells, as we will see later.

The function **f** is enabled when the arguments x are present at the input edges of the node. Then the specified function is applied to the weighted inputs of the cell (input x_i multiplied by weight w_i). The result-object is placed on the output edge S.

3.2 A non-linear filter model

To define the data flow graph of the processor we will consider a class of non-linear filters very useful in low-level image processing. These filters operate on a small neighborhood and act uniformly on the image, so that each pixel is replaced by a function of itself and its neighboring elements. These filters are studied in [3], and their general structure is described by the equation below (see [3] for demonstration):

$$y = f\left(\sum_{i=1}^{N} a_i (b_i g(x_i))_{(i)}\right), \tag{1}$$

where y is the output image at the central pixel of the N=MxM neighborhood; $x_i's$ are the image data in the neighborhood; g($\cdot$), f($\cdot$) are pointwise nonlinear functions used for the decoupling of the noise from the image, and a_i, b_i, i=1,...,N, are the filter coefficients. The values $b_i g(x_i)_{(i)}$ correspond to $b_i g(x_i)$ ordered according to their magnitudes (the smallest of them is $b_i g(x_i)_{(1)}$ and the largest of them is $b_i g(x_i)_{(N)}$).

Eq. 1 encompasses many classes of non-linear filters and edges detectors useful in image processing applications. Median filters, order statistic filters, erosion and dilation operations, range edge detectors, dispersion edge detectors, are some representative examples.

The median filter, for instance, can be implemented on Eq. 1 by defining $b_i = 1$, i =1,...,N, and $a_{N/2}$ =1, $a_i = 0$, $i \neq N/2$, with f($\cdot$)=g($\cdot$)= x_i.

We consider here that the functions $f(\cdot)$, $g(\cdot)$ are evaluated outside the processor by using look-up tables, for instance. Nevertheless, many filters (e.g., contraharmonic, median, rank filter, α-trimmed mean, etc) can be directly executed by considering, $f(\cdot)$=$g(\cdot)$=x It is interesting to note that even the convolution linear filter can also be represented by Eq. 1, where $f(\cdot)$=$g(\cdot)$= x, b_i are the coefficients of the filter, and a_i=1, i=1,...,N. All these algorithms have different statistical properties and are used to perform better in a particular kind of noise affecting an image.

3.3 The basic data flow graph of the processor

By analyzing Eq. 1, we can say that the kernel of its data structure is represented by a *sorting* algorithm. Hence, to match this data structure, the data flow graph of the processor will be based on a parallel sorting network.

Several parallel structures for sorting have been presented in the literature [4]. The *odd-even transposition sort* represents a good compromise between area cost and computing time, and has significant advantages, such as the simplicity of the network, which leads to a very uniform structure, and the high degree of parallelism.

Odd-even transposition sort can be described as follows [4]. Let x_1,...,x_n be a sequence of n elements to be sorted. In the odd (respectively, even) step of the algorithm, all element x_i of the sequence having an odd (respectively, even) subscript are compared with their successors and exchanged, in a *comparison-exchanged* operation, if $x_i > x_{i+1}$ (i $\in$ 1,...,n-1). The odd and even steps are executed in alternating order. After at most n steps the sequence is sorted

Using the model of the cells defined above, we can match the basic data flow graph of Eq. 1 as follows. Let **G(V,E)** be a graph where the set of vertices **V** is made up of cells connected by edges **E**, disposed in (N+1) columns and N rows, N being the number of the input values to be sorted. The ith cell on column j is represented by c_{ij}, with $0 \leq i \leq (N-1)$ and $0 \leq j \leq N$. **E** is composed of all edges of the form $(c_{ij}, c_{l,j-1})$ with $0 \leq l \leq (n-1)$ and:

- for j odd: $l = i$ or the binary representations of l and i are the same except the first bit (the least significant one).
- for j even: $l = i$ or the two least significant bits of l and i are different and $|l - i|$ =1.

Fig. 2 shows the data flow graph for N=9 sorting the input data such that the largest one is in the upper rightmost side of the network (for simplicity we omitted the weights w_i). The *comparison-exchange* operation mentioned here can be obtained by a pair of cells executing the functions max and min. The function min can be indirectly defined by programming the weights of the cells, since $S = min(x_j, x_k) = -max(-x_j, -x_k)$.

4 Some examples of reconfiguration

A non-programmable structure such as the one proposed in [3] permits only the execution of algorithms which are closely related to Eq. 1. The model of the cells considered here

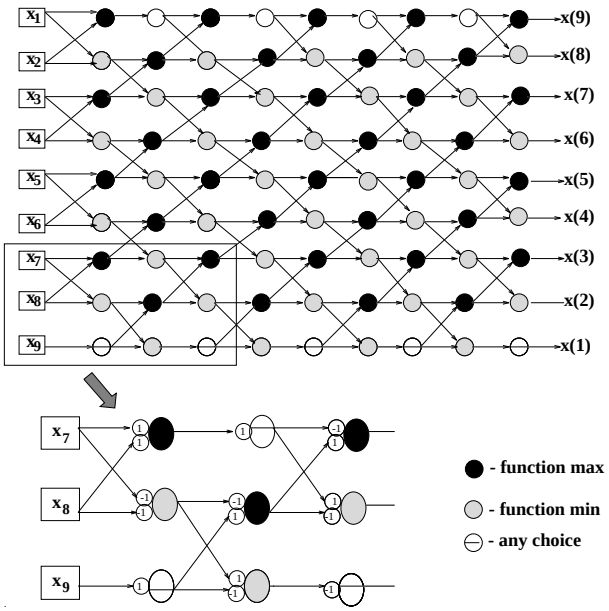

Figure 2: An example of sorting.

supplies the processor with programming and reconfiguration capabilities that allow it to match algorithms different from those included in this equation.

We highlight these aspects by the examples below:

Example 1: **The separable median filter**

This example shows how we can propagate data on the graph to match a particular algorithm.

The basic idea of the separable median filter is to split up a square neighborhood into segments (rows and columns), compute the median of these segments, and finally the median of the medians. Fig. 3 shows the graph of the algorithm for a 3x3 neighborhood. The 3x3 neighborhood considered here has lines A= $\{x_1, x_2, x_3\}$, B= $\{x_4, x_5, x_6\}$, and C= $\{x_7, x_8, x_9\}$

The edges of the cells not involved in the computation have weights $w = 0$. The one-input cells of the graph have weights $w = 1$, on this input, and $w = 0$ otherwise. We can say that these cells define a "short-circuit" on the graph, since all the input values x of the network are positive integers, and in this case $max(0, x_i) = add(0, x_i)$.

Example 2: **A measure of connectivity**

The data flow graph in Fig. 2 can be seen as a *butterfly* network in which a *leaf* node shares two trees. Hence, the operations of *reduction* and *distribution* of data, inherent to the FP style, can be executed in the network by means of trees combining the different primitive functions. Fig. 4 shows the graph of the Yokoi's operator [5] which defines a measure of connectivity used, for example, in algorithms considering the topological characteristics of

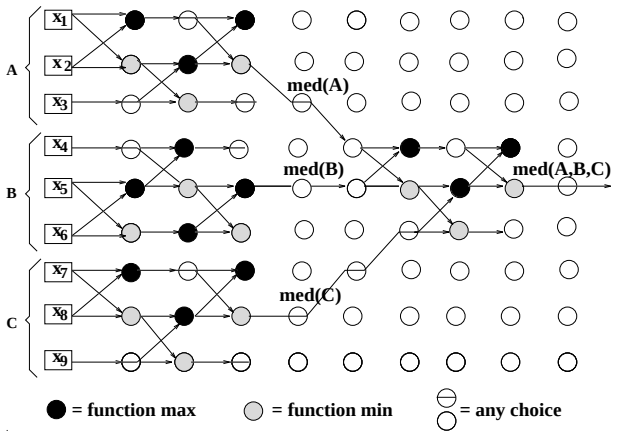

Figure 3: 2-D median filter.

an image. The Yokoi's connectivity number for the 4-connectedness case can be defined as follows:

$$\begin{aligned} CN_4 &= x_6(1 - x_3x_2) + x_2(1 - x_1x_4) + \\ &\quad x_4(1 - x_7x_8) + x_8(1 - x_9x_6) \end{aligned} \tag{2}$$

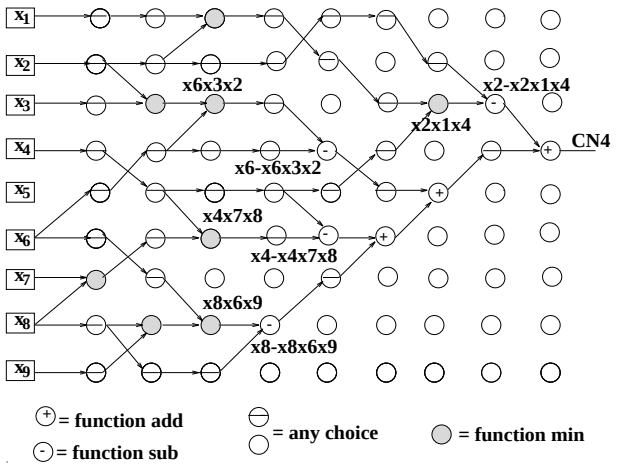

Figure 4: The graph of the Yokoi's operator.

The function SUB (subtraction) is executed on the cell by programming its weights w, such that $S = sub(x_1, x_2) = add(x_1, -x_2)$.

Example 3: **Maximum difference operator**

This image edge detection algorithm, operating on only a 2 by 2 region of pixels at each point, is executed by the graph of Fig. 5. In this case, by combining different trees in

the processor we transform two pixels of the image in parallel. This maximum difference operator is defined as:

$$\begin{aligned} D &= max(x_1, x_2, x_4, x_5) \\ d &= min(x_1, x_2, x_4, x_5) \\ M(x_1) &= D - d \end{aligned} \tag{3}$$

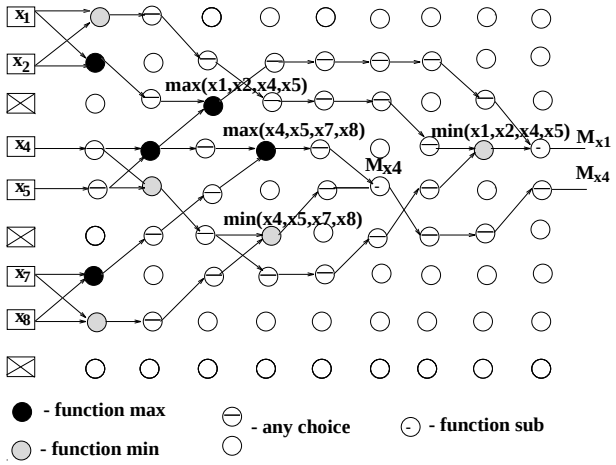

Figure 5: Maximum difference operator algorithm.

We have seen how to program the processor to match some standard low-level image processing algorithms. By considering the primitive functions and the logical flexibility embedded on the network we can execute many useful image transformations such as linear (convolution), nonlinear (represented by Eq. 1) and hybrid filters (e.g., the Wilcoxon filter), binary and gray scale morphological operations (erosion, dilation, thinning and expanding algorithms, hit or miss transformation, morphological edge detectors, gradient, white top hat and black top hat operations [6], etc), binary and gray scale geodesic operations (e.g., geodesic dilation and erosion, image reconstruction, etc), distance transformation and so on. Furthermore, these operations can be combined in parallel or in pipeline to define more complex programs in a modular form.

5 An implementation Approach

A direct implementation of this processor requires $O(\mathrm{N}^2)$ cells, where N is the number of pixels of the neighborhood as defined before. In order to obtain a satisfactory trade-off between area cost, reconfigurability and speed we propose an architecture whose diagram block is depicted in Fig. 6. To maintain the data parallelism inherent to the model we use a column of identical reconfigurable cells interconnected in a linear way by a reconfigurable network. The parallelism represented by the pipeline of columns in the original network is replaced by a sequence of computing steps followed by a configuration step. This *local*

reconfiguration is possible thanks to a fast programming unit which performs the cell configuration using only 4 bits per cell.

The operation of the network is synchronous and determined by the pixel rate. A global clock signal (GCLK), generated by the pixel clock, drives the pixel flow through the input (X(i)), and a local clock signal (LCLK) drives the computing and the configuration functions for each cell of the array. For every pixel clock the parallel data-flow representing a pixel neighborhood is entered, buffered, and processed N times by the vertical array. This generates the output pixels (Y(i)).

This implementation reduces the processor implementation cost from $O(N^2)$ to $O(N)$. The maximum frequency operation, Fpixel, is determined by the maximum internal frequency, Fint (Fpixel = Fint/N). Most of practical applications use small neighborhood (e.g., 3x3 and 5x5), and can be performed under real time constraints.

We developed a prototype of the processor using SRAM based Field Programmable Gate Array - FPGA. A technology independent library composed of a set of representative primitives to describe low level image processing algorithms is under development. This library allows the implementation of the algorithms in a Functional Programming style by exploiting the reconfiguration capabilities of the processor.

The hardware implementation considered here allows both static and dynamic reconfiguration. Static reconfiguration is performed for every new application by loading a control string (a microprogram) on the programming unit (see Fig. 6). The time needed to reconfigure all the array (*global reconfiguration*) is equivalent to N pixel clocks. So, the user can change the functionality of the processor at real time (during processing) without lost of the processed information. Dynamic reconfiguration can be performed since a signal to set a new microprogram loading can be easily generated by another processor in the processing chain. Hence, this architecture is cascadable and several processors can be linked together to construct more complex operators.

6 Conclusions

We have presented a special purpose cellular processor for image processing applications, based on the Functional Programming approach. This approach has been considered as an efficient mean to map image processing algorithms onto regular graphs due to the functional decomposition aspect inherent to these algorithms. The cells of the processor defined here represent a good tradeoff between hardware complexity and logical flexibility. By an easy programming (a visual programming) of the network we can match a large number of standard low-level image processing algorithms on the same basic structure.

References

[1] Backus, J. "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs", Comm. of the ACM, vol.21, n.8, Aug. 1978, pp.613-641.

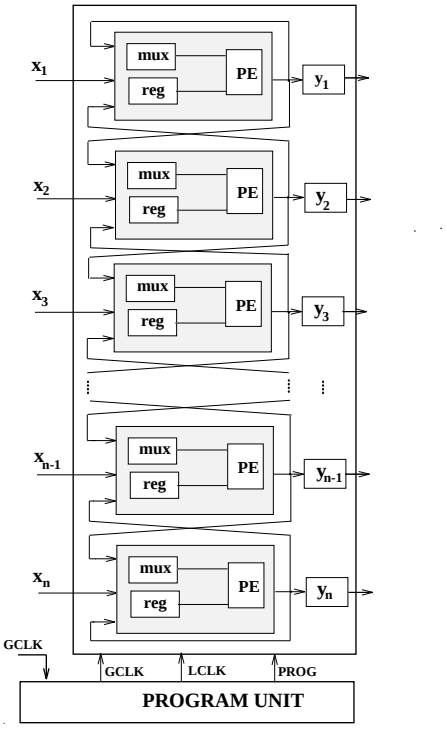

Figure 6: The basic processor structure.

[2] Zavidovique, B. and Allert, E. "Functional Mapping for Low-Level Image Processing Algorithms", in VLSI Signal Processing III, R. W. Brodersen et alli eds., IEEE Press, New York, 1988.

[3] Pitas, I. and Venetsanopoulos, A. N. "A New Filter Structure for Implementation of Certain Classes of Image Processing Operations", IEEE Trans. on Circuits and Systems, vol.35, n.6, June 1988, pp.636-647.

[4] Knuth, Donald E. *Sorting and Searching* in The Art of Computer Programming, vol.3, Addison-Wesley, Boston, 1973.

[5] S. Yokoi et alli, *"An Analysis of Topological Features at Digitized Binary Pictures Using Local features"*, Computer Graphics and Image Processing 4, March 1975, pp. 63-73.

[6] J. Serra, *Image Analysis and Mathematical Morphology: Theoretical Advances*, Prentice Hall, Englewood Cliffs, 1988.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 **$\mathcal{LL}$ – An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`