O Algoritmo KMP através de Autômatos[1]

Marcus Vinícius A. Andrade
Cláudio L. Lucchesi

Relatório Técnico DCC–94-03

O Algoritmo KMP através de Autômatos[†]

Marcus Vinícius A. Andrade[‡] Cláudio L. Lucchesi[§]

11 de abril de 1994

Abstract

In this article we present a description of the KMP algorithm through automata that makes the readability of this algorithm very straightforward. Moreover, this approach also facilitates considerably the analysis of the complexity of the algorithm, allowing us to devise easily a real time version of the KMP.

Sumário

Neste artigo nós apresentamos uma descrição do algoritmo KMP através de autômatos que torna a compreensão deste algoritmo bastante simples. Além disso, esta abordagem também facilita consideravelmente a análise de complexidade do algoritmo, a ponto de conseguirmos obter facilmente uma versão tempo real do mesmo, conforme apresentamos no artigo.

1 Introdução

O algoritmo de Knuth, Morris e Pratt [KMP77], freqüentemente denominado de algoritmo KMP, foi um dos primeiros algoritmos lineares

[*]O presente trabalho é parte da dissertação de Mestrado do primeiro autor, realizado sob a orientação do segundo autor. Suporte financeiro parcial do CNPq e da CAPES.

[†]O presente trabalho é parte da dissertação de Mestrado do primeiro autor, realizado sob a orientação do segundo autor. Suporte financeiro parcial do CNPq e da CAPES.

[‡]Aluno de Doutorado do DCC - IMECC - UNICAMP; e-mail : marcus@dcc.unicamp.br

[§]DCC - IMECC - UNICAMP; e-mail : lucchesi@dcc.unicamp.br

desenvolvidos para resolver o problema de *string matching*. Embora atualmente existam algoritmos mais eficientes na prática, a compreensão adequada do algoritmo KMP ainda é fundamental. Entretanto, embora esta importância teórica faça parte do senso comum, muitos programadores não o entendem com clareza (vide [Hor80], [Sed83], [HS91]).

O nosso objetivo neste artigo é apresentar uma descrição do algoritmo KMP que a nosso ver torna bastante simples a sua compreensão. Para realizar esta descrição, nós efetivamente lançamos mão de autômatos, de tal forma que o mecanismo de funcionamento do algoritmo se torna muito simples. Além disso, esta abordagem também facilita consideravelmente a análise de complexidade do algoritmo, a ponto de conseguirmos facilmente uma versão tempo real do mesmo.

A seguir, na seção 2 descreveremos o funcionamento do algoritmo supondo que o autômato já foi construído. Na seção 3 apresentaremos como construir o autômato. Na seção 4 apresentaremos algumas variações do algoritmo KMP cuja descrição também é simplificada pelo uso de autômatos. Finalmente, na seção 5 apresentaremos uma versão tempo real do algoritmo, que é extremamente simples, pelo menos no contexto de autômatos.

2 O Algoritmo KMP

Dada uma cadeia de caracteres $p = p_1 \ldots p_m$, suponha que queiramos obter *todas* as ocorrências de p num texto $t = t_1 \ldots t_n$. Inicialmente, é importante estabelecer que há uma *ocorrência* de p na posição k de t se e somente se $p_1 \ldots p_m = t_k \ldots t_{k+m-1}$.

Agora, dado o alfabeto Σ, seja $p = abcaabcaba$ e seja $\mathcal{A}$ o autômato da Figura 1. Conforme demonstramos em [And92], $\mathcal{A}$ é o autômato reduzido que reconhece $\Sigma^* p$.

No autômato $\mathcal{A}$ existem dois tipos de transições: as transições de *sucesso* (setas contínuas) e as transições de *falha* (setas pontilhadas). De posse deste autômato, o algoritmo KMP procede da seguinte forma: suponha que o autômato esteja no estado s, com $0 \leq s < m$ e seja t_k o caractere corrente no texto; se $t_k = p_{s+1}$ então realizamos a transição de

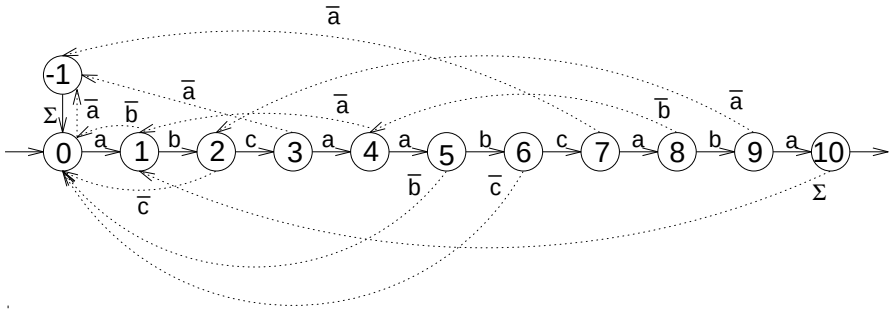

Figura 1: Autômato reduzido reconhecedor de *abcaabcaba*.

sucesso atingindo o estado $s+1$ e avançamos uma posição no texto, ou seja, o caractere corrente no texto passa a ser t_{k+1}; por outro lado, se $t_k \neq p_{s+1}$ então realizamos a transição de falha atingindo o estado que denotamos por $f(s)$ e, nesse caso, não avançamos o caractere corrente no texto. Vale ressaltar que os estados -1 e m são estados especiais, pois o estado -1 não possui transição de falha (ao atingir este estado, o caractere corrente do texto causa sempre uma transição de sucesso) e o estado m não possui transição de sucesso, pois este é o estado final (se este estado é alcançado então existe uma ocorrência de p na posição $k-m+1$ do texto e o processamento deve prosseguir a partir do estado $f(m)$). O processo descrito acima se inicia a partir do estado 0 com o caractere corrente no texto sendo t_1 e deve ser repetido até que o final do texto seja alcançado. Podemos então estabelecer uma definição precisa para a função de transição δ do autômato $\mathcal{A}$:

Definição 1 Dado $s : -1 \leq s \leq m$ e $a \in \Sigma$ então

$$\delta(s,a) = \begin{cases} s+1 & \text{se } (s=-1) \text{ ou } (0 \leq s < m \text{ e } a = p_{s+1}) \\ \delta(f(s),a) & \text{caso contrário.} \end{cases}$$

Na Figura 2 apresentamos o algoritmo *delta*, que implementa a função δ. Na Figura 3 apresentamos o algoritmo KMP, supondo dado o autômato $\mathcal{A}$.

Algoritmo delta (s, a);
Entrada : *O estado* s, $-1 \leq s \leq m$, *e o caractere* $a \in \Sigma$.
Saída : *O estado* $\delta(s, a)$.

```
begin
   if s = m then s := f(s);
   p_0 := a;                {sentinela}
   while a ≠ p_{s+1} do s := f(s);
   return s + 1;
end;
```

Figura 2: Uma implementação da função δ.

Algoritmo KMP;
Entrada : *O texto* t, *o padrão* $p \neq \epsilon$ *e o respectivo autômato* $\mathcal{A}$.
Saída : *Lista de posições em que* p *ocorre em* t.

```
begin
   s := 0;
   for k := 1 to n do
      begin
         s := delta (s, t_k);
         if s = m then write ('O padrão ocorre na posição ',
k − m + 1)
      end
end;
```

Figura 3: Uma implementação do algoritmo KMP.

Assim, podemos perceber que, uma vez construído o autômato $\mathcal{A}$, a compreensão e obtenção do algoritmo KMP se torna bastante simples. Agora, resta-nos descrever como obter o autômato $\mathcal{A}$.

3 A Construção do Autômato $\mathcal{A}$

Como a transição de sucesso de um estado $s : -1 \leq s < m$ sempre indica o estado $s + 1$ então a obtenção de $\mathcal{A}$ se resume a determinar o valor da transição de falha $f(s)$ dos estados $s : 0 \leq s \leq m$. Para determinar este valor, vamos analisar este tipo de transição.

Pela definição 1, temos que a transição de falha de um estado $s : 0 \leq s \leq m$ é efetuada em uma das duas situações: ou $s = m$ ou $s < m$ e o caractere corrente no texto (t_k) é diferente de p_{s+1}. Agora, para garantir o funcionamento correto do algoritmo, a transição de falha do estado s deve levar a um estado $i < s$ tal que a maior parte possível do prefixo de p já reconhecido até o estado s seja aproveitado na continuação do processamento.

Definição 2 Uma *borda* de p é um prefixo próprio de p que também é sufixo de p, isto é, q é uma borda de p se existem seqüências não vazias q' e q'' tais que $qq' = p = q''q$.

Portanto, a determinação da transição de falha do estado s a princípio corresponde a determinar o comprimento da maior borda de $p_1 \ldots p_s$, ou seja, o valor de $f(s)$ pode ser determinado calculando-se o maior i, com $0 \leq i \leq s - 1$, tal que $p_1 \ldots p_i$ seja um sufixo próprio de $p_1 \ldots p_s$.

Inicialmente, para $s : 0 \leq s \leq m$, vamos estabelecer que o valor de $f(s)$ deve ser tal que $p_1 \ldots p_{f(s)}$ seja a maior borda de $p_1 \ldots p_s$.

Pelas considerações acima, é fácil ver que $f(0) = -1$. Agora, para $s : 0 < s \leq m$, suponha que para todo $s' : 0 \leq s' < s$, $f(s')$ já foi calculada e suponha que queremos obter $f(s)$.

Observe que se $p_1 \ldots p_i$ é uma borda de $p_1 \ldots p_s$ então $p_1 \ldots p_{i-1}$ é uma borda de $p_1 \ldots p_{s-1}$. Assim, para obter a maior borda de $p_1 \ldots p_s$ basta verificar (em ordem decrescente de tamanho) qual é a primeira

dentre as bordas de $p_1 \ldots p_{s-1}$ que é seguida por um caractere igual a p_s. Visto que a lista das bordas de $p_1 \ldots p_{s-1}$ em ordem decrescente de tamanho é dada por $f(s-1)$, $f^2(s-1)$, $f^3(s-1)$, $\cdots$, então a maior borda de $p_1 \ldots p_s$ pode ser determinada comparando-se p_s com $p_{f(s-1)+1}$, $p_{f^2(s-1)+1}$, $\cdots$, nesta ordem, até que uma igualdade seja obtida. Agora, note que este processo é exatamente equivalente à realização de uma transição δ no autômato $\mathcal{A}$ a partir do estado $f(s-1)$ com o caractere p_s. Logo, o valor de $f(s)$ pode ser obtido fazendo-o igual a $\delta(f(s-1), p_s)$. Mais precisamente,

Definição 3 Para $0 \leq s \leq m$,

$$f(s) = \begin{cases} -1 & \text{se } s = 0 \\ \delta(f(s-1), p_s) & \text{caso contrário} \end{cases}$$

Conforme demonstramos em [And92], a definição acima produz um autômato que reconhece $\Sigma^* p$. Entretanto, este autômato é ligeiramente diferente do autômato $\mathcal{A}$ apresentado na Figura 1, sendo que esta diferença o torna menos eficiente do que o autômato $\mathcal{A}$. Para verificar este fato, sugerimos que o leitor utilize a definição acima com $p = abcaabcaba$ e compare o autômato obtido com aquele dado na Figura 1.

O motivo desta menor eficiência é porque existe uma informação importante que não foi considerada no cálculo de f. Esta informação advém da seguinte observação: para $0 < s < m$, se $p_{s+1} = p_{f(s)+1}$ então uma transição de falha no estado s implica necessariamente numa outra transição de falha no estado $f(s)$, ou seja, se $a \neq p_{s+1}$ então $\delta(s, a) = \delta(f(s), a) = \delta(f(f(s)), a)$. Portanto, quando p_{s+1} for igual a $p_{f(s)+1}$ então o valor de $f(s)$ pode ser atualizado para $f(f(s))$. Esta observação nos leva a estabelecer a seguinte definição:

Definição 4 Para $0 < s < m$, se $p_1 \ldots p_i$ é uma borda de $p_1 \ldots p_s$ e $p_{i+1} \neq p_{s+1}$ então dizemos que $p_1 \ldots p_i$ é uma *borda disjunta* de $p_1 \ldots p_s$.

Assim, para obter o valor de $f(s)$ devemos determinar a maior borda disjunta de $p_1 \ldots p_s$. Conforme descrevemos acima, a determinação da

maior borda disjunta é realizada determinando a maior borda e depois atualizando-a para garantir que esta borda é disjunta. Entretanto, é importante ressaltar que ao invés de calcular todos os valores de f e somente depois atualizá-los, por questões de eficiência, devemos realizar as atualizações em conjunto com o cálculo dos valores de f. Para isto é importante manter a seguinte ordem: para cada $s : 0 \leq s < m$, primeiro computamos o valor de $f(s+1)$ e depois atualizamos o valor de $f(s)$, pois para obter $f(s+1)$ necessitamos do valor original de $f(s)$. O autômato assim obtido, com a nova definição de f, é então dito *otimizado*. Na Figura 4, apresentamos uma implementação do algoritmo para o cálculo de $\mathcal{A}$.

Algoritmo Constroi_$\mathcal{A}$;
Entrada : *O padrão p.*
Saída : *Tabela com os valores de f.*

begin
  $f(0) := -1$;
  for $s := 0$ **to** $m - 1$ **do**
    begin
      $f(s+1) :=$ delta $(f(s), p_{s+1})$;
      if $f(s) \geq 0$ **and** $p_{f(s)+1} = p_{s+1}$ **then** $f(s) := f(f(s))$
    end
end;

Figura 4: Uma implementação do algoritmo para construir $\mathcal{A}$.

4 Variações do Algoritmo KMP

Dentre as diversas variações do algoritmo KMP, aquela onde a utilização de autômatos parece ser mais adequada é no algoritmo que permite a

busca incremental[1].

Esta variação do algoritmo KMP surge da observação de que, no início do processamento do texto, não precisamos ter o autômato $\mathcal{A}$ totalmente calculado, ou seja, não é necessário ter o valor da transição de falha de todos os estados, pois uma vez que a construção de $\mathcal{A}$ é efetuada de modo incremental então podemos proceder esta construção sob demanda, da seguinte forma: iniciamos o processamento do texto, comparando os caracteres do padrão com os caracteres do texto até obtermos a primeira diferença ou até atingirmos o final do padrão. Supondo que a etapa anterior se encerrou na posição s do padrão então, a partir de $f(0)$, obtemos $f(1), \cdots f(s)$ e este último valor informa em que estado devemos prosseguir com o processamento do texto. Se $s = m$ então $\mathcal{A}$ já foi totalmente determinado e portanto, o processamento prossegue normalmente. Porém, se $s < m$ e se posteriormente, durante o processamento do texto, atingirmos o estado s (maior estado para o qual f já foi calculada) então calculamos $f(s+1), \cdots, f(i)$, onde i é tal que $i = m$ ou $i < m$ e p_{i+1} é diferente do caractere corrente no texto. Em ambos os casos, o valor de $f(i)$ indica em que estado o processamento do texto deve prosseguir, sendo que se $i = m$ então uma ocorrência do padrão no texto foi obtida. Uma implementação destas idéias pode ser obtida em [Bar81] e em [Tak86].

5 Versão tempo real do algoritmo KMP

Uma das vantagens da modelagem do KMP com autômatos é a facilidade com que se obtém uma versão tempo real, mesmo quando o autômato não for otimizado.

A versão tempo real KMPR (Figura 5) utiliza um buffer de comprimento $r = \lfloor \frac{m+3}{2} \rfloor$, para implementar uma *fila* de caracteres. Para facilitar a implementação desta fila adotamos um buffer circular alocado em um vetor cujos índices variam entre 0 e $r-1$. Esta fila inicialmente

[1] Um exemplo típico de busca incremental ocorre nas funções de busca de palavras de alguns editores de texto que realizam a busca à medida que o usuário fornece a palavra.

é vazia e se destina a acolher caracteres do texto a ser processado. Ao invés de usar a função *delta*, a versão tempo real utiliza a função *transita*, que efetua apenas uma transição por chamada. O algoritmo KMPR executa então no máximo 2 transições por caractere da entrada, deixando para depois outras transições necessárias. Este adiamento é a razão da existência do buffer.

Algoritmo KMPR;
Entrada : *Texto t, padrão* $p \neq \epsilon$ *e o autômato* $\mathcal{A}$*, otimizado ou não.*
Saída : *Lista de posições em que p ocorre em t.*

```
    procedure transita;
        begin
            if s = m or (s > −1 and b_head ≠ p_{s+1}) then s := f(s)
            else
                begin
                    s := s + 1;
                    head := (head + 1) mod r;
                end
        end;
begin
   s := 0; r := (m + 3) div 2; head := 0; tail := r − 1;
   for k := 1 to n do
       begin
           tail := (tail + 1) mod r; b_tail := t_k; transita;
           if head ≠ (tail + 1) mod r then transita;
           if s = m then write ('p ocorre na posição ', k − m + 1)
       end
end;
```

Figura 5: Uma implementação tempo real do algoritmo KMP.

Por exemplo, se considerarmos o texto $t = aaabaaa$ e o padrão $p = aaa$ cujo autômato não otimizado é dado na Figura 6, a tabela na Figura 7 mostra o estado em que se encontra o autômato e o conteúdo do buffer após o tratamento de cada posição k do texto ($k = 0$ indica a situação imediatamente após a inicialização).

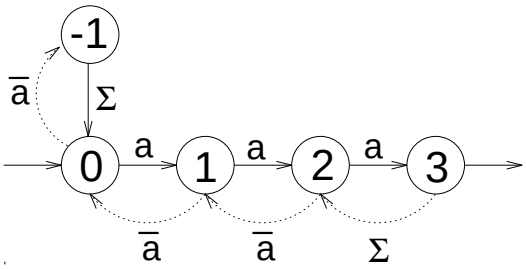

Figura 6: Autômato não otimizado para o padrão aaa.

k	0	1	2	3	4	5	6	7
t_k		a	a	a	b	a	a	a
s	0	1	2	3	1	-1	1	3
buffer	ϵ	ϵ	ϵ	ϵ	b	ba	a	ϵ

Figura 7: Rastreio do algoritmo KMPR, $p = aaa$ e $t = aaabaaa$.

Para verificar a correção do algoritmo KMPR, observe que se denotarmos por l o comprimento do buffer no início de cada iteração, então naquele ponto temos a seguinte invariante:

$$2l \leq m - s \tag{1}$$

De fato, é fácil ver que a equação (1) é válida no início da computação. Além disso, se numa dada iteração são executadas 2 transições, dentre as quais ϕ são de falha, então a tabela da Figura 8 mostra quais as variações sofridas por l e s, respectivamente denotadas Δl e Δs, em todos os casos

possíveis. Observa-se então que a validade da equação (1) é preservada nesses casos.

ϕ	Δl	Δs
0	-1	2
1	0	≤ 0
2	1	$\leq$ -2

Figura 8: Variação Δl do comprimento do buffer e variação Δs do estado do autômato não otimizado, quando são executadas duas transições numa iteração do algoritmo KMPR, das quais ϕ são de falha.

Finalmente, numa iteração em que é executada apenas uma transição, então necessariamente essa transição é de sucesso e o buffer fica vazio: nesse caso, a desigualdade $s \leq m$ implica trivialmente a validade de (1). De fato, (1) é uma invariante do algoritmo KMPR.

O tamanho $\lfloor \frac{m+3}{2} \rfloor$ do buffer é justificado a partir de (1), dado que $s \geq -1$ e portanto $2l \leq m+1$. Assim, é possível que $\lfloor \frac{m+1}{2} \rfloor$ posições do buffer estejam ocupadas ao início de uma iteração, o que exige tamanho $\lfloor \frac{m+3}{2} \rfloor$ para acomodar o novo caractere t_k proveniente do texto.

Vamos agora fazer alguns comentários sobre a correção do algoritmo KMPR. Em primeiro lugar, a seqüência de estados atingidos pelo algoritmo KMPR é igual à seqüência correspondente do algoritmo KMP, porém com possível atraso. Em segundo lugar, consideremos uma iteração genérica, seja s_0 o estado em que se encontra o autômato ao início da iteração. Se o estado final m é atingido na primeira chamada de *transita* da iteração então necessariamente $s_0 = m-1$, o que implica que ao início da iteração o buffer estava vazio, por (1); nesse caso, o buffer é novamente esvaziado quando da entrada no estado m e a segunda chamada de *transita* não é executada. Por outro lado, se o estado final m é atingido na segunda chamada de *transita*, a desigualdade (1) implica que o buffer está vazio ao final da iteração. Portanto, toda vez que o estado final m é atingido numa iteração o buffer fica vazio ao seu final

e a indicação da ocorrência do padrão justamente encontrada é feita "a tempo".

O algoritmo KMPR está portanto correto e é, obviamente, uma versão tempo real do KMP. Além disso, podemos concluir que mesmo a versão não otimizada do algoritmo KMP executa em média não mais do que 2 transições por caractere do texto e é portanto um algoritmo linear.

Finalmente, convém recordar que a versão otimizada do algoritmo KMP executa não mais do que $O(\log(m))$ transições, no pior caso, para cada caractere da entrada [KMP77]; a versão otimizada, da mesma forma que a não otimizada, admite também uma versão tempo real com no máximo 2 transições por caractere de entrada, desde que se utilize um buffer para amortizar o número maior de transições que ocorre em alguns casos do cálculo da função δ.

6 Conclusões

Conforme vimos acima, a utilização efetiva de autômatos nos permite obter uma descrição bastante clara do algoritmo KMP e além disso, conforme apresentado em [And92], esta abordagem também possibilita que a prova de corretude e demonstração de algumas propriedades importantes do algoritmo KMP sejam realizadas de forma bem mais simples e clara do que as originalmente propostas. Finalmente, vimos que uma versão em tempo real do algoritmo KMP pode ser obtida facilmente graças a esta abordagem.

7 Bibliografia

[And92] M. V. A. Andrade. Métodos eficientes para reconhecimento de padrões em textos. Master's thesis, DCC, Unicamp, setembro de 1992.

[Bar81] G. Barth. An alternative for the implementation of the Knuth-Morris-Pratt algorithm. *Information Processing Letters*, 13(5):134–137, 1981.

[Hor80] R. N. Horspool. Practical fast searching in strings. *Software-Practice and Experience*, 10:501–506, 1980.

[HS91] A. Hume e D. Sunday. Fast string searching. *Software-Practice and Experience*, 21(11):1221–1248, 1991.

[KMP77] D. E. Knuth, J. H. Morris, e V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[Sed83] R. Sedgewick. *Algorithms*. Addison-Wesley, Reading, Mass., 1983.

[Tak86] T. Takaoka. An on-line pattern matching algorithm. *Information Processing Letters*, 22(6):329–330, 1986.

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*

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

reltec@dcc.unicamp.br