Um estudo sobre a Ação Elementar de Consulta no Formalismo Adaptativo

I. Chaer, R. L. A. Rocha

***Resumo* — Este trabalho discute o mecanismo de consulta especificado no Formalismo Adaptativo, levantando algumas questões que ainda não foram bem estudadas nas aplicações existentes. É proposta uma nova definição do seu comportamento, abrangente o suficiente para servir a qualquer aplicação do Formalismo sem restringir a sua capacidade e, acredita-se, sem impor novas dificuldades ao desenvolvimento. Também é delineado um método, restrito a situações nas quais o dispositivo subjacente à camada adaptativa tem poder equivalente às gramáticas regulares, capaz de satisfazer os requisitos dessa definição.**

***Palavras-chave* — dispositivo adaptativo (*adaptive device*), busca (*search*), consulta (*query*), autômato adaptativo (*adaptive automaton*).**

I. Introdução

Desde a sua definição [8], o Formalismo Adaptativo foi muito usado em tarefas de análise de linguagens como uma extensão natural dos modelos de Autômatos de Estados Finitos [2][8][9] e, mais tarde, como uma camada independente do mecanismo de processamento subjacente [10][11], consistindo a partir de então em um método de extensão do poder computacional de qualquer Dispositivo Guiado por Regras. No entanto, devido à falta de métodos de construção de modelos, as aplicações existentes foram desenvolvidas de maneira *ad-hoc*, ou seja, específica para atender os problemas individuais de cada implementação. A definição original do Formalismo tem potencial para utilização em um universo de problemas muito mais amplo do que aquele que foi explorado, e alguns pontos ainda não foram suficientemente estudados. Um dos pontos que apresentam mais evidência é o da operação de consulta – das três ações adaptativas elementares, é a que se aplica no maior número de situações, e, ao mesmo tempo, aquela com maior potencial para aumentar a complexidade computacional de uma aplicação.

Para tratar a questão adequadamente é necessário, inicialmente, expor a definição do Formalismo Adaptativo, o que é feito na seção II. A seção III traz uma exposição sobre linguagens formais, discutindo pontos relevantes para a questão da ação elementar de consulta no Formalismo Adaptativo. Na seção IV fala-se da operação de consulta no Formalismo, analisando como o problema tem sido tratado, o que foi deixado de lado – por não se ter feito necessário até então – e é proposta uma definição mais detalhada. Na Seção V é delineado um método para realizar a consulta de acordo com a definição proposta. Finalmente, a seção VI traz as considerações finais do trabalho.

Este trabalho recebeu apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) através do programa de Bolsas de Mestrado.

I. Chaer é estudante de Mestrado junto ao Laboratório de Linguagens e Técnicas Adaptativas, Escola Politécnica da USP, São Paulo/SP, Brasil; e-mail: iuri.chaer@poli.usp.br.

R. L. A. Rocha é pesquisador do Laboratório de Linguagens e Técnicas Adaptativas, Escola Politécnica da USP, São Paulo/SP, Brasil; fone: 55 11-3091-5583; e-mail: luis.rocha@poli.usp.br.

II. O Formalismo Adaptativo

O Formalismo Adaptativo consiste fundamentalmente em um mecanismo de auto-modificação que pode ser aplicado a qualquer dispositivo guiado por regras [5][10]. Esse mecanismo, definido ele mesmo como um dispositivo guiado por regras, age como uma camada superior que dá, mesmo aplicado sobre um dispositivo com capacidade equivalente a uma gramática regular, poder computacional para simular a Máquina de Turing [13].

Nas subseções seguintes será formalizado o conceito de dispositivo guiado por regra e de dispositivos adaptativos, foco de estudo deste trabalho. Elas são uma simplificação do exposto em [5] e [10] – recomenda-se aos interessados em uma exposição mais detalhada que leiam esses trabalhos.

A. Dispositivos guiados por regras

A definição de *dispositivo guiado por regras* engloba alguns dos principais formalismos usados como referência no estudo da computação, tais como a Máquina de Turing, os Autômatos de Estados Finitos e as Árvores de Decisão. São dispositivos determinados por conjuntos finitos de configurações e de regras, capazes de, quando alimentados com uma cadeia de símbolos, emitir (ou não) uma cadeia de saída e um julgamento de aceitação ou rejeição.

Formalmente, o dispositivo guiado por regras D pode ser definido como uma ênupla $D = (C, \Sigma, \Phi, c_0, C_a, R)$, onde:

- C é o conjunto de todas as configurações que D pode assumir,
- Σ é o alfabeto de entrada – o conjunto de todos os símbolos que podem ser consumidos por D (incluindo ε, o símbolo nulo),
- Φ é o alfabeto de saída de D (também incluindo ε),
- c_0 é a configuração inicial do dispositivo ($c_0 \in C$),
- C_a é o conjunto das configurações de aceitação de D ($C_a \subseteq C$)
- e R é o conjunto de regras que define as operações

em D ($R \subseteq C \times \Sigma \times C \times \Phi$).

Cada regra $r_i \in R$ é, assim, dada por uma ênupla $r_i = (c_i, \sigma, c_j, \phi)$, de maneira que a cada passo o dispositivo, estando em uma configuração c_i, consome um símbolo $\sigma \in \Sigma$, passa a um estado c_j e emite um símbolo $\phi \in \Phi$. Partindo da configuração c_0 e sendo alimentado com uma cadeia de símbolos s, o dispositivo D vai consumindo os símbolos de s e mudando de configuração até chegar a um ponto onde não haja nenhuma regra de R que possa ser aplicada ou que s seja esgotada. Se nesse momento o dispositivo estiver em uma configuração $c_i \in C_a$ e todos os símbolos de s houverem sido consumidos, diz-se que s foi *aceita* por D. Se o dispositivo houver terminado o processamento sem que as condições de aceitação tenham sido satisfeitas, diz-se que s foi *rejeitada*.

Note que a definição não limita, em nenhum momento, a aplicação de regras dependendo do símbolo nulo ε nem a existência de múltiplas regras para uma mesma configuração e símbolo consumido. Essas situações são chamadas *não-determinismos* porque, ao chegar a um ponto onde ocorra algum desses eventos, é possível prosseguir por mais de um caminho (ou seja, o progresso não está univocamente determinado). Nessas situações todas as alternativas serão processadas. Se alguma delas terminar em alguma situação de aceitação, conforme descrito anteriormente, a cadeia é considerada *aceita*.

B. Dispositivos Adaptativos

Conforme exposto no início desta seção, um dispositivo adaptativo consiste em um dispositivo guiado por regras D encapsulado por um mecanismo adaptativo A. O mecanismo A associa a cada regra de transição do dispositivo subjacente D duas operações de alteração estrutural sobre D, uma anterior e outra posterior à aplicação da regra, cada uma delas (ou ambas) podendo ser nula. Assim, o dispositivo adaptativo D_a pode ser definido formalmente como a dupla $D_a = (D, A)$, com D conforme a descrição na subseção anterior e $A \subseteq O_a \times R \times O_a$. As regras adaptativas em A podem agir sobre o dispositivo D alterando: o seu conjunto de possíveis configurações C, de configurações de aceitação C_a, o conjunto de regras R.

O conjunto de operações adaptativas O_a contém, além da operação nula, combinações das três ações adaptativas elementares:

- consulta, designada pelo símbolo ?;
- remoção, representada como -;
- e inserção, com o símbolo +.

Essas operações têm de ser aplicadas na ordem especificada acima: primeiro consultas, depois remoções e, ao final, inserções. Em [5] é proposto que a toda operação de remoção e inserção esteja associada uma consulta – o atual trabalho adota esse procedimento, centralizando nessa ação elementar os percursos pelo caminho do dispositivo. As operações de inserção e remoção têm como objeto os conjuntos C, R e A. Com isso, o que se tem ao exercitar um dispositivo adaptativo são, além das transições de configuração características dos dispositivos guiados por regras, transições entre diferentes dispositivos guiados por regras. A definição exata da operação de consulta está sendo propositalmente adiada – a sua caracterização em trabalhos anteriores não é suficientemente precisa para as necessidades desta pesquisa, e a discussão sobre o assunto requer uma seção exclusiva.

III. Tratamento de Linguagens Formais

Em [12] Chomsky estabeleceu as bases de uma hierarquia de linguagens definida a partir do tipo de restrições impostas pelo conjunto de regras necessárias para gerá-las. Essa hierarquia acaba por definir, como corolário, uma ordem entre os dispositivos computacionais, já que a capacidade reconhecimento de cada nível da hierarquia implica na capacidade de reconhecimento de todas aquelas que estão abaixo. Um dispositivo capaz de reconhecer linguagens do tipo 3 só é capaz de reconhecer esse tipo de linguagem. Um que reconheça linguagens do tipo 2 é também capaz de reconhecer todas as do tipo 3; e assim sucessivamente até o tipo 0, cujos aceitadores são capazes de reconhecer todos os demais tipos de linguagem.

Sendo α e β símbolos do alfabeto de uma linguagem, e **A**, **B** e **C** símbolos não-terminais usados na gramática que gera essa linguagem, pode-se representar as alternativas de regras de produção de cada gramática da seguinte maneira [1]:

- Linguagens do tipo 0 (recursivamente enumeráveis): $\alpha \rightarrow \beta$
- Linguagens do tipo 1 (sensíveis a contexto): $\alpha\mathbf{A}\beta \rightarrow \alpha\mathbf{B}\beta$
- Linguagens do tipo 2 (livres de contexto): $\mathbf{A} \rightarrow \mathbf{BC}$
- Linguagens do tipo 3 (regulares): $\mathbf{A} \rightarrow \alpha\mathbf{B}$

De maneira que linguagens recursivamente enumeráveis são geradas por gramáticas que permitem qualquer regra de produção, inclusive a troca de símbolos terminais. Linguagens sensíveis a contexto são mais restritas, permitindo, no máximo, a troca de um símbolo não-terminal dado um contexto local de símbolos terminais. Linguagens livres de contexto não permitem nenhuma produção onde seja analisado um símbolo terminal, mas permitem produções gerando mais de um símbolo não-terminal, e linguagens regulares permitem somente derivações de não-terminais a terminais e, no máximo, um não-terminal, sendo que todas as derivações da gramática resultando em algum não-terminal tem de ter os não-terminais na mesma posição em relação aos terminais gerados (não é aceitável que aconteça $\mathbf{A} \rightarrow \alpha\mathbf{B}$ e $\mathbf{B} \rightarrow \mathbf{A}\alpha$ na mesma gramática).

Dispositivos adaptativos, conforme descrito em seções anteriores, têm poder computacional equivalente à Máquina de Turing (i.e. são capazes de aceitar uma linguagem do tipo 0) mesmo quando o mecanismo adaptativo é sobreposto a um formalismo capaz apenas de lidar com linguagens do tipo 3 [13].

Tomando como referência um autômato de estados finitos como dispositivo subjacente, é fácil ver que as operações adaptativas necessárias para reconhecer uma linguagem do tipo 2 ficam restritas às transições e estados vizinhos à transição sendo ativada.

Para reconhecer a maioria das linguagens de programação

– um subconjunto particularmente restrito das linguagens do tipo 1 – lida-se com situações onde é possível estabelecer um número pequeno de estruturas onde há de fato dependência de contexto (e.g. declaração de parâmetros, escopo de estruturas condicionais), e as alterações no dispositivo subjacente ficam restritas aos módulos responsáveis por elas. No entanto, para linguagens com dependências de contexto seguindo em fluxos distintos (como, por exemplo, as anáforas nas linguagens naturais, que podem ser relativas a especificadores que as precedem ou as sucedem), ou onde a própria estrutura de dependências de contexto varia. Nessas condições, é provável que a estrutura do dispositivo subjacente seja alterada radicalmente e de maneiras imprevisíveis *a priori*, inclusive de maneira que fique impossível reconhecer (ou recompor) a estrutura original. Esse tipo de situação não foi estudada em nenhum trabalho anterior com o Formalismo Adaptativo, mas somente perseguindo os casos mais extremos é possível estabelecer soluções que valham em qualquer situação, e é certo que, dentro do conjunto infinito de linguagens dos tipos 0 e 1, o subconjunto que gera esse tipo de comportamento em dispositivos adaptativos não é nada desprezível.

As implicações do tipo da linguagem sendo processada sobre os passos computacionais de um dispositivo adaptativo têm impacto especialmente no controle do dispositivo subjacente. No modelo seguido neste trabalho, onde as operações de caminhamento no dispositivo ficam concentradas na ação adaptativa elementar de consulta, é para ela que será transferida a complexidade adicional.

IV. A Operação de Consulta

Na exposição da seção anterior, observa-se que os dispositivos adaptativos são capazes de reconhecer linguagens dos tipos 1 e 0, exceto em casos particulares, podem sofrer alterações extensas nos seus dispositivos subjacentes. As abordagens ao problema da consulta que podem ser encontradas nas aplicações existentes do formalismo adaptativo [2][5][9] seguem a descrição original em [8], onde a operação de busca recebe uma ênupla que representa a relação que define uma regra de transição adaptativa:

$$(o_{pré}, c_0, \sigma, c_1, \phi, o_{pós})$$

Sendo:

- $o_{pré} \in O_a$ a operação adaptativa anterior à transição;
- $c_0 \in C$ o estado de origem da transição;
- $\sigma \in \Sigma$ o símbolo consumido na transição;
- $c_1 \in C$ o estado de destino da transição;
- $\phi \in \Phi$ o símbolo de saída na transição e
- $o_{pós} \in O_a$ a operação adaptativa posterior à transição.

Esse modelo de consulta é bastante restrito por ser baseado nos rótulos dos estados e em relações envolvendo somente um símbolo de entrada ou saída. Com isso, ou se define rótulos com um valor semântico *a priori* que não vai poder ser alterado ao longo da execução, ou restringe-se o dispositivo suficientemente para que buscas baseadas em somente um símbolo não resultem em ambigüidades. Claramente nenhuma dessas restrições pode ser assumida em um mecanismo genérico para a implementação da ação adaptativa elementar de consulta, exceto em soluções pontuais, *ad-hoc*. Pistori toca esse problema em [5], onde propõe que a operação de consulta seja abordada como um problema de satisfação de relações entre as variáveis, e levanta a questão de refinamento dos resultados de uma busca.

A questão central para conceber um mecanismo genérico de consulta para um dispositivo guiado por regras é decidir quais parâmetros são suficientes para gerar resultados precisos, e a inclusão da camada adaptativa adiciona o requisito de que não se recorra a informações relativas à forma original do dispositivo. A resposta natural a essa questão é adicionar aos parâmetros a própria funcionalidade do setor do dispositivo que se deseja encontrar, e fazer com que as consultas funcionem recursivamente, para que os seus resultados possam ser refinados buscando sub-módulos específicos de cada módulo funcional. Por causa desse sistema de refinamento da busca, será abandonada, nessa proposta, a idéia defendida em [4][5][14] de, havendo co-referência entre as variáveis passadas nas operações de consulta, localizar somente os resultados que satisfaçam a essas relações: esse procedimento adiciona muita complexidade ao sistema e só permite a localização através de detalhes estruturais do dispositivo subjacente. A proposta é, assim, que uma chamada à operação de consulta consistiria em uma seqüência contendo no mínimo uma instância da seguinte ênupla:

$$(o_{pré}, c_0, \sigma, c_1, \phi, o_{pós}, f)$$

Sendo o último elemento, f, a função semântica que caracteriza o sub-módulo funcional a ser localizado, e os elementos anteriores e posteriores à transição passam a referenciar, então, às regiões anteriores e posteriores ao sub-módulo f. A passagem de informações na operação de consulta é por valor, e não referência, de maneira que as incógnitas devem ser caracterizadas pela passagem de um valor inválido, equivalente à constante indefinida encontrada na maioria das linguagens de programação (mas não igual ao símbolo vazio ε). Se f ficar indefinido em alguma ênupla da consulta, assume-se, naturalmente, que a intenção é localizar uma única transição, e a operação na recursão originada por essa ênupla reverte ao modo da definição original. Quando f estiver definido, o símbolo consumido σ também assume uma interpretação mais geral – sendo o interesse na consulta localizar o sub-módulo dentro do dispositivo, são retornados os símbolos consumidos em todas as transições externas ao sub-módulo que levam a ele; no caso de não haver nenhuma transição desse tipo (e.g. sub-módulos que compartilham o estado inicial do dispositivo), o símbolo assume um valor inválido. Cada ênupla da seqüência passada na invocação da operação define uma recursão, onde será feita uma nova busca restrita às sub-máquinas localizadas na recursão anterior. O algoritmo de consulta mais simples poderia ser implementado, em pseudocódigo, da seguinte maneira:

```
consulta(padrões, dispositivoSubjacente, índice)
  dispositivosLocais = [];
```

```
  foreach sub-módulo in dispositivoSubjacente
    if satisfaz(sub-módulo, padrões[índice])
      push(dispositivosLocais, sub-módulo)
  índice = índice + 1
  if índice >= length(padrões)
    return dispositivosLocais
  else
    dispositivos = []
    foreach sub-módulo in dispositivosLocais
      push(dispositivos,
           consulta(padrões, sub-módulo, índice))
  return dispositivos


satisfaz(padrão, disp)
  resultados = []
  if  ( (not defined padrão.f and
         not defined disp.f) or
         disp.f == padrão.f) and
      indefOuIgual(padrão.o_pré, disp.o_pré) and
      indefOuIgual(padrão.c_0, disp.c_0) and
      indefOuIgual(padrão.σ, disp.σ) and
      indefOuIgual(padrão.c_1, disp.c_1) and
      indefOuIgual(padrão.ϕ, disp.ϕ) and
      indefOuIgual(padrão.o_pós, disp.o_pós)
    return true
  else
    return false


indefOuIgual(a, b)
  if not defined(a) or a==b
    return true
  else
    return false
```

Dessa maneira, recorrendo a vetores para tratar as multiplicidades de resultados e parâmetros na recursão, o que se faz é verificar cada sub-módulo do dispositivo em busca daqueles capazes de satisfazer à seqüência de padrões definida. O algoritmo dado testa exaustivamente todas as possibilidades, e sem dúvida pode ser melhorado. A obtenção dos sub-módulos que compõem um dispositivo e a determinação da função semântica deles dependem do formalismo subjacente sendo usado no dispositivo adaptativo. Note que as próprias regras do dispositivo são tratadas como sub-máquinas atômicas sem função semântica marcada – elas são definidas univocamente pelos seus outros parâmetros –, e que a reversão para a definição original ocorre naturalmente no algoritmo.

É importante, entretanto, manter em mente que essa proposta, apesar de ser capaz de aumentar muito o poder da ação de consulta, implica em um grande aumento na complexidade computacional da sua execução. Isso porque, tomando como referência de dispositivo subjacente o autômato de estados finitos, mesmo sendo definida uma forma normal biunívoca para definir a relação entre autômatos e funções sintáticas (i.e., somente um autômato implementando cada função e somente uma função sendo implementada por cada autômato), no pior caso haverá recursões equivalentes ao problema de isomorfismo de sub-grafos: um problema conhecidamente NP Completo.

Uma vez proposta a interface de um mecanismo de consulta que pode ser aplicado a um método genérico de construção de dispositivos adaptativos, dada a observação já feita quanto à complexidade da sua computação, é importante discutir o seu tratamento. É condição *sine qua non*, uma vez que respostas em tempo infinito não são suficientes para o modelo adaptativo, que seja estabelecida uma forma normalizada para a descrição da semântica subjacente às possíveis sub-módulos do dispositivo. Tendo essa forma normalizada, é possível em um número de passos $O(2^n)$ realizar qualquer consulta por comparação exaustiva; $O(2^{n \cdot n})$ considerando as recursões do procedimento proposto. No entanto, restringindo o dispositivo guiado por regras subjacente do dispositivo adaptativo, há métodos de busca com custo em tempo muito melhor na maioria dos casos. Na seção seguinte será proposto um método para executar consultas em autômatos de estados finitos adaptativos, método esse que certamente pode ser estendido a qualquer dispositivo adaptativo cujo formalismo subjacente tenha poder de computação equivalente ao das gramáticas regulares.

V. Um Método de Consulta para Autômatos de Estados Finitos Adaptativos

O autômato de estados finitos (AF) é um formalismo extremamente atraente como camada subjacente de um dispositivo adaptativo devido à sua simplicidade: sendo capaz de reconhecer somente linguagens regulares, é normalmente bastante simples projetar e compreender o funcionamento de um autômato desse tipo. O motivo pelo qual ele foi escolhido para este trabalho, no entanto, é o ferramental disponível para o seu tratamento. Em [5], Hopcroft apresenta um algoritmo com complexidade assintótica $O(n \cdot log(n))$ para minimizar um autômato de estados finitos determinístico (AFD). O AFD mínimo é uma descrição única para uma linguagem regular [1][6], e, por isso, um ótimo candidato para o elemento ***f*** requerido para os parâmetros de consulta definidos na seção anterior.

Árvores e AFDs são muito semelhantes estruturalmente. Estabelecendo uma relação de ordem para o alfabeto Σ reconhecido por um AFD é possível usar os algoritmos de caminhamento pré-ordem e pós-ordem exigindo somente uma pequena modificação para tratamento de ciclos (o caminhamento precisa ser interrompido sempre que um for encontrado). Isso é um fato particularmente interessante porque, uma vez estabelecida a relação de ordem e a forma de caminhamento, pode-se então caracterizar o autômato através de uma cadeia de comprimento $O(n+m)$ símbolos, sendo n o número de estados e m o número de regras de transição do AFD.

Sendo o AFD mínimo uma descrição única para um AF e havendo a possibilidade de descrevê-lo com uma simples cadeia de caracteres, existe a possibilidade de indexar a semântica inerente a cada sub-módulo de um AF em estruturas onde a busca pode ser executada com custo extremamente baixo, tais como *hashes* e árvores binárias, reduzindo o custo de cada recursão em uma consulta do pior caso mencionado

anteriormente, $O(2^n)$, a $O(1)$ ou $O(log(n))$. No entanto, há um custo colateral: é necessário gerar, para cada sub-módulo possível do dispositivo subjacente, as classes de equivalência da linguagem que ele reconhece, a um custo $O(2^n)\cdot(O(n\cdot log(n))+O(n))$ – ou seja, $O(2^n)$. Incorre-se nesse custo no momento da indexação, e, a cada atualização do índice, em um custo de $\Omega(n\cdot log(n))$ e $O(2^n)$, dependendo da influência da alteração sobre o dispositivo como um todo. Para suprir a questão da recursão é necessário que o índice construído relacione, para cada classe de equivalência, todos os sub-módulos funcionais contidos no módulo responsável pela classe em questão. Isso não é particularmente custoso ou complicado uma vez que todos esses módulos têm mesmo que ser gerados *bottom-up*, mas essa indexação resulta também em um custo em espaço polinomial em relação ao número de regras no dispositivo subjacente. Caso não seja possível garantir que o AF subjacente é determinístico, haverá necessidade de converter cada sub-módulo no seu análogo determinístico mínimo, uma tarefa executada em $O(2^n)$ passos com o algoritmo apresentado em [3] e [7] que terá de ser repetida para cada sub-módulo indexado (inclusive podendo aumentar o número e de estados de cada sub-módulo a 2^e), motivo pelo qual é extremamente recomendável usar somente AFDs.

A complexidade apresentada pelo método proposto decai a um custo inferior ao do pior caso apresentado na seção IV. No entanto, isso não invalida o método como uma maneira de reduzir o custo real do procedimento. Note que a geração de um novo índice de sub-módulos caracterizados pelos seus AFDs mínimos só ocorre quando há mudanças no dispositivo subjacente, e, mais que isso, as atualizações na maioria das situações possíveis não chegará ao pior caso. O custo, em uma aplicação que execute mais consultas do que inserções e remoções, será certamente menor do que se fosse usado o algoritmo exaustivo, já que o custo de cada re-indexação será amortizado entre todas as consultas. Considerando que, no modelo assumido neste trabalho, é necessária uma consulta para a execução de cada inserção ou remoção, dada uma aplicação genérica do formalismo é extremamente provável que o procedimento descrito resulte em ganhos no tempo de processamento. Além disso, se o conceito de consultas pela semântica for adotado completamente (i.e., os rótulos dos estados forem ignorados em todas as operações de consulta), é possível abstrair a estrutura real do dispositivo subjacente, criando a possibilidade de realizar nele operações de otimização (e.g. remoção de redundâncias) sem prejuízo; potencialmente melhorando o desempenho tanto do sistema de indexação para consulta quanto do uso do autômato para reconhecer cadeias.

Para exemplificar o funcionamento do método proposto, será usado o AF não-determinístico da figura 1:

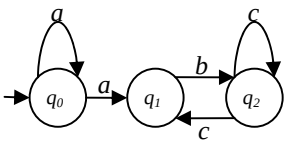

Fig. 1. Exemplo de autômato de estados finitos não-determinístico

Usando o método proposto nesta seção, é necessário, em primeiro lugar, determinar o AFD mínimo de cada sub-módulo do AF apresentado. Na figura 2 são mostradas as versões determinísticas mínimas dos sub-módulos contendo estados ligados por transições iniciados em estados com transições de saída:

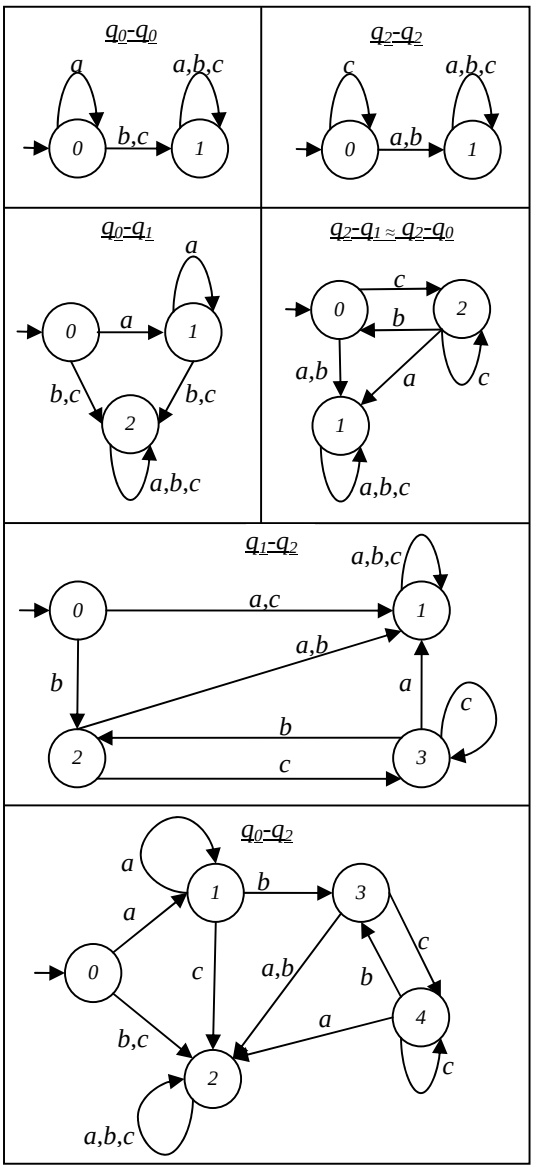

Fig. 2. Sub-módulos minimizados do AF não-determinístico da figura 1

Nesse exemplo, os estados já foram rotulados de acordo com a ordem de visitação seguindo o algoritmo pré-ordem assumindo que $a<b<c$. Dessa maneira, o índice para consultas teria a seguinte aparência (omitindo as operações adaptativas e

o símbolo de saída, dado que não foram definidos estados de aceitação ou mecanismo adaptativo):

TABELA I
ÍNDICE PARA BUSCAS NO AF DA FIGURA 1

c_0	σ	c_1	f
q_0	*a*	q_0	*indef*
q_0	*a*	q_1	*indef*
q_1	*b*	q_2	*indef*
q_2	*c*	q_1	*indef*
q_2	*c*	q_2	*indef*
q_0	-	q_0	*011111*
q_2	*b*	q_2	*110111*
q_0	-	q_1	*122122222*
q_2	-	q_1	*112102222*
q_2	-	q_0	*112102222*
q_1	*a*	q_2	*121111113123*
q_0	-	q_2	*122132222114234*

Cada número mostrado na coluna f representa uma transição consumindo o símbolo que viria naquela ordem – por exemplo, na sexta linha, *011111* significa que, do estado *0*, alcança-se o estado *0* consumindo *a*, *1* consumindo *b* e *1* consumindo *c*; do estado *1*, atinge-se *1* consumindo *a*, *b* ou *c*.

VI. CONCLUSÃO

Neste trabalho foi discutida a ação adaptativa elementar de consulta. Inicialmente foi exposto o seu papel dentro do Formalismo Adaptativo, ressaltando sua importância para o funcionamento de um dispositivo adaptativo e o universo restrito de aplicações que lhe foram destinadas até o momento.

oi mostrado que existem situações para as quais a definição da ação adaptativa elementar de consulta que foi usada nos trabalhos existentes é insuficiente, e propostos novos parâmetros para que essa ação possa ser executada em aplicações genéricas do Formalismo. Foram discutidas algumas questões sobre as dificuldades de implementar a ação da consulta satisfazendo os requisitos da nova proposta, em especial a complexidade da sua computação, exponencial no tempo em relação ao número de configurações do dispositivo subjacente.

Finalmente, foi delineado um algoritmo para mitigar o custo de uma operação genérica como a proposta, e discutido, *grosso modo*, o seu custo computacional. Acredita-se que a proposta feita tem uma relação custo-benefício favorável, mas não foram desenvolvidas aplicações reais para demonstrar a validade dessa conjectura. Isso é deixado como tarefa para trabalhos futuros – tanto em experimentos baseados na reprodução de aplicações já existentes quanto verificando o comportamento da proposta em linguagens cujas gramáticas exijam alterações extensas no dispositivo adaptativo que as reconhece.

REFERÊNCIAS

[1] A. Salomaa. *Formal Languages.* Academic Press, 1973.

[2] D. P. Shibata. "Tradução Grafema-Fonema para a Língua Portuguesa Baseada em Autômatos Adaptativos", Dissertação de Mestrado, USP, São Paulo, 2008.

[3] H. Lewis e C. Papadimitriu, *Elements of the Theory of Computation.* Prentice-Hall, 1998.

[4] H. Pistori e J. J. Neto, "AdapTools: Aspectos de Implementação e Utilização," *Boletim Técnico PCS*, Escola Politécnica, São Paulo, 2003.

[5] H. Pistori, "Tecnologia Adaptativa em Engenharia de Computacão: Estado da Arte e Aplicações," Tese de Doutorado, USP, São Paulo, 2003.

[6] J. E. Hopcroft. "An nlogn algorithm for minimizing the states in a finite automaton." in The Theory of Machines and Computations, ed. Z Kohave. New York, Academic Press, 1971.

[7] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to automata theory, languages, and computation*, 2nd. ed. Ed. Addison-Wesley, 2001.

[8] J. J. Neto, "Contribuições à metodologia de construção de compiladores", Tese de Livre Docência, USP, São Paulo, 1993.

[9] J. J. Neto, "Adaptive Automata for Context-Sensitive Languages," *ACM SIGPLAN NOTICES*, Vol. 29, n. 9, pp. 115-124, September, 1994.

[10] J. J. Neto e M. Moraes, "Formalismo adaptativo aplicado ao reconhecimento de linguagem natural," em *Anais da Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, 19-21 de Julho, 2002, Orlando, Florida, EUA.

[11] M. K. Iwai, "Um formalismo gramatical adaptativo para linguagens dependentes de contexto," Tese de Doutorado, USP, São Paulo, 2000.

[12] N. Chomsky, "Three models for the description of language," IEEE Transactions on Information Theory, Vol. 2, n. 3, pp. 113-124, 1956.

[13] R. L. Rocha, "Um método de escolha automática de soluções usando tecnologia adaptativa.", Tese de Doutorado, USP, São Paulo, 2000.

[14] R. L. Rocha, "A Structural Operational Semantics Approach to Adaptive Devices", em *Anais do Congresso de Lógica Aplicada à Tecnologia*, 21-23 de Novembro, 2007, São Paulo.

Iúri Chaer nasceu na capital de São Paulo, Brasil, a três de Março de 1981. Formou-se Engenheiro Eletricista pela Escola Politécnica da USP ao final de 2003.

Trabalhou por cinco anos no setor privado em sistemas computacionais para tratamento de grandes massas de dados, e, posteriormente, participou do primeiro Programa de Residência em Tecnologia oferecido na Universidade de Minas Gerais, em 2007. Atualmente pesquisa Análise Semântica de Linguagens Naturais e Mecanismos de Inferência.

Ricardo Luis A. Rocha é natural do Rio de Janeiro-RJ e nasceu em 29/05/1960. Graduou-se em Engenharia Elétrica modalidade Eletrônica na PUC-RJ, em 1982. É Mestre e Doutor em Engenharia de Computação pela EPUSP (1995 e 2000, respectivamente). Suas áreas de atuação incluem Tecnologias Adaptativas, Fundamentos de Computação e Modelos Computacionais.

Dr. Rocha é membro da ACM (Association for Computing Machinery) e da SBC (Sociedade Brasileira de Computação).