# UMA PROPOSTA DE ENSINO DA DISCIPLINA DE LINGUAGENS FORMAIS E AUTÔMATOS PARA UM CURSO DE ENGENHARIA DA COMPUTAÇÃO

**Thiago Carvalho Pedrazzi** [1] - thiago.pedrazzi@poli.usp.br

**Ivone Penque Matsuno** [1,2] - ivone.matsuno@poli.usp.br

**Ricardo Luis de Azevedo da Rocha** [1] - luis.rocha@poli.usp.br

[1]Departamento de Engenharia de Computação e Sistemas Digitais – PCS
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav.3, nº.158
05508-900 – São Paulo – SP

[2] Departamento de Engenharia de Computação e Ciência da Computação
Faculdades Associadas de São Paulo – FASP
Rua José Antonio Coelho, nº 879 – Paraíso
04011-062 – São Paulo – SP

***Resumo:*** *No ensino de linguagens formais, os diferentes formalismos normalmente utilizados e suas respectivas notações fazem com que os alunos tenham dificuldades em perceber diretamente a evolução do poder computacional dos mesmos. Neste trabalho abordamos dois aspectos do ensino de linguagens formais: o uso de um modelo computacional embasado em um formalismo evolutivo, e o uso concomitante de ferramentas para auxiliar o ensino de Fundamentos da Computação. Apresentamos no artigo um breve resumo dos formalismos e das linguagens formais que são abordados a partir da notação empregada, uma descrição geral da ferramenta utilizada e um conjunto de exemplos para cada tipo de linguagem abordada e sua respectiva implementação na ferramenta.*

***Palavras-chaves:*** *Linguagens formais, Autômatos finitos, Metodologia de ensino.*

## 1. INTRODUÇÃO

No ensino de linguagens formais, os diferentes formalismos e suas respectivas notações fazem com que os alunos não percebam diretamente a evolução da expressividade computacional dos mesmos. Esta variedade de conceitos pode atrapalhar o aluno, visto que em cada tópico da disciplina uma classe de formalismo é adotada. Além disso, é necessário um certo nível de abstração para a familiarização e manipulação destes formalismos.

Nesta pesquisa o objetivo é apresentar uma proposta que aborde os formalismos de forma evolutiva, utilizando uma mesma notação durante todo o processo, a qual será adicionada gradativamente de recursos que aumentem a sua expressividade computacional, visando o acompanhamento da evolução das linguagens formais. Em paralelo a esta abordagem deve-se utilizar uma ferramenta que também possibilite uma melhor interação com os modelos.

Dentre as diversas ferramentas analisadas seguindo CHESÑEVAR, COBO, e YURCIK (2003) e as ferramentas desenvolvidas no laboratório de Linguagens e Tecnologias Adaptativas (LTA)[1], a ferramenta Adaptools[2] mostrou ser a mais aderente ao processo de ensino proposto para as disciplinas de Fundamentos da Computação, devido aos fatores discutidos nas próximas seções.

Este artigo está divido da seguinte forma: Na seção 2 é apresentado um breve resumo dos formalismos e linguagens formais que serão abordados. Na seção 3 é apresentada uma descrição geral do Adaptools. Na seção 4 são apresentados os exemplos para cada tipo de linguagem abordada e sua respectiva implementação na ferramenta. Na seção 5 são apresentados as conclusões e resultados da aplicação dessa abordagem.

## 2. PROPOSTA PARA LINGUAGENS FORMAIS

Com o objetivo de simplificar a diversidade das notações no ensino de linguagens formais e dos seus respectivos formalismos propõe-se o uso de uma única notação para o tratamento de todas as classes de linguagens em conjunto com a utilização de um único ambiente computacional para a simulação dos formalismos empregados em sala de aula.

Pretende-se com isso diminuir a curva de aprendizado dos formalismos, e concentrar os esforços dos alunos na assimilação dos conceitos da disciplina.

As disciplinas de fundamentos da computação baseiam-se basicamente no mesmo processo de ensino: inicialmente, com os conceitos mais simples das teorias de linguagens formais e autômatos (linguagens do tipo 3), avançando gradualmente ao nível mais complexo destas teorias (linguagens do tipo 0). O diagrama na figura 1 indica o grau de abrangência de cada tipo de linguagem. Observa-se, por exemplo, que o tipo de linguagem 1 (linguagens sensíveis ao contexto) abrange todos os casos de linguagens do tipo 2 (linguagens livres de contexto) que, por sua vez, abrange todos os casos de linguagens do tipo 3 (linguagens regulares).

Na tabela 1 estão associados os tipos de linguagens com os respectivos formalismos reconhecedores clássicos. Na terceira coluna desta tabela encontra-se a proposta deste trabalho.

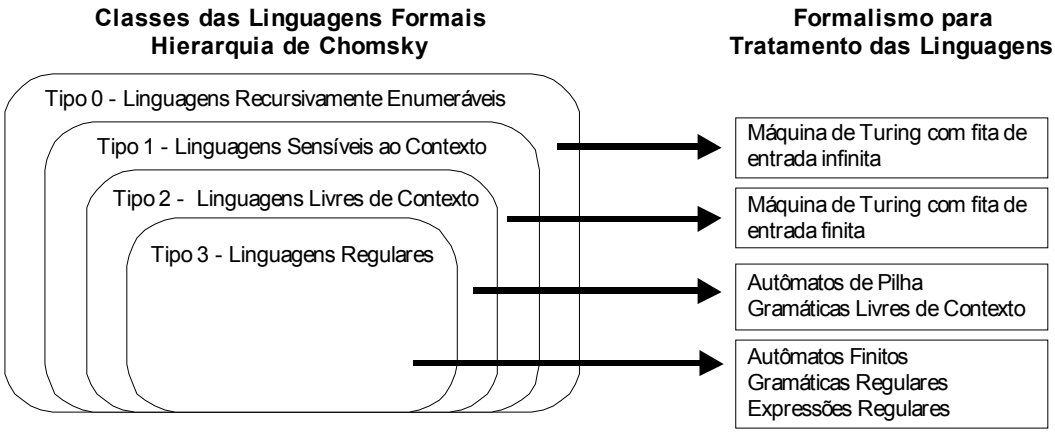


Figura 1 – Hierarquia de Chomsky

Tabela 1 – Linguagens, reconhecedores e propostas.

[1] Site do LTA: www.pcs.usp.br/~lta

[2] Desenvolvida por Hemerson Pistori em PISTORI (2003)

| Linguagem | Reconhecedor | Proposta |
|---|---|---|
| Tipo 3: Linguagens Regulares | Autômatos Finitos | Autômatos Finitos |
| Tipo 2: Linguagens Livres de Contexto | Autômatos de Pilha | Autômatos de Pilha Estruturado |
| Tipo 1: Linguagens Sensíveis ao Contexto | Máquina de Turing | Autômatos Adaptativos |
| Tipo 0: Linguagens Enumeráveis Recursivamente | Máquina de Turing | Autômatos Adaptativos Extendidos |

Em geral, os alunos apresentam maior facilidade na compreensão de diagramas de estados e transições (notação de grafos orientados). Esta prerrogativa foi utilizada no momento da avaliação do ambiente computacional mais interessante à proposta de ensino.

A abordagem utilizada no ensino de tratamento sintático de linguagens será operacional, ou seja, definida sobre a operação de determinados dispositivos. Este dispositivo deve ser suficientemente simples para facilitar a visualização do processo do reconhecimento e não permitir ambigüidades em sua operação.

Exemplos destas máquinas (AFD, APE e AA) serão apresentados em detalhe nas próximas seções.

## 2.1. Linguagens regulares

Para o estudo das linguagens regulares, conforme anteriormente indicado, utilizar-se-á um dispositivo reconhecedor de cadeias chamado AFD (Autômato Finito Determinístico). Esta classe de dispositivo representa uma categoria de reconhecedores clássicos, sendo o conceito dos mesmos muito bem assimilado a priori.

Um AFD M pode ser representado por uma quíntupla da forma $M = (Q, \Sigma, \delta, q_0, F)$, onde:

M: Nome do autômato representado.
Q: Conjunto de estados do autômato M.
$\Sigma$: Conjunto de símbolos os quais compõe o alfabeto da linguagem.
$\delta$: Conjunto de transições do autômato.
$q_0$: Estado inicial do autômato M.
F: Conjunto de estados finais do autômato M.

Cada transição do conjunto $\delta$ será mapeada segundo $\mathbf{Q} \times \mathbf{S} \rightarrow \mathbf{Q}$, descrevendo que um estado e um símbolo de entrada mapeiam um estado. Para a ilustração do formalismo, mostrar-se-á um exemplo de autômato finito. Este autômato M1 é capaz de reconhecer cadeias de entrada as quais contenham as subcadeias 'aa' ou 'ab' como prefixo.

$M1 = (Q, \Sigma, \delta, q_0, F)$ com:
$Q = \{q_0, q_1, q_2, q_3\}$,
$\Sigma = \{\text{'a'},\text{'b'}\}$,
$\delta = \{(q_0,a) \rightarrow q_1, (q_0,b) \rightarrow q_3, (q_1,a) \rightarrow q_2, (q_1,b) \rightarrow q_2, (q_2,a) \rightarrow q_2, (q_2,b) \rightarrow q_2,$
$(q_3,a) \rightarrow q_3, (q_3,b) \rightarrow q_3\}$
$q_0 = q_0$,
$F = \{q_2\}$.

Em HOPCROFT e ULLMAN (1979) maiores detalhes sobre a natureza e o funcionamento do autômato finito podem ser encontrados.

## 2.2. Linguagens livres de contexto

As linguagens livres de contexto caracterizam-se por representar o aninhamento sintático na formação das palavras. O autômato finito simplesmente não consegue tratar esse tipo de linguagem, pois é necessário controlar a parte inicial da cadeia com a parte final. Desta forma,

para tratar esse tipo de linguagem acrescenta-se uma pilha que serve como memória para o autômato.

Dentre as diversas formalizações existentes para o autômato de pilha, propõe-se o uso do Autômato de Pilha Estruturado (APE) proposto em NETO (1987). Nesta formalização, o autômato é uma máquina de estados composta por um conjunto de submáquinas com transições que consomem símbolos de entrada ou fazem chamada ou retorno de outra submáquina. Estas submáquinas são na verdade autômatos finitos, diferenciando-se dos mesmos apenas pelas transições de chamada e retorno de submáquinas.

Uma chamada de submáquina implica em guardar o próximo estado na pilha e continuar o reconhecimento da cadeia a partir do estado inicial da submáquina em questão. Ao término do processamento desta cadeia e sendo o estado corrente um estado final, a análise da cadeia continuará, então, a partir do estado armazenado no topo da pilha, o qual será desempilhado. Este procedimento é chamado de transição de retorno de submáquina.

Um APE N pode ser representado pela óctupla $N= (Q, A, \Sigma, \Gamma, P, Z_0, q_0, F)$ em que:

$Q$ = Conjunto de estados do APE (todos os estados de todas as submáquinas),

$\Sigma$ = Conjunto de símbolos os quais compõe o alfabeto da linguagem,

$\Gamma$ = Conjunto de símbolos os quais compõe o alfabeto da pilha,

$Z_0$ = Símbolo inicial da pilha,

$q_0$ = Estado inicial do APE,

$F$ = Conjunto de estados finais do APE,

$A$ = Conjunto de submáquinas do formato $a_i = (Q_i, \Sigma_i, P_i, E_i, S_i)$, onde:

$Q_i \subseteq Q$ é o conjunto dos estados de $q_{ij}$ da submáquina $a_i$

$\Sigma_i \subseteq \Sigma$ é o conjunto de símbolos de entrada da submáquina $a_i$

$E_i \subseteq Q_i$ é estado inicial da submáquina $a_i$ ( contém o único estado de entrada da submáquina $a_i$ ($E_i=\{q_{i0}\}$ )

$S_i \subseteq Q_i$ é conjunto de estados finais da submáquina $a_i$

$P_i \subseteq P$ é conjunto de produções da submáquina $a_i$ que podem ser representadas da seguinte forma:

Transições internas de submáquina $a_i$ : $(\gamma, q_{ik}, \alpha) \rightarrow (\gamma, q_{ij}, \beta)$

Transições de chamada de submáquina $a_m$ : $(\gamma, q_{ik}, \alpha) \rightarrow (\gamma(i, j), q_{m0}, \beta)$

Transições de retorno de submáquina $a_m$ : $(\gamma(m, n), q_{ik}, \alpha) \rightarrow (\gamma, q_{mn}, \beta)$

Nessa proposta a pilha é utilizada somente para o controle do aninhamento sintático, que é a característica que difere uma construção livre de contexto de uma construção regular. Desta maneira, a pilha é utilizada de maneira ótima. A cada nova instância do aninhamento, uma submáquina será chamada, sendo esta chamada a responsável pelo tratamento da parte interna do aninhamento.

## 2.3. Linguagens sensíveis ao contexto

Como podemos observar, o formalismo conhecido como Autômato de Pilha Estruturado (APE) não possui a expressividade computacional necessária à descrição desta dependência de contexto. Faz-se necessária então a apresentação de um novo formalismo, capaz corresponder à nova necessidade.

Normalmente, o dispositivo utilizado no ensino deste tópico nas disciplinas de Fundamentos da Computação é a Maquina de Turing. Como é sabido, a Maquina de Turing tem pouca semelhança (à primeira vista) com os dispositivos anteriormente utilizados, criando para o aluno uma separação conceitual entre Linguagens Regulares, Linguagens Livres de Contexto e Linguagens Sensíveis ao Contexto. O aluno tem este tópico como isolado, não percebendo qualquer ligação com os outros assuntos abordados anteriormente.

Didaticamente é mais interessante a modificação de algum formalismo já dominado, visando a redução da curva de aprendizado, do que a introdução de um novo formalismo, ocasionando um novo processo de aprendizado.

Visando a eliminação desta barreira entre os tópicos de Fundamentos da Computação, será introduzido o formalismo do Autômato Adaptativo proposto em NETO (1993) com o intuito de substituição da Máquina de Turing como dispositivo reconhecedor de linguagens sensíveis ao contexto. Este novo formalismo é totalmente fundamentado sobre o já ensinado e dominado autômato de pilha estruturado (APE), havendo apenas a adição de algumas características que permitem automodificação.

Estas características lhe conferem a possibilidade de, a cada nova transição, poder alterar a sua própria estrutura, ou seja, adicionar ou remover estados e transições, por meio das chamadas às funções adaptativas. Isso lhe permite memorizar em sua própria estrutura as construções necessárias para o tratamento das dependências de contexto.

As funções adaptativas são declaradas basicamente em termos de ações adaptativas elementares, que são:

**Pesquisa** (representada pelo símbolo **?**),
**Inclusão** (representada pelo símbolo +),
**Exclusão** (representada pelo símbolo -)

Além de uma estrutura de suporte, como declaração de variáveis, geradores, parâmetros entre outras. Maiores informações, incluindo a completa formalização do conceito do autômato adaptativo podem ser encontradas em NETO (1993) e NETO (1994). Em ROCHA e NETO (2000), encontra-se a demonstração da equivalência de expressividade computacional entre a Máquina de Turing e o Autômato Adaptativo.

## 3. AMBIENTE COMPUTACIONAL ADAPTOOLS

A ferramenta Adaptools implementa um ambiente de execução para o formalismo conhecido como autômato adaptativo. Este formalismo, como pode ser observado em ROCHA e NETO (2000) é Turing-poderoso, ou seja, tem o mesmo poder da máquina de Turing. Isso quer dizer que qualquer 'algoritmo' pode ser traduzido em um autômato adaptativo e, portanto, executado na Adaptools.

Os conceitos AFD e APE são considerados subcasos do conceito de Autômato Adaptativo. Portanto, exemplos de todos os conceitos anteriormente ensinados podem ser executados na ferramenta Adaptools, o que lhe confere um aspecto unificador, pode ser utilizada desde os conceitos mais básicos de reconhecedores (autômatos finitos) até o conceito mais avançado (autômatos adaptativos).

Esta característica é muito interessante sob o ponto de vista didático já que não há necessidade do aprendizado de uma nova ferramenta ao longo do processo didático de Fundamentos da Computação, como normalmente ocorre, além disso, o resultado obtido em MATSUNO, PEDRAZZI e ROCHA (2004) mostra que o formalismo mais propício para o aprendizado dos alunos é de fato o formalismo operacional, ou seja, baseado em autômatos, e com baixa complexidade. No decorrer do curso, conforme há aumento de complexidade dos recursos computacionais, alguns recursos da ferramenta podem ser ensinados gradativamente aos alunos, havendo um aprendizado incremental na manipulação da ferramenta.

### 3.1 Notação e aspectos de utilização

Devido à linguagem a qual foi implementada, a ferramenta Adaptools é dotada de uma aparência e navegabilidade peculiares. Seus recursos gráficos permitem a exibição do autômato em formato gráfico (grafo orientado), além do formato padrão de regras (algébrico). Entre as opções de execução, pode-se controlar a velocidade da simulação, executar uma simulação passo-a-passo, e visualização de variáveis, entre outros.

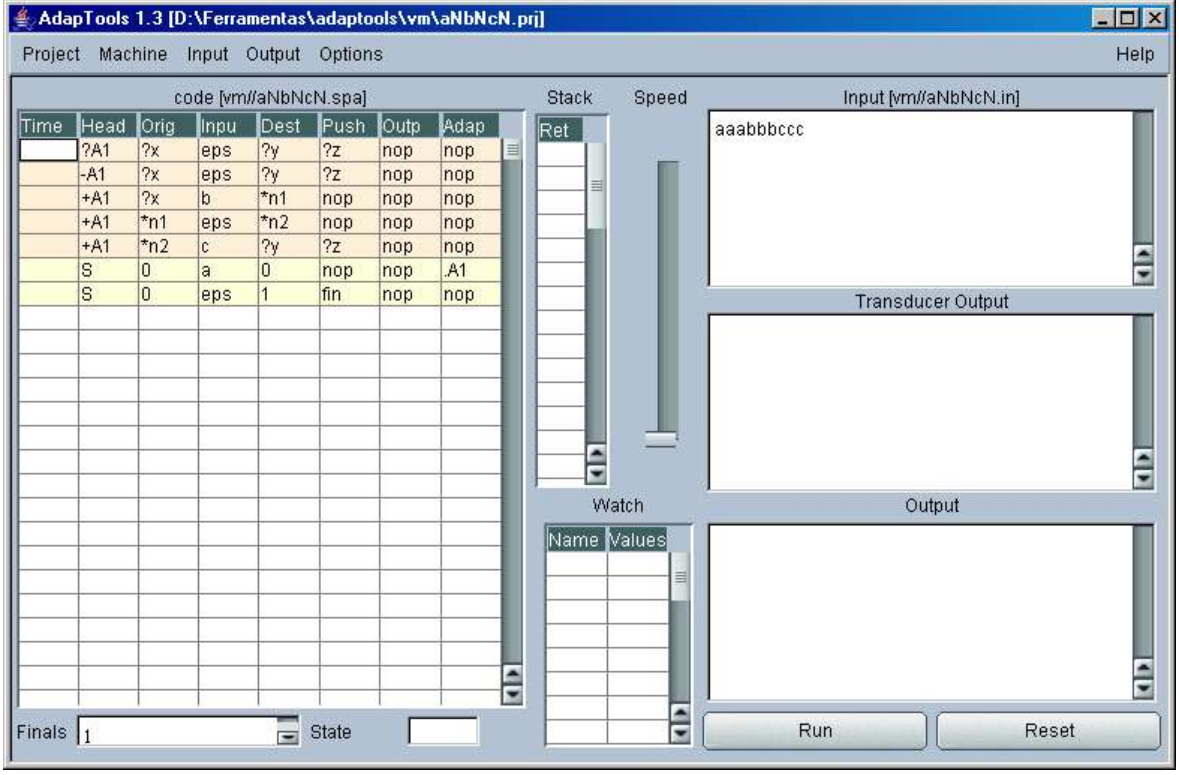


Figura 2 – Tela inicial do ambiente computacional Adaptools

O interior da Adaptools é composto por uma máquina virtual que executa uma versão simplificada do autômato adaptativo originalmente descrito em NETO (1993). Esta versão simplificada consiste de sete campos: cabeçalho, estado de origem, símbolo de entrada, estado de destino, empilha, símbolo de saída e ação adaptativa.

O campo **cabeçalho** (**Head**) descreve o nome da submáquina ou o nome da função adaptativa precedida por um dos seguintes símbolos: **?** (ação elementar de pesquisa), + (ação elementar de inserção) ou – (ação elementar de remoção).

O campo **estado de origem** (**Orig**) indica o nome do estado corrente.

O campo **símbolo de entrada** (**Inpu**) índica o símbolo de entrada.

O campo **estado de destino** (**Dest**) indica o nome do próximo estado.

O campo **empilha** (**Push**) indica o nome do estado a ser empilhado na pilha (estado de retorno de submáquina).

O campo **símbolo de saída** (**Outp**) indica o símbolo de saída (no caso de transdutores).

O campo **ação adaptativa** (**Adap**) indica o nome de uma função adaptativa precedida ou sucedida por um ponto (**.**), indicando se a ação deve ser executada antes ou após a execução da regra correspondente.

Maiores detalhes sobre os aspectos de implementação e utilização do Adaptools podem ser encontrados em PISTORI (2003), e PISTORI e NETO (2003).

## 4. PROCEDIMENTO PARA A SIMULAÇÃO DE AUTÔMATOS

Nesta seção serão apresentados os procedimentos de conversão de autômatos reconhecedores de linguagens do tipo 1, 2 e 3 para a notação do Adaptools. Ilustrar-se-á a proposta deste trabalho da seguinte forma: para cada tipo de linguagem apresenta-se a construção do autômato para reconhecê-la e suas respectivas notações: formal (algébrica), em grafo-orientado e na ferramenta de simulação.

### 4.1. Linguagens regulares e autômatos finitos determinísticos

Conforme explicado na seção 2.1, o formalismo abordado para o tratamento de linguagens regulares é o Autômato Finito Determinístico. Na simulação de autômatos finitos

determinísticos no ambiente computacional Adaptools serão utilizadas somente as colunas:

Cabeçalho (Head), Estado de Origem (Orig), Símbolo de Entrada (Inpu), Estado de Destino (Dest) e Empilha (Push), esta última somente para a indicação do estado final do autômato, devendo todas as outras células ser completadas com o caractere especial **nop**.

O procedimento para a conversão de um AFD M = (Σ, Q, δ, q0, F) para a notação do adaptools é descrito abaixo.

$\forall\ q_i \in Q$, $\forall \alpha \in \Sigma$, para cada regra da forma $\delta(q_i,\alpha) = q_j$, deve-se criar uma das seguintes regras no Adaptools:

$(M, q_i, \alpha, q_j, nop, nop, nop)$, se $q_j \notin F$ , ou

$(M, q_i, \alpha, q_j, fin, nop, nop)$, se $qj \in F$.

A primeira regra na tabela deve indicar o estado inicial de M.

Para exemplificar, considere a linguagem regular $L_1$={w ∈ {a,b}* | w tem 'aa' ou 'ab' como prefixo}, o AFD M1 para reconhecer $L_1$ é representado na figura 3.

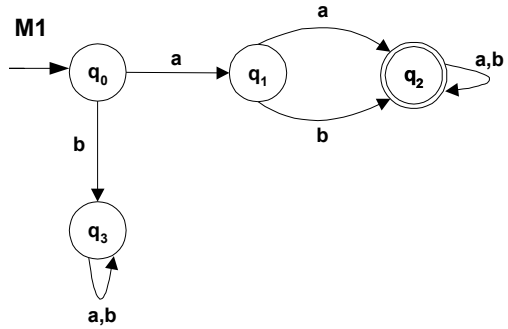


Figura 3 – AFD para reconhecer cadeias com prefixo 'aa' ou 'ab'

Tabela 2 – Tabela com transições e formato do Adaptools do autômato M1

| Transição | Head | Orig | Inpu | Dest | Push | Outp | Adap |
|---|---|---|---|---|---|---|---|
| $\delta(q_0,a)=q_1$ | M1 | 0 | a | 1 | nop | nop | nop |
| δ(q0,b)=q3 | M1 | 0 | b | 3 | nop | nop | nop |
| δ(q1,a)=q2 | M1 | 1 | a | 2 | fin | nop | nop |
| δ(q1,b)=q2 | M1 | 1 | b | 2 | fin | nop | nop |
| δ(q2,a)=q2 | M1 | 2 | a | 2 | fin | nop | nop |
| δ(q2,b)=q2 | M1 | 2 | b | 2 | fin | nop | nop |
| δ(q3,a)=q3 | M1 | 3 | a | 3 | nop | nop | nop |
| δ(q3,b)=q3 | M1 | 3 | b | 3 | nop | nop | nop |

## 4.2. Linguagens livres de contexto e autômatos de pilha estruturados

Para a simulação de autômatos de pilha estruturados (APE) no Adaptools deve-se acrescentar as transições que simulam a chamada e o retorno de submáquina. Observa-se que ao ampliar a capacidade de representação da linguagem, os formalismos, tanto dos APEs quanto da notação no Adaptools, sofrem alterações mínimas, e os conceitos aprendidos até esse tipo de linguagem não são totalmente modificados, e sim incrementados sutilmente. Neste novo formato, a coluna **empilha (Push)** será também utilizada para o armazenamento do estado de retorno e não somente para indicar os estados finais como no caso da representação do AFD.

Para a representação de um APE M (composto de submáquinas $A_i$) na notação do Adaptools, cada transição será convertida da seguinte forma:

Passo 1: Como no AFD, a primeira regra de transição deve indicar estado inicial da submáquina em questão.

$(\gamma, q_{ik}, \alpha) \rightarrow (\gamma, q_{ij}, \beta) \quad \Rightarrow \quad (A_i, q_{ik}, \alpha, q_{ij}, nop, nop, nop)$

A coluna **cabeçalho** (**Head**) indicará o nome da submáquina a qual a transição pertence.

Todas as transições internas de submáquina seguem o mesmo padrão (e, portanto, mesmo formato) das transições do AFD.

Transições de chamada de submáquina $A_m$:

$(\gamma, q_{ik}, \alpha) \rightarrow (\gamma(i, j), q_{m0}, \alpha) \quad \Rightarrow \quad (A_m, q_{ik}, eps, q_{m0}, q_{ij}, nop, nop)$

Transições de retorno de submáquina $A_m$:

$(\gamma(m, n), q_{ik}, \alpha) \rightarrow (\gamma, q_{mn}, \alpha) \quad \Rightarrow \quad (A_i, q_{ik}, \alpha, q_{ij}, nop, nop, nop)$

$\gamma(q_i,\alpha) = qj$ => $(M, q_i, \alpha, qj, nop, nop, nop), \forall\alpha \in \{\Sigma \cup \varepsilon\}$, se $qj \notin F$

$\gamma(q_i,\alpha) = qj$ => $(M, q_i, \alpha, qj, fin, nop, nop), \forall\alpha \in \{\Sigma \cup \varepsilon\}$, se $qj \in F$

$\gamma(qi,\alpha) = qj$ => $(M, q_i, eps, q_0', qj, nop, nop), \forall\alpha \in V, \forall q_j \in Q$, $q_0'$ é o estado inicial da submáquina $\alpha$ e $q_j$ é o estado de retorno, quando a submáquina a finalizar sua execução.

Passo 2: Para todos os estados finais, incluir a seguinte regra de transição.

$(M, q_f, eps, pop, nop, nop, nop), \forall q_f \notin F$

Esse tipo de transição é utilizada na indicação de transições de retorno de submáquina. Se a pilha estiver vazia encerra-se a execução, caso contrário continua a análise a partir do estado retirado da pilha.

Considere o APE M2 para a linguagem livre de contexto $L_2 = \{w \in \{a,b\}^* \mid w=a^nb^n, n\geq 0\}$, representado na figura 4.

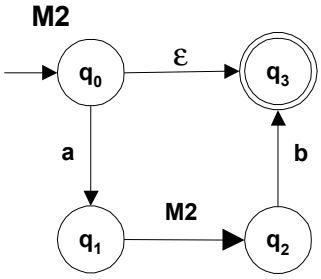


Figura 4 – APE para reconhecer cadeias do tipo $a^nb^n$.

Tabela 3 – Tabela com transições e formato do Adaptools do autômato M2

| Transição | Head | Orig | Inpu | Dest | Push | Outp | Adap |
|---|---|---|---|---|---|---|---|
| $\delta(q_0,a)=q_1$ | M2 | 0 | a | 1 | nop | nop | nop |
| $\delta(q_0,\varepsilon)=q_3$ | M2 | 0 | eps | 3 | fin | nop | nop |
| $\delta(q_1,M2)=q_2$ | M2 | 1 | eps | 2 | 0 | nop | nop |
| $\delta(q_2,b)=q_3$ | M2 | 2 | b | 3 | fin | nop | nop |
| | M2 | 3 | eps | pop | nop | nop | nop |

Considere o APE M3 para a linguagem livre de contexto $L_3 = \{wcw^R \mid w \in \{a,b\}^*\}$, representado na figura 5.

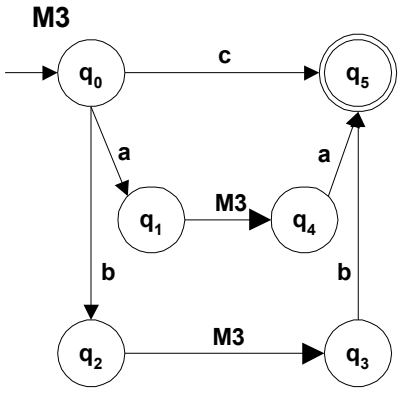


Figura 5 – APE para reconhecer cadeias no formato $wcw^R$

Tabela 4 – Tabela com transições e formato do Adaptools do autômato M3

| Transição | Head | Orig | Inpu | Dest | Push | Outp | Adap |
|---|---|---|---|---|---|---|---|
| $\delta(q_0,a)=q_1$ | M3 | 0 | c | 1 | fin | nop | nop |
| $\delta(q_0,b)=q_2$ | M3 | 0 | a | 2 | fin | nop | nop |
| $\delta(q_0,c)=q_5$ | M3 | 0 | b | 4 | 0 | nop | nop |
| $\delta(q_1,M3)=q_4$ | M3 | 2 | eps | 0 | 3 | nop | nop |
| $\delta(q_2,M3)=q_3$ | M3 | 3 | a | 1 | fin | nop | nop |
| $\delta(q_3,b)=q_5$ | M3 | 4 | eps | 0 | 5 | nop | nop |
| $\delta(q_4,a)=q_5$ | M3 | 5 | b | 1 | fin | nop | nop |
| | M3 | 1 | eps | pop | nop | nop | nop |

### 4.3. Linguagens sensíveis ao contexto e autômatos adaptativos

Na simulação de Autômatos Adaptativos (AA) no ambiente computacional Adaptools, será necessária a utilização de toda a sua capacidade como reconhecedor, ou seja, além de todos os recursos utilizados para simulação de Autômatos de Pilha Estruturados (APE), será necessária a utilização do recurso de declaração de funções adaptativas. Isso será feito através da utilização do campo **ação adaptativa** (**Adap**) de uma transição, indicando se existe uma

chamada a Função Adaptativa associada a esta transição, seguindo a notação descrita na seção 3.1.

Não será necessária a conversão de nenhum formato para o formato do Adaptools, visto que o conceito pode ser mapeado diretamente.

Como exemplo, será exercitado um caso clássico de reconhecimento de cadeias para a linguagem sensível a contexto $L_4 = \{w \in \{a, b, c\}^* \mid w=a^nb^nc^n, n \geq 0\}$ (PISTORI e NETO 2003):

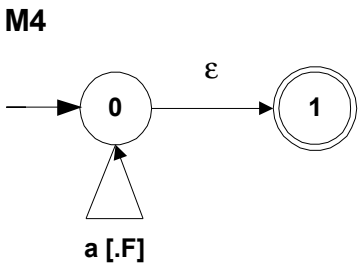


Figura 6 – Autômato Adaptativo para reconhecer cadeias do tipo $a^nb^nc^n$

Tabela 5 – Tabela com transições e formato do Adaptools do autômato M4

| Transição | Head | Orig | Inpu | Dest | Push | Outp | Adap |
|---|---|---|---|---|---|---|---|
| | ?F | ?x | eps | ?y | ?z | nop | nop |
| | -F | ?x | eps | ?y | ?z | nop | nop |
| | +F | ?x | b | *n1 | nop | nop | nop |
| | +F | *n1 | eps | *n2 | nop | nop | nop |
| | +F | *n2 | c | ?y | ?z | nop | nop |
| $\delta(q_0,a)=q_0$ | M4 | 0 | a | 0 | nop | nop | .F |
| $\delta(q_0,\varepsilon)=q_1$ | M4 | 0 | eps | 1 | fin | nop | nop |

O autômato descrito acima funciona da seguinte maneira: recebendo um caractere ‘a’, a transição executa a função adaptativa F, a qual procura por uma transição em vazio, remove esta transição e inclui três novas transições e 2 novos estados, respectivamente para o consumo dos caracteres ‘b’ e ‘c’ correspondentes ao caractere ‘a’ recebido anteriormente. Isso é executado a cada caractere ‘a’ recebido. Quando receber o caractere ‘b’, indicando que acabaram-se os caracteres ‘a’s, haverá uma transição para reconhecer cada ‘b’ necessário (na mesma quantidade de caracteres ‘a’ reconhecidos anteriormente) e após haverá uma transição para reconhecer cada ‘c’.

Este exemplo mostra a simplicidade de soluções para problemas complexos que a tecnologia adaptativa permite. Em NETO (2000) podem ser encontrados outros problemas de alta complexidade solucionados por meio de autômatos adaptativos.

## 5. CONSIDERAÇÕES FINAIS

Experimentou-se esta nova proposta de ensino em um curso de Engenharia da Computação de uma universidade pública, comparando os resultados com a proposta de ensino tradicional, avaliada em um curso de Engenharia da Computação de uma faculdade particular cujos resultados estão em MATSUNO, PEDRAZZI e ROCHA (2004).

Analisando o formalismo mais facilmente assimilado, a quase totalidade optou pelos autômatos, tanto por seu modo de funcionamento (processando e aceitando ou rejeitando a cadeia de entrada) quanto pela sua possível representação visual (diagrama de estados e transições).

Isto motiva a alteração da proposta de trabalho atual, a qual utiliza o autômato de estados como principal formalismo apenas no caso das linguagens regulares (tópico inicial da disciplina). No próximo tópico clássico das disciplinas de Fundamentos da Computação (linguagens livres de contexto), as gramáticas livres de contexto são utilizadas, gerando uma certa dificuldade para os alunos que não compreenderam bem as propriedades das gramáticas regulares.

## REFERÊNCIAS BIBLIOGRÁFICAS

AHO, A. V.; ULLMAN, J. D. **The theory of parsing, translation and compiling - vol 1.** Prentice Hall, 1972.

CHESÑEVAR, C. I.; COBO, M. L.; YURCIK, W. Using Theoretical Computer Simulators to Formal Languages and Automata Theory. In: SIGCSE'2003, 2003, Reno, Nevada. **Proceedings.** ACM, 2003. Disponível em: http://cs.uns.edu.ar/~cic/publications.htm.
HOPCROFT, J. E.; ULLMAN, J. D. **Introduction to automata theory, languages and computation.** Addison-Wesley, 1979.
MATSUNO, I. P.; PEDRAZZI, T. C.; ROCHA, R. L. A. Avaliação do Uso de Conceitos de Linguagens Formais e Autômatos em um Curso de Engenharia da Computação. In: Resumo Aceito para o COBENGE 2004, **Anais**, UnB:Brasília, 2004.
NETO, J. J. **Introdução à compilação.** Rio de Janeiro: Editora LTC, 1987.
NETO, J. J. **Contribuição à metodologia de construção de compiladores.** 1993. Tese (Livre docência em Engenharia) – Escola Politécnica, Universidade de São Paulo, São Paulo.
NETO, J. J. Adaptive automata for context-dependent languages. **SIGPLAN Notices, vol 29, no. 9.** September, 1994.
NETO, J. J. Solving Complex Problems Efficiently with Adaptive Automata.
**CIAA 2000 - Fifth International Conference on Implementation and Application of Automata.** London, Ontario, Canadá, Julho 2000.
PISTORI, H. **Tecnologia adaptativa em engenharia da computação: estado da arte e aplicações.** 2003. Tese (Doutorado em Engenharia Elétrica) - Escola Politécnica, Universidade de São Paulo, São Paulo.
PISTORI, H.; NETO, J. J. A free software for the development of adaptive automata. **Proceedings of the IV Workshop on Free Software - WSL (IV International Forum on Free Software).** Porto Alegre. s.n., 2003.
ROCHA, R. L. A.; NETO, J. J. Autômato adaptativo, limites e complexidade em comparacao com maquina de Turing. In: Proceedings of the second Congress of Logic Applied to Technology – LAPTEC 2000. **Proceedings,** São Paulo, p. 33-48, 2000.

# A FORMAL LANGUAGES AND AUTOMATA THEORY TEACHING PROPOSAL FOR A COMPUTING ENGINEERING COURSE

***Abstract:*** *In teaching formal languages, the different formalisms usually taught and their respective notations add some hardships to students, who can't see their power evolution. In this paper we approach two aspects of education of formal languages: the use of a computational model based in an evolutional formalism, and the concomitant use of tools to support the education of Foundations of Computer Science. We show a brief summary of the formalisms and the formal languages in their notation, a general description of the used tool and a set of examples for each type of formal language and its respective implementation in the tool.*

***Keywords:*** *Formal Languages, Finite Automata, Teaching Methodology.*