AdapTools: Aspectos de Implementação e Utilização

Hemerson Pistori e João José Neto
Universidade de São Paulo, Escola Politécnica,
São Paulo, Brasil, 05508-900
joao.jose@poli.usp.br

Abstract

This work presents AdapTools, a computational environment in which adaptive automata can be project, implemented and tested. The computational environment includes debugging tools, like step-by-step execution, graphical visualization of stacks and variables and adaptive automata animation. The package also offers project control mechanisms, simultaneous execution of multiple machines and a large amount of examples.

Keywords: Adaptive Automata, Software Engeneering, Free-Software.

Resumo

Neste trabalho apresentaremos o AdapTools, um ambiente computacional em que autômatos adaptativos podem ser projetados, implementados e testados. O ambiente computacional inclui recursos de depuração, como execução passo-a-passo, visualização de variáveis e pilhas, e animação gráfica; bem como mecanismos para controle de projetos, execução simultânea de múltiplos dispositivos (com comunicação entre si) e uma série de exemplos de autômatos.

Palavras chave: Autômatos Adaptativos, Engenharia de Software e Software-Livre.

1 Introdução

Neste trabalho apresentaremos o AdapTools [1], um ambiente computacional em que autômatos adaptativos [2, 3] podem ser implementados, depurados e experimentados. O núcleo deste sistema é uma máquina virtual que executa uma versão ligeiramente modificada de um autômato adaptativo. As modificações no formalismo original visam principalmente a simplificação e a uniformização no formato de especificação de transições internas, transições externas e ações adaptativas elementares. Com isto, todos os elementos do dispositivo passam a poder ser apresentados através de uma única tabela. O ambiente computacional inclui ainda recursos de depuração, como execução passo-a-passo, visualização de variáveis e pilhas, e animação gráfica; bem como mecanismos para controle de projetos, execução simultânea de múltiplos dispositivos (com comunicação entre si) e uma série de exemplos de autômatos.

A próxima seção apresenta a versão modificada de autômatos adaptativos utilizada no AdapTools. Em seguida, apresentamos algumas características da interface do sistema com o usuário. Na seção 4 mostraremos como rotinas semânticas podem ser atreladas a um autômato adaptativo. As possibilidades da integração entre diferentes autômatos na construção de sistemas mais complexos e a utilização da máquina virtual do AdapTools em sistemas externos são discutidas nas seções 5 e 6. A seção 8 aponta algumas direções e melhorias previstas para as futuras versões do AdapTools.

2 Autômatos Adaptativos no AdapTools

O formato escolhido para representar um autômato adaptativo no AdapTools é uma tabela com 7 colunas: (1) cabeçalho (*head*), (2) estado de origem (*orig*), (3) símbolo de entrada (*inpu*), (4) estado de destino (*dest*), (5) empilha (*push*), (6) símbolo de saída (*outp*) e (7) ação adaptativa (*adap*). As linhas desta tabela são de 3 tipos básicos: transição interna, transição externa (chamadas de sub-máquina e retorno) e ação adaptativa elementar. A primeira coluna desta tabela, o cabeçalho, contém o nome da sub-máquina ou da função adaptativa à qual o elemento (na linha) pertence. No caso de linhas referentes a ações adaptativas elementares, o nome da função adaptativa é precedida de um símbolo indicando o tipo da ação adaptativa elementar: ?, − e +, para ações elementares de consulta, remoção e inserção, respectivamente. As colunas de estados de origem e destino, e símbolos de entrada e saída, carregam justamente o que os seus nomes

[1] Versão 1.4.2

indicam (para se utilizar o AdapTools na execução de simples transdutores de estados finitos, estas são as únicas colunas necessárias). Estas 4 colunas poderão conter também variáveis, geradores e parâmetros, no lugar de constantes, no caso das linhas que representam ações adaptativas elementares. Seguindo a proposta apresentada em [5], variáveis, geradores e parâmetros são implicitamente declarados, sendo identificados pelos prefixos ?, $*$ e %, respectivamente [2].

As demais colunas estão relacionadas com a especificação das camadas que extendem o transdutor de estados finitos para um autômato de pilha estruturado e autômato adaptativo. A coluna empilha contém o endereço (estado) de retorno de uma chamada de sub-máquina. Retornos de sub-máquina são identificados pela presença da palavra reservada *pop*, na coluna de estado de destino. Ações adaptativas são acopladas à transições através de última coluna. Os símbolos desta coluna possuem o formato *A.P*, onde *A* e *P* são nomes de funções adaptativas anteriores (A) e posteriores (P), ambos opcionais, seguidos ou não de uma seqüência, entre parênteses, de parâmetros, separados por vírgula. A palavra reservada *nop* deve ser inserida em todos os campos que não estão sendo utilizados na especificação de um determinado dispositivo.

Ainda para evitar a utilização de estruturas extras, além da tabela, optamos por codificar os estados iniciais e finais do autômato da seguinte maneira: a coluna empilha deve conter a palavra reservada fin, quando o estado de destino for um estado final (além disto, o estado de origem das transições de retorno de sub-máquina também é automaticamente identificado pela máquina virtual como um estado final). O estado inicial de uma sub-máquina será sempre aquele que se encontra na primeira linha cujo cabeçalho contém o nome da sub-máquina. A listagem abaixo descreve cada uma das palavras reservadas e construções especiais utilizadas no AdapTools:

nop Pode ser utilizado apenas nas colunas 5, 6 e 7 para indicar ausência de estado de retorno, símbolo de saída ou ação adaptativa.

eps Representa a cadeia vazia, ϵ.

pop Quando o estado destino é *pop* temos um transição de retorno de sub-máquina. Ou seja, o próximo estado será retirado (*pop*), em tempo de execução, da pilha de chamadas de sub-máquinas.

fin Pode ser colocado na coluna empilha para indicar que o estado de destino da transição é um estado final.

spc Usado na coluna de símbolo de entrada para representar os caracteres ASCII que, em geral, funcionam como separadores: espaço, salto de linha e tabulação.

?sta Variável especial que pode ser utilizada apenas em ações adaptativa elementares. Em tempo de execução, é instanciada com o valor do estado atual do autômato.

?inp Variável especial instanciada com o valor do próximo símbolo da entrada.

As palavras reservadas seguintes podem ser utilizadas apenas na coluna de símbolo de entrada. Elas simplificam a especificações de determinados conjuntos de transições, que de outra forma, exigiria a inserção de várias linhas na tabela que define um autômato:

n..m Faixa de valores ASCII de *n* a *m*. Por exemplo, a presença do valor "*p..t*" na coluna de símbolo de entrada de uma transição indica que a mesma consome qualquer um dos símbolos *p*, *q*, *r*, *s* e *t*.

digit O mesmo que 0..9;

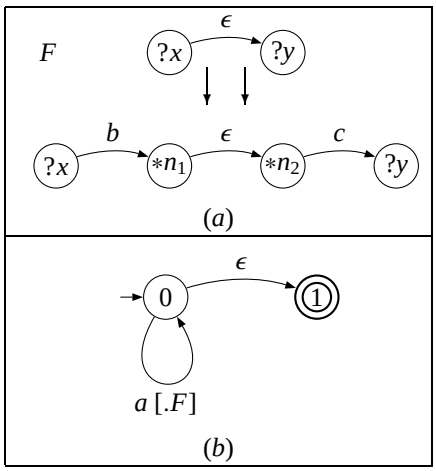
letter O mesmo que *a..z* e *A..Z*;

special Qualquer símbolo que não seja nem letra nem dígito.

other Qualquer símbolo que não possa ser consumido estando no estado de origem da transição (lembrando que deste estado podem, e geralmente estarão, saindo outras transições). Transições com este símbolo estão geralmente associada a uma função adaptativa de tratamento de erro, através de uma ação adaptativa.

As figuras 2.(a) e 2.(b) mostram um autômato adaptativo para $a^n b^n c^n$ usando a representação gráfica sugerida em [5], ao lado da representação tabular, empregada no AdapTools (figura 2.(c)). A representação tabular deve ser interpretada da seguinte forma: na linha 6, temos a transição que consome uma seqüência de símbolos *a*. A última coluna desta linha indica que a função adaptativa *F* deve ser executada após (.*F*) o consumo de cada símbolo *a*. A

[2] Até a versão 1.4.2 o AdapTools utilizava o prefixo "?p" para denotar parâmetros



	Head	Orig	Inpu	Dest	Push	Outp	Adap
1	?*F*	?*x*	*eps*	?*y*	?*z*	*nop*	*nop*
2	−*F*	?*x*	*eps*	?*y*	?*z*	*nop*	*nop*
3	+*F*	?*x*	*b*	∗*n*1	*nop*	*nop*	*nop*
4	+*F*	∗*n*1	*eps*	∗*n*2	*nop*	*nop*	*nop*
5	+*F*	∗*n*2	*c*	?*y*	?*z*	*nop*	*nop*
6	*S*	0	*a*	0	*nop*	*nop*	*.F*
7	*S*	0	*eps*	1	*fin*	*nop*	*nop*

(*c*)

Figura 1: Autômato adaptativo que reconhece $a^n b^n c^n$
(a) Mecanismo adaptativo (b) Mecanismo subjacente (c) Código objeto da máquina virtual do AdapTools

função adaptativa *F* realiza uma busca por uma transição contendo o símbolo *eps* na coluna de entrada (transição em vazio), na linha 1 (ação elementar de consulta), remove esta transição (linha 2) e insere três novas transições, que consomem cadeias *bc* (linhas 3 e 5), além de manter uma transição vazia no autômato, estrategicamente posiciona para garantir o reconhecimento correto da linguagem (linha 4).

2.1 Diferenças em Relação a Definição Original

A operação do autômato adaptativo no AdapTools segue a proposta descrita no capítulo [5], no que tange à camada adaptativa. A camada subjacente, o autômato de pilha estruturado, funciona de maneira análoga a da definição apresentada em [2], com uma exceção: na versão atual, a devolução de símbolos para a cadeia de entrada só é possível através de rotinas semânticas. Reproduzimos na figura 2 um exemplo de autômato adaptativo, o simulador de pilha, apresentado em [2], fornecendo em seguida, na figura 3 o autômato equivalente no AdapTools.

Produções Iniciais

```
( 1 , "B" ) : → 4
( 2 , "B" ) : → 3
( 3 , ")" ) : → 4
( 1 , "(" ) : → 2 , 𝒜(2,3,1)
```

Função Adaptativa

```
𝒜(i,j,n) = { k* , m* :
           + [ ( k , "B" ) : → m ]
           + [ ( m , ")" ) : → j ]
           + [ ( i , "(" ) : → k , 𝒜(k,m,i) ]
           + [ ( n , "(" ) : → i , 𝒜(i,j,n) ]
           + [ ( n , "(" ) : → i ]
```

Figura 2: Simulador de pilhas na versão original

2.2 Formato do Arquivo Gerado pelo AdapTools

O formato de um arquivo contendo o "código objeto" de um autômato adaptativo é basicamente um *dump* da tabela mostrada na interface gráfica do AdapTools, com o símbolo de ponto-e-vírgula sendo usado como separador de colunas. A primeira linha do arquivo deve conter a cadeia "[Version] 2". Este recurso foi utilizado para permitir alterações no formato do arquivo em futuras versões, mantendo-se a compatibilidade com os arquivos anteriores. O AdapTools

	Head	Orig	Inpu	Dest	Push	Outp	Adap
1	S	1	B	4	fin	nop	nop
2	S	2	B	3	nop	nop	nop
3	S	3	)	4	nop	nop	nop
4	S	1	(	2	nop	nop	.A(2,3,1)
5	+A	*k	B	*m	nop	nop	nop
6	+A	*m	)	$\%_2$	nop	nop	nop
7	+A	$\%_1$	(	*k	nop	nop	.A(*k,*m,$\%_1$)
8	-A	$\%_3$	(	$\%_1$	nop	nop	.A($\%_1$,$\%_2$,$\%_3$)
9	+A	$\%_3$	(	$\%_1$	nop	nop	nop

Figura 3: Simulador de pilha no AdapTools

aceita também arquivos sem esta primeira linha, e sem as colunas de cabeçalho e chamadas de funções adaptativas, neste caso, o arquivo corresponde apenas a um autômato estruturado (sem a camada adaptativa).

3 Interface do Sistema

A figura 4 apresenta a tela principal do aplicativo, acrescida de marcadores (letras em azul) para os principais componentes de interface, que serão descritos a seguir. O principal componente é a tabela (figura 4.(a)) contendo o código do autômato adaptativo. As tarjas vermelha e lilás marcam, respectivamente, a transição e a ação adaptativa elementar (se for o caso) sendo executadas a cada momento.

As três caixas de textos à direita da janela estão associadas a cadeia de entrada (figura 4.(d)), a cadeia de saída (figura 4.(e)), para os casos em que o autômato funciona como um transdutor; e a uma saída auxiliar de texto (figura 4.(f)). Esta saída auxiliar de texto é geralmente manipulada por rotinas semânticas, associadas as regras de transição do autômato, mas especificadas utilizando a linguagem de programação em que o AdapTools foi desenvolvido (ver seção 4). Uma cor diferente para o fundo da caixa de texto de entrada é utilizada para marcar a parte de cadeia já lida pelo autômato.

Os botões, mostrados nas figuras 4.(j) e 4.(k), são utilizados para iniciar, dar continuidade a execução (no modo passo a passo) e reiniciar a máquina virtual. A velocidade de execução pode ser controlada por uma barra de rolagem (figura 4.(c)), que quando arrastada até seu ponto mais inferior, coloca a máquina em execução passo a passo. A pilha de chamadas de sub-máquinas, os estados finais e o estado atual são apresentados através dos componentes (b), (g) e (h), respectivamente. Finalmente, o componente (i) é utilizado para mostrar o conteúdo de variáveis e geradores instanciados por ações elementares de consulta. Durante a execução de ações elementares de consulta, uma terceira tarja colorida é utilizada para mostrar as transições que satisfazem as restrições impostas.

A barra de menus agrega as operações usuais para manipulação de arquivos, que neste caso, incluem arquivos que armazenam a especificação de autômatos (menu *Machine*), cadeias de entrada (menu *Input*) e saída semântica (menu *Output*). O AdapTools oferece algumas opções para agregar diferentes arquivos em um único projeto, facilitando assim a sua manipulação. Arquivos de projetos possuem extensão "prj" e podem ser criados e recuperados através das opções do menu de projeto (menu *Project*). A utilização de projetos facilita sobremaneira o trabalho com sistemas que incluem múltiplas máquinas (ver próxima seção) e permitem ainda que um arquivo texto, de ajuda, seja associado ao projeto. Este arquivo de ajuda é automaticamente carregado e disponibilizado ao usuário através do menu de ajuda (menu *help* - na extremidade direita da barra de menus). O menu de ajuda oferece acesso também a um hipertexto contendo um tutorial sobre o AdapTools.

3.1 Visualização Gráfica

Usando o menu de opções (*options*) ou o clique invertido do mouse sobre alguma das transições mostradas na tabela com o código objeto, é possível ativar o módulo de visualização gráfica da AdapTools. Este módulo, construído a partir do software livre OpenJGraph [3], mostra graficamente o mecanismo subjacente do autômato adaptativo, animando as modificações sofridas durante sua execução. Um algoritmo de desenho automático de grafos, implementado no OpenJGraph, busca posicionar os vértices e arestas que representam graficamente o autômato de forma que a

[3]Disponí vel gratuitamente em http://openjgraph.sourceforge.net/

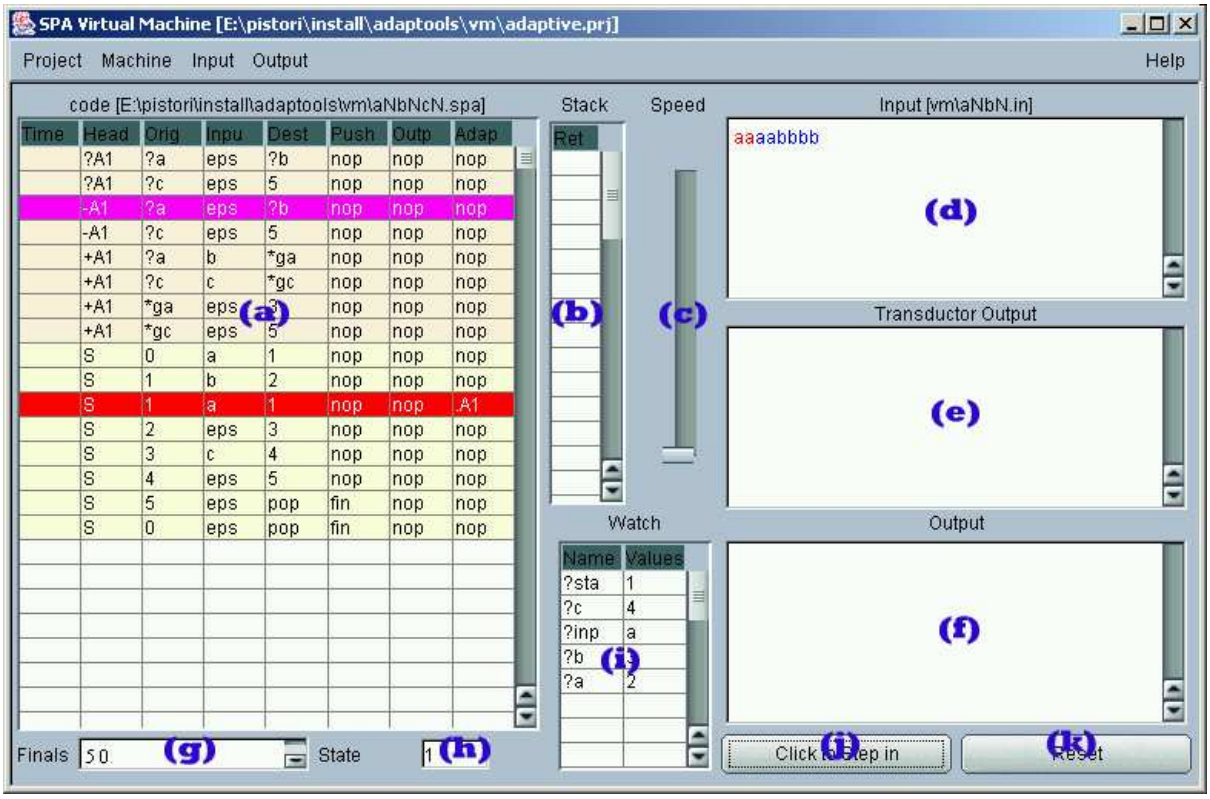


Figura 4: Janela principal do AdapTools

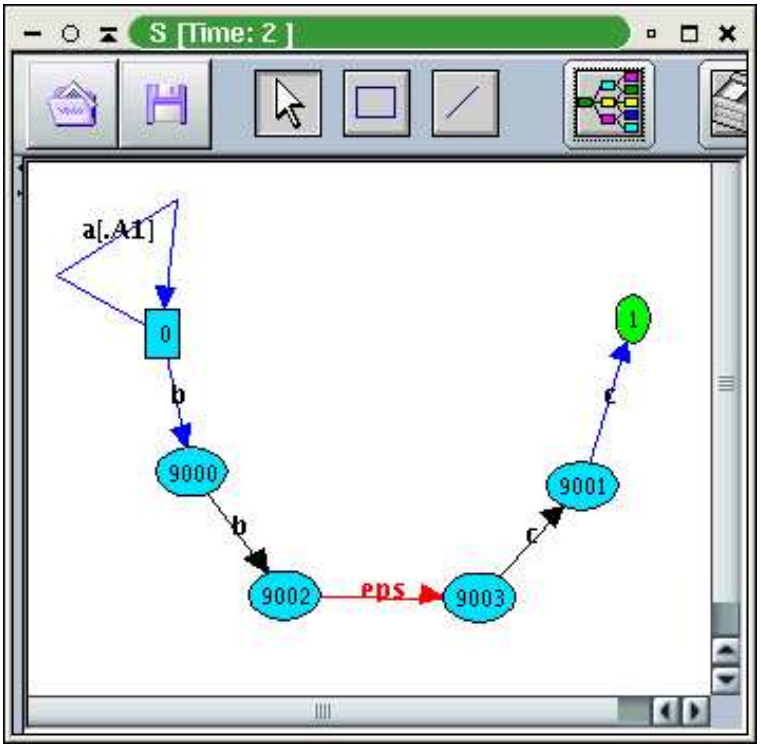


Figura 5: Módulo de animação gráfica do AdapTools

visualização seja facilitada. Este algoritmo é inspirado em leis da física relacionadas com atração e repulsão entre corpos. Os vértices, neste algoritmo, modelam corpos que se repelem, ligados entre si por cordas elásticas (arestas), que contra-balanceiam a força de repulsão. O objetivo final do algoritmo é encontrar um ponto de equilíbrio para o conjunto de vértices (corpos), respeitando valores pré-definidos para as variáveis envolvidas no modelo (força de atração, elasticidades,etc). O módulo de animação permite ainda a interação do usuário, que pode movimentar os vértices, através do mouse, buscando uma melhor visualização. A figura 5 mostra a janela de animação gráfica do AdapTools. Diferentes cores são utilizadas para destacar transições recém inseridas (preto), transições removidas (vermelho) e estados finais (verde).

4 Tratamento de Rotinas Semânticas

Rotinas semânticas podem ser introduzidas em um autômato adaptativo através da coluna de símbolo de saída (*outp*). Ao executar o autômato, a máquina virtual, antes de enviar o símbolo de saída para a cadeia de saída, faz uma chamada para um procedimento interno, pré-definido, chamado *execute*. Este procedimento deve analisar o símbolo de saída e retornar para a máquina virtual uma cadeia qualquer de símbolos (que pode ser vazia), para ser impressa na saída auxiliar de texto do AdapTools. O procedimento *execute* pode ser substituído por outro, através da manipulação de códigos fontes em Java e a recompilação do sistema. Para auxiliar o processo de escrita de rotinas semânticas, o pacote AdapTools oferece uma classe chamada *Adaptools.vm.Semantics*, que pode ser especializada na construção de novos módulos para tratamento de rotinas semânticas. Estes módulos podem, internamente, utilizar todos os recursos oferecidos pelo Java (um dos exemplos do AdapTools usa uma saída sonora, ao invés de textual). O AdapTools implementa algumas rotinas semânticas padrão, que podem ser utilizadas sem necessidade de recompilação de código.

O programa abaixo define as rotinas semânticas que são automaticamente carregadas pelo AdapTools. Para utilizar estas rotinas basta inserir o nome das mesmas no campo de símbolo de saída do autômato (coluna *outp*). A adição de novas rotinas semânticas é obtida pela inclusão de novos métodos na classe *adaptools.vm.DefaultSemantics* (e recompilação do fontes do AdapTools). Para cada nova rotina, é preciso também adicionar um novo teste ao método *execute*, da classe *adaptools.vm.DefaultSemantics*. Este teste faz a ligação entre o conteúdo da coluna *outp*, passado como parâmetro (*name*) para o método *execute*, e o método que implementa a rotina semântica.

```
package adaptools.vm;

import java.util.*;
import javax.swing.*;

public class DefaultSemantics extends adaptools.vm.Semantics  {

  StringBuffer value;

  public Vmds vmds;

  public DefaultSemantics(Vmds vmds) {
    this.vmds = vmds;
        value = new StringBuffer();
   }

   public Token execute(String name) {
     if(name.equals("_initBuffer")) return(_initBuffer());
          else if(name.equals("_appendToBuffer")) return(_appendToBuffer());
          else if(name.equals("_metaSymbol")) return(_metaSymbol());
          else if(name.equals("_bufferToken")) return(_bufferToken());
          else if(name.startsWith("_token")) return(_token(name));
          else return(null);
   }

   private Token _initBuffer() {
```

```
        value = new StringBuffer();
        return null;
    }

    private Token _appendToBuffer() {
        value.append(vmds.curToken().value);
        return null;
    }

    private Token _metaSymbol() {
        return new Token(vmds.curToken().value,vmds.curToken().value);
    }

    private Token _bufferToken() {
        String temp = value.toString();
        _initBuffer();
        return new Token(temp,temp);
    }

    private Token _token(String temp) {
        temp = temp.substring(temp.indexOf('(')+1,temp.indexOf(')'));
        return new Token(temp,temp);
    }
}
```

A estrutura *Token* que é retornada por todas as rotinas semânticas é definida em *Token.java*, da seguinte forma:

```
package adaptools.vm;

public class Token extends Object  {

    public String label;
    public String value;

    public Token(String label, String value) {
                this.label = label;
                this.value = value;
    }
}
```

Ou seja, ela é apenas um par de *strings*. Os comandos *vmds.curToken().label* e *vmds.curToken().value* podem ser utilizados dentro de qualquer rotina semântica para se obter o nome e o valor do símbolo corrente na cadeia de entrada do autômato. Quando a entrada do autômato é apenas uma cadeia de caracteres, o AdapTools automaticamente transforma cada caracter *c* em um token cujos campos, *label* e *value*, contêm o mesmo valor (a cadeia "c"). Agora, quando a entrada do autômato está conectada à saida de outro autômato, as rotinas semânticas podem ser utilizadas para criar *tokens* mais sofisticados.

Segue abaixo uma pequena descrição das cinco rotinas semânticas implementadas em *adaptools.vm.DefaultSemantics* (o *underscore* que antecede os nomes das rotinas é apenas um convenção para facilitar a identificação de rotinas semânticas no código de um autômato) :

_initBuffer Cria um espaço temporário para armazenamento sequencial de caracteres.

_appendToBuffer Insere o próximo símbolo da cadeia de entrada no espaço temporário de armazenamento.

_bufferToken Devolve, na forma de um *token*, a cadeia armazenada no espaço temporário. Esta rotina também reinicializa o espaço temporário de armazenamento.

_token(*x*) Recebe como parâmetro uma cadeia de caracteres *x*, e devolve um *token* com nome e valor iguais a *x*.

_metaSymbol Devolve o próximo símbolo da cadeia de entrada como um *token* (campos *label* e *value* são preenchidos com o símbolo da cadeia de entrada).

A classe *adaptools.vm.DefaultSemantics* é apenas uma das possíveis especializações da classe abstrata *adaptools.vm.Semantics*. Outras especializações podem ser criadas caso se deseje separar as rotinas semânticas associadas a diferentes autômatos. Para indicar qual a classe que implementa a semântica de determinado autômato (quando esta classe não é a *adaptools.vm.DefaultSemantics*), é necessário alterar o método *getSemanticObject*, da classe *adaptools.vm.Kernel*, e recompilar o sistema. Por exemplo, para associar as rotinas semânticas implementadas por uma determinada classe *adaptools.contrib.MySemantic*, ao autômato chamado "*MyMachine.spa*", deve-se adicionar ao corpo do método *getSemanticObject* o seguinte teste:

```
if(  vmds.getObjectFileName().indexOf("MyMachine.spa") != -1 )
     return( new adaptools.contrib.MySemantic(vmds) );
```

O código completo do método *getSemanticObject* na versão atual do AdapTools é mostrado abaixo. É importante notar no código abaixo que, para determinar a semântica que deve ser carregada, vários dos testes procuram apenas por uma sub-cadeia dentro do nome da autômato. Com isto, qualquer autômato cujo nome contenha a cadeia *lex*, por exemplo, terá sua semântica associada à do analisador léxico do compilador Wirth-AdapTools (implementada pela classe *adaptools.exemples.ccl.Semantics*).

```
public Semantics getSemanticObject() {
  if(  vmds.getObjectFileName().indexOf("sintatico") != -1 ) {
    System.out.println("CC Semantics Loaded");
    return( new adaptools.examples.cc.Semantics(vmds) );
  }
  else if(vmds.getObjectFileName().indexOf("ccr") != -1 ) {
    System.out.println("CC with Error Recovery Semantics Loaded");
    return( new adaptools.examples.ccr.Semantics(vmds) );
  }
  else if(vmds.getObjectFileName().indexOf("rsw") != -1 ) {
    System.out.println("RSW Semantics Loaded");
    return( new adaptools.examples.rsw.Semantics(vmds) );
  }
  else if(  vmds.getObjectFileName().indexOf("lex") != -1 ) {
    System.out.println("Lexico Semantics Loaded");
    return( new adaptools.examples.ccl.Semantics(vmds) );
  }
  else if(  vmds.getObjectFileName().indexOf("talker") != -1 ) {
    System.out.println("Voice Semantics Loaded");
    return( new adaptools.examples.voice.Semantics(vmds,soundPlayer) );
  }
  else if(  vmds.getObjectFileName().indexOf("extraiEventos") != -1 ) {
    System.out.println("extraiEventos Semantics Loaded");
    return( new adaptools.examples.eventos.Semantics(vmds) );
  }
  else {
    System.out.println("Default Semantics Loaded");
```

```
    return( new adaptools.vm.DefaultSemantics(vmds) );
  }
}
```

5 Comunicação entre Máquinas Virtuais

A saída de uma rotina semântica (figura 4.(f)) de uma máquina *A* pode ser redirecionada para uma outra máquina *B* através da opção de conexão (*Connect*) do menu *input* na máquina *B*. Este recurso pode ser usado, por exemplo, na construção de compiladores, quando a análise léxica e sintática são executadas por módulos distintos. O tipo básico de dados transferidos de uma máquina para outra é um *token*, formado por duas cadeias de caracteres, representando seu nome (*token.label*) e seu conteúdo (*token.value*). Os *tokens* devem ser gerados e lidos através de rotinas semânticas, que como vimos na seção anterior, requerem um trabalho adicional por parte do usuário. Para facilitar um pouco esta tarefa o AdapTools já inclui algumas rotinas semânticas que facilitam o trabalho com *tokens* (sem necessidade de se entrar nos fontes do sistema).

6 Integração com Outros Sistemas

A máquina virtual do AdapTools pode ser executada sem os recursos gráficos apresentados nas seções anteriores, facilitando assim a integração com outros sistemas externos. Nos casos em que o sistema externo esteja sendo desenvolvido em Java, esta integração é trivial, e consiste basicamente em incluir e utilizar a classe *Runner* do pacote AdapTools no sistema em questão. Para sistemas desenvolvidos em outras linguagens não existe ainda um mecanismo para facilitar esta integração, no entanto, é possível realiza-la, ainda que de forma rudimentar, utilizando a versão do AdapTools que pode ser executada através da linha de comando do sistema operacional. Esta versão pode ser integrada à outros sistemas utilizando-se os recursos do próprio sistema operacional para comunicação entre processos (e.g. *pipes* no linux).

7 Arquitetura do Sistema

Destacaremos agora alguns elementos do projeto do AdapTools que poderão ser úteis a futuros desenvolvedores ou àqueles que tenham a intenção de reutilizar total ou parcialmente os códigos fontes do AdapTools em outros projetos. Uma característica fundamental deste projeto é a separação entre: (1) a máquina virtual que implementa o mecanismo subjacente (*Kernel.java*), (2) o mecanismo adaptativo (*Adapter.java*) e (3) os tipos de dados abstratos de apoio à execução da máquina virtual, como por exemplo, a tabela que armazena o autômato, a pilha de chamada de submáquinas e o estado corrente, entre outros (*Vmds.java*[4]).

O módulo *Vmds.java* é, basicamente, uma interface (no sentido utilizado em programação em java: *keyword interface*), que deve ser implementada para oferecer as estruturas de dados requeridas pelos módulos *Adapter.java* e *Kernel.java*. Duas implementações são disponibilizadas no pacote. A primeira (*Viewer.java*) utiliza os próprios componentes visuais do pacote Swing e implementa a interface gráfica com o usuário . A segunda (*VmdlImpl.java*) oferece as estruturas de dados utilizadas pelo módulo textual (linha de comando) do AdapTools (*Runner.java*).

8 Considerações sobre a Máquina Virtual do AdapTools

O formato tabular utilizado na especificação de um autômato adaptativo não foi, a principio, criado para ser utilizado por um usuário "comum" de tecnologia adaptativa, mas para ser gerado automaticamente por um módulo que permitisse a especificação de autômatos em uma linguagem mais acessível (possivelmente gráfica). No entanto, como na versão atual o formato tabular é o único instrumento para inserção e manipulação de especificações de autômatos do AdapTools, acabamos por permitir algumas simplificações que descaracterizaram um pouco a máquina virtual. A principal delas sendo a sintaxe da coluna de ação adaptativa. Devido a esta coluna, a máquina virtual do AdapTools acaba tendo que implementar internamente mecanismo rústicos de análise léxica e semântica, para interpretar seu conteúdo. Em versões posteriores, quando tivermos um mecanismo de mais alto nível para a especificação dos autômatos, esta máquina virtual deverá ser reprojetada.

[4] *Virtual Machine Data Structures*

Além da questão acima citada, o projeto atual da máquina virtual não considerou requisitos de desempenho. Uma importante meta para as próximas versões do AdapTools é a construção de uma máquina virtual otimizada, possivelmente com implementações em linguagem de máquina (e não apenas em byte-codes Java) e considerando as questões de desempenho na execução de funções adaptativas, em autômatos grandes, levantadas em [5].

A possibilidade de se realizar a escrita de símbolos na cadeia de entrada, um mecanismo bastante útil na especificação de determinados problemas no projeto de compiladores [2], também é uma questão que deve ser abordada nas próximas implementações do AdapTools. A adição deste mecanismo deverá ser precedida, provavelmente, de uma modificação na estrutura tabular dos autômatos adaptativos. Uma primeira sugestão seria a adição de uma nova coluna para especificar o destino de cada símbolo da saída. Especial cuidado deverá ser tomado em relação à comunicação inter-máquinas, pois a entrada de uma máquina pode estar conectada a uma segunda máquina. Esta segunda máquina é executada de forma simultânea à primeira (cada uma em seu *thread*), e portanto, tratamentos de concorrência entre processos mais sofisticados deverão ser implementados. Uma outra sugestão para trabalho futuro seria justamente permitir a execução distribuída e concorrente da máquina virtual do AdapTools.

Também seria muito interessante oferecer ao usuário final uma maneira mais simples de trabalhar com rotinas semânticas, que não exijam recompilação de códigos fontes. A primeira alternativa a ser investigada poderia ser a criação de um mecanismo que permita a inserção de código Java diretamente na especificação do autômato, ou eventualmente em uma gramática que possa ser convertida para autômato adaptativo (algo parecido com o *yacc* para a linguagem C). A segunda envolve a implementação de conjuntos mínimos, mas expressivos, de rotinas semânticas básicas (inserção em pilha, remoção de pilha, escrita na saída padrão, etc) que possam ser utilizadas na solução de problemas em um determinado domínio (e.g. construção de compiladores). A versão atual já apresenta algumas rotinas semânticas básicas, no entanto, elas foram projetadas para resolver problemas bastante específicos, como a implementação de um compilador Wirth-AdapTools [1]. Um estudo mais profundo sobre o conjunto de rotinas semânticas "ideal", para determinados domínios de aplicação, deverá ser realizado futuramente.

Por último, novos módulos deverão ser acrescentados ao AdapTools para permitir a manipulação de outros tipos de dispositivos adaptativos. Esta generalização poderia começar com a criação de um formato tabular para a especificação de dispositivos adaptativos apresentada em [4], em que as colunas referentes ao mecanismo subjacente pudessem ser definidas dinamicamente, em função do tipo de dispositivo a ser implementado. Uma outra alternativa seria a criação de compiladores capazes de converter um dispositivo adaptativo específico em um autômato adaptativo que pudesse ser executado pelo AdapTools.

9 Conclusões

Apresentamos neste trabalho uma ferramenta para a área da tecnologia adaptativa que pode ser utilizada, pelo menos de quatro maneiras diferentes: (1) no projeto e desenvolvimento de soluções baseadas em autômatos adaptativos, (2) como uma ferramenta educacional para ensino da tecnologia adaptativa, (3) na construção de ambientes de apoio ao aprendizado em áreas como construção de compiladores e teoria dos autômatos (usando apenas o mecanismo subjacente do autômato adaptativo) e (4) como o passo inicial na implementação de outros dispositivos adaptativos, como a árvore de decisão adaptativa apresentada em [5].

Como continuidade deste trabalho visualizamos pelo menos quatro objetivos importantes: (1) otimização da máquina virtual com implementações em linguagens de máquina para diferentes plataformas, (2) aprimoramento da execução simultânea e comunicação entre diferentes autômatos adaptativos, com implementação distribuída e generalizando os mecanismos de leitura e escrita de símbolos (e.g. permissão para escrita na entrada), (3) criação de maneiras mais amigáveis para tratamento semântico e (4) generalização do AdapTools para permitir a utilização de outros dispositivos adaptativos, além dos autômatos.

O pacote AdapTools pode ser obtidos no *site* do laboratório de linguagens e técnicas adaptativas da Universidade de São Paulo (http://www.pcs.usp.br/ lta) ou diretamente no endereço http://www.ucdbnet.com.br/adaptools. A página do AdapTools oferece ainda diversos exemplos de autômatos, tutoriais, listas de pendências, documentação do sistema e indicações de como participar no desenvolvimento deste pacote, que é distribuído como software livre.

Referências

[1] J. J. Neto. Compilador de gramáticas descritas em notação de wirth modificada. Technical report, PCS-POLI-USP, São Paulo, Brasil, 1988.

[2] J. J. Neto. Contribuição à metodologia de construção de compiladores. *Post-Doctoral Tese de Livre Docência, Escola Politécnica, Universidade de São Paulo*, 1993.

[3] J. J. Neto. Adaptive automata for context-sensitive languages. *SIGPLAN NOTICES*, 29(9):115–124, September 1994.

[4] J. J. Neto. Adaptative rule-driven devices - general formulation anda case study. In *CIAA'2001 Sixth International Conference on Implementation and Application of Automata*, pages 234–250, Pretoria, South Africa, July 2001.

[5] Hemerson Pistori. *Tecnologia Adaptativa em Engenhara de Computação: Estado da Arte e Aplicações*. PhD thesis, Universidade de São Paulo, São Paulo, Brasil, 2003.