A Free Software for the Development of Adaptive Automata

Hemerson Pistori[1 2] **, João José Neto**[2]

[1]Depto. Engenharia de Computação – Universidade Católica Dom Bosco
79117-900 Campo Grande, MS, Brasil

[2]Laboratório de Linguagens e Tecnologias Adaptativas
Escola Politécnica – Universidade de São Paulo
05508-900 São Paulo, SP, Brasil

pistori@ec.ucdb.br, joao.jose@poli.usp.br

Abstract. *Although recent, adaptive technology is already being used to solve complex problems in areas such as compiler construction, natural language processing, computational vision and robotics. This papers introduces AdapTools, a free-software environment that helps the development of solutions based on adaptive automata. Since adaptive automata generalize finite state and structured pushdown automata, AdapTools may also be used as a tool for the development of traditional automata, as well as an educational tool.*

1. Introduction

The former traces of adaptive devices, which achieved its most general formalization in [Neto, 2001], appeared in the early work of van Wijngaarden [van Wijngaarden, 1974]. Although van Wijngaarden's two-level grammars cannot be considered adaptive devices, in the sense that they do not change its own structure at execution time, all of them increase the representative power of formal devices, like context-free grammars, without loosing much of its "ease of use".

The last thirty years witnessed the upcoming of several types of adaptive devices, all of them adding some self-modifying or dynamic characteristic to a specific formalism, in order to overcome its limited expressive power. Among them are: modifiable grammars [Burshteyn, 1992]; self-modifying finite automata [Rubinstein and Shutt, 1994] and dynamic grammars [Boullier, 1994]. An attempt to generalize the above concepts, capturing the meaning of adaptiveness and divorcing it from the concept of a subjacent device (e.g. context free grammars, finite-state automata) gave rise to the formalism called rule-driven adaptive device. This work paved the way for the construction of adaptive devices based on subjacent mechanism as diverse as Markov chains [Basseto and Neto, 1999] and decision trees [Pistori et al., 2003]. Examples of applications of adaptive devices may be found in a variety of areas, like automatic music composition [Basseto and Neto, 1999], natural language processing, robotic navigation [Junior et al., 2000], pattern recognition [Costa et al., 2002] and computational vision [Pistori et al., 2003].

Though most of the adaptive theory is now mature, it still lacks some widely accepted related software tool. Most works on implementation of adaptive devices developed so far are deeply attached to specific projects or to specific subjacent mechanisms, and may not be easily adapted for use in other situations. The present work is a significant attempt in this direction, represented by the software named AdapTools, described below.

Adaptools is still far from implementing the full generality of the adaptive device theory. Nonetheless, its design foresees many of such extensions. Currently, AdapTools implements a virtual machine that executes a slightly modified version of the originally described adaptive automata [Neto, 1993, Neto, 2000] enhanced with an integrated development environment containing some graphical visualization, debugging, project maintenance and editing resources.

Adaptools is distributed under GNU's Public License (GPL), being available at the laboratory of adaptive languages and technologies' web site [1].

Besides having been designed as a development environment for adaptive devices, Adaptools may also be used as an excellent educational tool for disciplines like computer theory and compiler construction. Since the traditional finite state automata, non-deterministic ϵ-automata and structured pushdown automata are all specializations of adaptive automata, AdapTools may be used as a virtual laboratory where the student may practice experimentally the concepts. Several illustrating examples have been included in the package. One of the most important is a compiler-compiler that converts a grammar specification, in Wirth notation, of a language L, and produces an adaptive automaton that recognizes L (this automaton may be simulated by AdapTools); an extremely simple and powerful prototype of a text-to-speech system and an error-recovery scheme for handling simple syntactical errors are also available. Adaptools is already being used as an extra-class educational reinforcement tool in some disciplines of the computer engineering program offered in our institution.

The following section presents a brief description and an example of adaptive automata, as defined in Adaptools. Next, in section 3, an overview of AdapTools features is shown. The last section is reserved for conclusions and future work.

2. Adaptive Automata - Adaptools Version

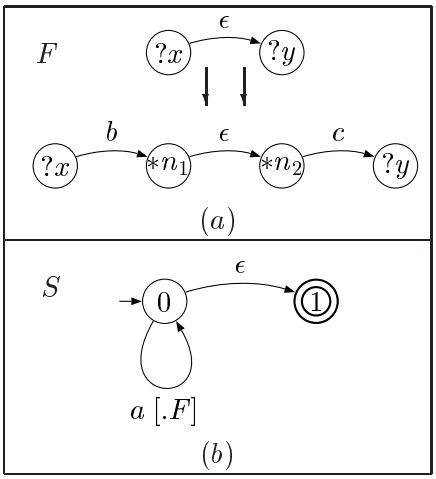
Conceptually, an structured pushdown automaton may be regarded as a set of finite-state automata (called sub-machines) that allows two additional kinds of transitions: sub-machine calls and returns, specifying changes in the flow of execution from one sub-machine to another, with a pushdown stack for holding return addresses. An adaptive automaton is an structured pushdown automaton that allows dynamic modifications in its set of transitions. Such changes are accomplished by performing the so-called query, deletion and insertion elementary actions, grouped in adaptive functions, that are called from and may operate on any transition, before or after its execution. See [Neto, 1993] for more details.

In Adaptools, the form chosen to represent adaptive automata is a table with seven columns and four basic kinds of lines: normal transition, sub-machine calls and returns, and query, deletion an insertion elementary adaptive actions. A *header* column holds names of sub-machines and adaptive functions. In lines representing elementary adaptive actions, the header also informs the type of action: $?$,$-$ or $+$, for query, deletion or insertion, respectively. The columns for *origin and destination states* and *input and output symbols* have the usual interpretation, except that, for adaptive functions, they may refer to variables (in order to express patterns), and generators, representing states that must be created dynamically. Variables and generators are denoted with prefixes $?$ and $*$, respectively. The $push$ column is used in sub-machine calls, to indicate the return state. Sub-machine returns are indicated by the reserved keyword pop placed in the *destination state* column. The last column, for *adaptive action*, is used to link a transition to a pair of adaptive functions and contains a string with the form $B.A$, where B and A are names of adaptive functions that are to be executed Before and After the transition (any one of them may be omitted). The reserved keyword nop may be used anywhere to indicate that the column is not being used. Empty transitions hold the string eps instead of an input symbol.

In order to avoid extra structures, the initial and final states of an automaton are encoded in a somewhat cumbersome way: The $push$ column must have the keyword fin when the *destination state* is a final state, and, by convention, the *origin state* of the first line of each sub-machine always denotes its initial state.

Figure 1 shows an adaptive automaton that recognizes the context-dependent language $a^n b^n c^n$, using both Adaptools table representation, figure 1.(c), and a graphical representation for its only adaptive function, figure 1.(a), and sub-machine, figure 1.(b). The device works as follows: each time some symbol a is consumed (line 6) the adaptive function F finds the only empty

[1] http://www.pcs.usp.br/~lta - download section



	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	$*n1$	nop	nop	nop
4	$+F$	$*n1$	eps	$*n2$	nop	nop	nop
5	$+F$	$*n2$	c	$?y$	$?z$	nop	nop
6	S	0	a	0	nop	nop	$.F$
7	S	0	eps	1	fin	nop	nop

(c)

Figure 1: Adaptive Automaton that Recognizes $a^n b^n c^n$
(a) Adaptive Mechanism (b) Subjacent Mechanism (c) Adaptools Object Code

transition of the automata (line 1), removes it (line 2) and insert two new transitions consuming the string bc (lines 3 and 5) with and empty transition strategically put between them (line 4).

In the original definition of adaptive automata [Neto, 1993], variables and generators are explicitly declared and adaptive function calls are allowed to appear inside adaptive functions. Avoiding such features in AdapTools originates a simpler tabular representation of the machine. Implicit declaration of variables and generators certainly do not affect the expressiveness of the device, but an study of the impact of not permitting adaptive function outside the subjacent layer is yet to be conducted. However, the broad range of cases already implemented in AdapTools, including regular, context-free and context-dependent languages, suggests that this modification preserved much of the power of the original device, which is Turing-powerful.

3. Adaptools

Adaptools is a software environment where adaptive automata can be implemented, tested and experimented. The core of this software is a virtual machine that executes adaptive automata represented in tabular form, such as the one depicted in figure 1.(c). The execution of transitions and elementary adaptive actions, as well as the contents of stacks, variables and generators, may be traced, step by step. A powerful graph drawing tool, based on the ForceDirect algorithm featured by OpenJGraph package, may be used for the visualization of modifications accomplished by the automata. The algorithm models nodes and edges as electrical forces and logarithmic springs, respectively, and uses the physical laws of repulsion and tension to automatically find a balanced layout for the graph representing the automaton. By allowing human intervention, pretty good animations are obtained, that really enhance the visualization and comprehension of the devices being simulated. Another important feature is the simultaneous execution of multiple devices, by using Java multi-threading resources. An example of this feature, available in the package, is a parser and a lexical analyzer, that work simultaneously to produce object code for Adaptools' virtual machine. An enhanced version of this example, which features syntactical error-recovering capabilities in the generated code is another powerful tool easily generated within AdapTools, used in compiler construction courses.

4. Conclusions

In this work we introduced a free-software which can be used in, at least, three ways: (1) in the development of solutions using the powerful emerging technology based on adaptive automata, (2) as an educational tool for teaching and training in adaptive automata[2], and weaker traditional

[2]A somewhat related case study can be found in [Leonardi, 1999]

formalisms incorporated by them, like finite-state automata and structured pushdown automata and (3) as the basis for the development of a more general tool that will handle adaptive devices with subjacent mechanisms other than automata. The development of a generalized version of AdapTools that works with adaptive decision trees is already in course.

References

Basseto, B. A. and Neto, J. J. (1999). A stochastic musical composer based on adaptative algorithms. In *Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação. SBC-99.*, volume 3, pages 105–130, PUC-RIO, Rio de Janeiro, Brazil.

Boullier, P. (1994). Dynamic grammars and semantic analysis. Technical Report 2322, INRIA.

Burshteyn, B. (1992). USSA - universal syntax and semantics analyser. *ACM SIGPLAN Notices*, 27(1):42–60.

Costa, E. R., Hirakawa, A. R., and Neto, J. J. (2002). An adaptive alternative for syntactic pattern recognition. In *Proceeding of 3rd International Symposium on Robotics and Automation, ISRA*, pages 409–413, Toluca, Mexico.

Junior, J. R. A., Neto, J. J., and Hirakawa, A. R. (2000). Adaptive automata for independent autonomous nagivation in unknown enviroment. In *Proceedings of IASTED International Conference on Applied Simulation and Modelling - ASM 2000*, Banff, Alberta.

Neto, J. J., Pariente, C. B. and Leonardi, F. (1999). Compiler construction - a pedagogical approach. In *Proceedings of the V International Congress on Informatic Engineering - ICIE 99*, Buenos Aires, Argentina.

Neto, J. J. (1993). Contribuição à metodologia de construção de compiladores. *Tese de Livre Docência, Escola Politécnica, Universidade de São Paulo*. [In Portuguese]

Neto, J. J. (2000). Solving complex problems efficiently with adaptative automata. In *Conference on the Implementation and Application of Automata - CIAA 2000*, Ontario, Canada.

Neto, J. J. (2001). 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.

Pistori, H., Neto, J. J., and Costa, E. R. (2003). Utilização de tecnologia adaptativa na detecção da direção do olhar. In *Anais da Conferencia Internacional de la Sociedad Peruana de la Computacion SPC'2003*, Lima, Peru. [In Portuguese]

Rubinstein, R. and Shutt, J. N. (1994). Self-modifying finite automata. In *Proceeding of the 13th IFIP World Computer Congress*, pages 493–498, Amsterdam: North-Holland.

van Wijngaarden, A. (1974). Revised report on the algorithmic language algol 68. *Acta Informatica*, 5:1–236.