Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 109C (2017) 1176–1181

Procedia Computer Science

www.elsevier.com/locate/procedia

International Workshop on Adaptive Technology (WAT 2017)

DInAton: A Didactic and Interative Language for Learning Adaptive Automata by Construction

Italo S. Vega[a], João José Neto[b], Francisco S. Marcondes[c]

[a]*Departamento de Ciência da Computação, Pontifícia Universidade Católica de São Paulo R. Marquês de Paranaguá, 111, São Paulo, SP, Brasil*

[b]*Escola Politécnica, Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo Av. Prof. Luciano Gualberto, s/n, Travessa 3, 158, São Paulo, SP, Brasil*

[c]*Brazilian Institute of Education, Science and Technology (IFSP), R. Pedro Vicente, 625, São Paulo, Brazil*

Abstract

In order to reach the community interested in some basic elements of the adaptive automata theory, an interpretable language called *DInAton* (Didactical and Interactive Adaptive Automata Construction Language) was projected. In the language design, the criteria of similarity, orthogonality and parsimony were considered with the intention of preserving the adherence to the adaptive theory and of conceiving an interactive environment for learning its main foundations. As a research result, we obtained a grammar for the *DInAton* language that served as the basis for the development of a functional prototype with support for incremental construction of finite and adaptive automata.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: adaptive automata, grammar design, interpreter, interactive learning environment, DInAton

1. Introduction

The adaptive automata theory was introduced by José Neto [1] as contribution to the methodology used in the development of programming language compilers. One of the main points of such theory refers to the automaton adaptivity model, expressed as three basic constructed: actions, functions and transitions eligible as adaptive. The language recognition occurs through topological transformations of an underlying device, triggered by an operation of an adaptive mechanism.

Inspired by this theory, Cereda and José Neto [1] developed a library to support programming experiments based on adaptivity. The primary outcome of such effort was a library named AA4J. This library was designed towards adherence to the original formalism seen in José Neto's theory. Hereof, the implementation support for adaptive

* Corresponding author, +55 11 3091-5402.
E-mail addresses: italo@pucsp.br (Italo S. Vega)., jjneto@usp.br (João José Neto)., yehaain@gmail.com (Francisco S. Marcondes).

Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.05.393

mechanisms can be organized in three levels, according to automata categories. The first one refers to the finite automata support, representing sets of states, symbols and transitions (henceforth named finite transitions). Regarding the set of states, there is a distinction in terms of nature and role of each of them. States, in themselves, have a nature of their own, though they can be in the role of *initial* or *accepting* states. The second category supports structured pushdown automata, according to José Neto[1,2]. *Submachine call transitions* are added to the existing expressive features of a finite automaton. Additionally, each call transition (**call**) has a linked callback instruction (**return**). The effect of a call transition involes the use of a pushdown structure to handle calls and returns. The third category supports the representation of adaptive automata, in which such devices add features for expressing adaptive actions, functions and transitions.

However, in an earlier work by José Neto, an additional concern is identified: the elaboration of a didactic material regarding language processors[2]. Vega points out that the use of a didactic material must be in harmony with the learning environment and suggests a focus centered on narrative structures known as *Goal-Obstacle-Catastrophe-Reaction-Dilemma-Decision* (OCC-RDD)[3]. Such narratives are designed for an environment called *Hybrid and Interactive Presential Learning Environment* (HIPLE). The combination of these ideas with a style of interaction based on interpretation emerges as the basis for building a learning environment contemplating the adaptive automata theory.

The core of a mechanical process of interpretation is the language that defines sentential forms of interaction. Considerations about the design of artificial languages intersect Chomsky's initial paths[4], with further developments by Watt and Findlay[5], as well as Friedman, Wand and Haynes[6] And Kaplan[7], for example. However, Bentley's ideas[8] had a marked influence on the results achieved and reported in this paper. The research question addressed here may be summarized as "how to design the *DInAton* language — *Didactical and Interactive Adaptive Automata Construction Language*?" It is a language specially designed for incremental construction of adaptive automata, suitable for HIPLE environments.

2. DInAton Language Design

Several aspects were considered when designing the *DInAton* language, emphasizing its interpretive characteristics, proximity to typical programming languages, mechanisms for textual and graphical representation of under construction automata, as well as its adherence to the semantic model proposed by José Neto in[1]. In a more general context, the criteria proposed by Bentley[8] guided the *DInAton* language design in a greater or smaller scale: orthogonality, generality, parsimony, completeness, similarity, extensibility and openness. These criteria refer to both language instructions and aspects of their implementations.

2.1. DInAton Language Features

Adherence to the theoretical formalism — In line with Bentley, similarity refers to the relationship between the constructs of the language and the elements of the domain to be manipulated. In this project, the elements of the domain are those established by the *AA4J* library. As a means to enable the use of adaptive technology in programming, Cereda and José Neto[9] developed a library for implementing adaptive automata, using the Java programming language. The initiative is based on the original theory introduced by José Neto[1] and aims at being fully adherent with the proposed formalism. From a practical viewpoint, the adaptive automaton implementation may be used to represent types 2 and 3 recognizers as well, as the formalism is proven to be Turing equivalent[10].

As to accomplish semantic equivalence to the adaptive theory, this library will be used to construct language recognizers and query strings. Additionally, the DInAton environment will ensure the construction of valid automata, such that the corresponding implementation will reflect consistently in the library domain. The environment also brings refinements to the library, as new features arise in order to enhance the user experience while interacting with the environment.

Underlying automaton specification — Another criterion of Bentley considered in the design of the *DInAton* language was orthogonality: each instruction has a purpose different from the other. The semantic object refers to the one implemented by Cereda and José Neto[9]. From a syntactic viewpoint, the sentential adherence to the theoretical proposal must be attenuated when considering the constraints imposed by the execution model. In the structure of a

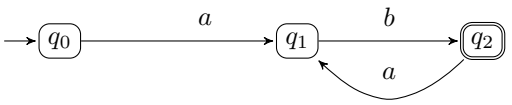

Fig. 1. Automaton graphical representation generated by the DInAton interpreter

deterministic finite automaton, four elements are specified: $AFD = (Q, \Sigma, \delta, q_0, F)$. The *DInAton* grammatical rule provides the following forms using the Parr notation[11] (simplified grammar):

```
grammar DInAton;
dinAton: (instruction | error)* EOF ;
instruction : createAutomaton
            | changeMainAutomaton | insert | drop
            | view | accept | load ;

createAutomaton
    : 'create' 'finite'   n=ID #createFiniteAutomaton
    | 'create' 'pushdown' n=ID #createPushdownAutomaton
    | 'create' 'adaptive' n=ID #createAdaptiveAutomaton ;
```

Finite automaton construction support — In addition to the similarity criterion, the instructions were also designed with parsimony, trying to minimize the amount of syntactic elements needed to conduct a single interaction with the interpretation environment. Thus, in the interpretive mode with finite automata construction support, the following sequence of interactions creates a new automaton identified by `f1` and defines its initial and acceptance states (the symbol > marks the interpreter prompt):

```
> create finite f1
Finite automaton f1 created.
> insert into f1 start q0
f1.start = q0
> insert into f1 accept [q2]
f1.accept = [q2]
```

For instance, let there be an automaton $f_{f1} = (\{q_0, q_1, q_2\}, \{a, b\}, \{(q_0, a) \mapsto q_1, (q_1, b) \mapsto q_2, (q_2, a) \mapsto q_1\}, q_0, \{q_2\})$. The corresponding specification in the *DInAton* environment of interpretation is presented as follows:

```
> insert into f1 transitions [t1=q0,a->q1, t2=q1,b->q2, t3=q2,a->q1 ]
f1.transitions [t1=q0,a->q1, t2=q1,b->q2, t3=q2,a->q1 ]
```

At any time, the recognition of a given sequence of symbols can be requested. For example, what would be the reaction of the automaton so far designed, when asked to recognize the string $w = ab$?

```
> accept a b
yes
```

Multiple representation support — The **view** language command produces an automaton representation, both in textual and graphical forms, being the latter a diagram — the result corresponds to that shown in Figure 1:

```
> view diagram
f1.dot generated
```

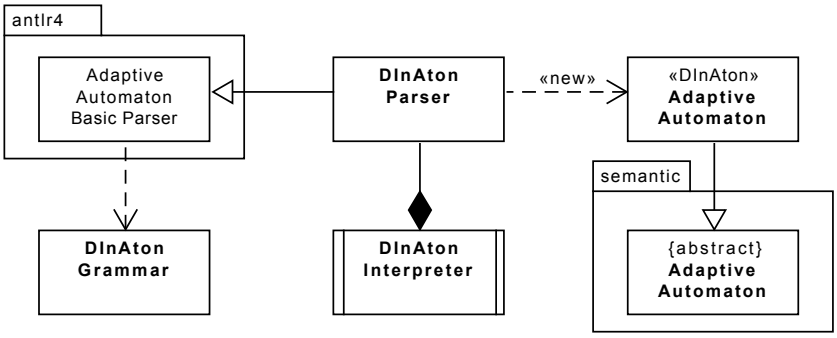

Fig. 2. Interpreter logical architecture

2.2. Grammar Specification and Semantic Object

Although inspired by seminal results of Chomsky[4], José Neto[1] and Ramos[12], the grammar project was boosted by Parr's work[11]. Parr's *ANTLR4* tool provides not only a grammar specification metalanguage, but also integrates comfortably with the object-oriented programming principles. Broadly speaking, each grammar production rule establishes a particular sentential structure and originates a connection point with the semantic model of the *DInAton* interpreter. In Section 2.1 there is an excerpt of the *DInAton* grammar specification, indicated by the `grammar DInAton;` construct. Then, using a BNF notational style, the production rules are introduced, such that, to the left of the `:` separator, a non-terminal is recognized. To the right, the derivation possibilities are listed until the symbol `;` is reached. Let us take the rule that establishes the interpreter instructions format as an example. Several actions are supported, such as automata creation, main automaton replacement inside the interpretation environment, features for the insertion and removal of topology elements of the automaton under construction, external representation generation, test of acceptance of strings of a language and load of an external automaton specification.

Regarding the linkage points in the semantic model, the *ANTLR4* tool offers two possibilities, namely *visitor* and *listener* based approaches. Parr discusses both in detail, presenting their advantages and disadvantages. For *DInAton*, we chose the *listener* approach, originating an architecture in which the structural elements shown in the UML class diagram presented in Figure 2 are highlighted. The main interpretation process is in an active object of class `DInAton Interpreter`. This object is responsible for constructing and interacting with a `DInAton Parser` object at each instruction execution request from the interpreter user. An operational style was used in order to specify the semantic rules of the *DInAton* language, influenced by Rowlet's work[13]. As part of its reaction behaviour to a string recognition, the `DInAton Parser` object collaborates with the `Adaptive Automaton` object that effectively encapsulates access to the *AA4J* library.

3. Result

The results so far produced by the *DInAton* language-based learning environment project are promising. The adherence of features offered by the *AA4J* library to the adaptive automata theory simplifies the design of the *DInAton* language. It starts from the premise that there is a semantic model of the theory appropriately implemented, allowing to focus the attention of the project on the interaction model with the learner.

Another point to highlight is the idea of an incremental construction of adaptive automata. It favours the design of learning environments that gradually explore the learner's cognitive processes. A typical situation is that in which a finite automaton is constructed, its properties are investigated (mainly the ones from level 3 recognition languages), and then it is used as the underlying device of a new adaptive automaton. Depending on the adaptive transitions that define such device, level 1 languages become recognizable by coupling the adaptive mechanism with the underlying finite automaton.

In addition to the study of automata properties, but still as an item of interest in an interactive learning environment, the textual and graphical representation features of the *DInAton* language should be punctuated. The **view** instruction with its different options becomes an important tool to observe the automaton built by the learner. It is possible to inspect a textual representation of the states of the automaton with `view text f1 states` (assuming`f1` is the automaton identifier). A graphical representation of the transitions can be calculated by executing the `view diagram f1 transitions` statement.

4. Discussion

Based on the features offered by the *DInAton* language, interactive learning environments can be designed such that automata-building experiments are performed. What happens when one removes one of the states of the automaton? What if it's in the initial state role? What if the state participates in state transitions? Questions such as these can be formulated in experiments that help learners develop their intuition about adaptive theory. Following in the same direction of automata construction, other experiments may redirect attention to the study of language recognizers. Variations in the automaton topology resulting from the execution of instructions within the interpretive environment enable string recognition tests (strings that may belong or not to the target language of the experiment), which favour the cognitive processes of comprehension.

The *DInAton* language evolution also establishes new directions for a refactoring of the *AA4J* library interfaces. The incremental construction nature of *DInAton* automata should also be contemplated by a review of the *AA4J* library, as its original design did not address requirements for use in an interpretive environment. Additionally, new features are needed in order to offer textual and graphical representations whenever needed by the language interpreter, as well as a straightforward mechanism for persisting the semantic object for later use. It is important to note that there is an ongoing effort on providing adaptive automata specifications using the XML markup language [14], such that a format mapper can correctly generate the corresponding semantic object in the *AA4J* domain.

As the theory of computation is found in the curricular bases of the higher education courses, it is hoped to open a way for the adaptive theory to reach this level of foundation. One of the main barriers to be addressed is the design of learning environments that favour the presentation of adaptive theory in a more didactic way. The expectation of an incremental automata construction tends to be an interesting option in this direction.

The *DInAton* language is still in the definition process. Currently, it supports the construction of finite automata and adaptive automata with certain topological constraints (many of them due to their use in a learning environment). It is intended to refine *DInAton* with support to the construction of structured pushdown automata and composition of submachines. The interpreter for the *DInAton* language is an experimental system still in heavy development. Contributions will be very well received.

References

1. J. J. Neto, Contribuições à metodologia de construção de compiladores, Tese de livre docência, Escola Politécnica da Universidade de São Paulo, São Paulo (1993).
2. J. J. Neto, M. E. S. Magalhães, Reconhecedores sintáticos - uma alternativa didática para uso em cursos de engenharia, in: XIV Congresso Nacional de Informática, Sâo Paulo, 1981, pp. 171–181.
3. I. S. Vega, Fábulas OCC-RDD: histórias didáticas para ambientes interativos híbridos e presenciais de aprendizagem, Tecnologia Educacional 12 (212) (2016) 105–118, ISSN 0102-5503.
   URL `http://www.abt-br.org.br/images/rte/212.pdf`
4. N. Chomsky, Three models for the description of language, IRE Trans. Inf. Theory (1956) 113—-124doi:10.1109/TIT.1956.1056813.
5. D. A. Watt, W. Findlay, Programming language design concepts, Wiley, 2004, pages = I-XVIII, 1-473.
6. D. P. Friedman, M. Wand, C. T. Haynes, Essentials of Programming Languages, 3rd Edition, The MIT Press, 2008.
7. R. M. Kaplan, Constructing Language ProProcess for Little Languages, John Wiley and Sons Inc., 1994, ISBN: 0-471-59754-6.
8. J. Bentley, Little language, Communications of the ACM 29 (8) (1986) 711–721.
9. P. R. M. Cereda, J. J. Neto, AA4J: uma biblioteca para implementação de autômatos adaptativos, in: Memórias do X Workshop de Tecnologia Adaptativa — WTA 2016, Escola Politécnica da Universidade de São Paulo, 2016, pp. 16–26, ISBN: 978-85-86686-86-3.
10. R. L. de Azevedo da Rocha, J. J. Neto, Autômato adaptativo, limites e complexidade em comparação com máquina de turing, in: Proceedings of the second Congress of Logic Applied to Technology — LAPTEC 2000, Faculdade SENAC de Ciências Exatas e Tecnologia, São Paulo, 2001.
11. T. Parr, The Definitive ANTLR Reference: Building Domain-Specific Languages, Pragmatic Bookshelf, 2007.

12. M. V. M. Ramos, J. J. Neto, I. S. Vega, Linguagens Formais: Teoria, Modelagem e Implementação, 1st Edition, Bookman, Porto Alegre, 2009.
13. T. Rowlett, The Object-Oriented Development Process: Developing and Managing a Robust Process for Object-Oriented Development, Prentice Hall, 2000, ISBN-13: 978-0130306210.
14. P. R. M. Cereda, J. J. Neto, XML2AA: geração automática de autômatos adaptativos a partir de especificações XML, to appear in: Memórias do XI Workshop de Tecnologia Adaptativa — WTA 2017 (January 2017).