Adaptive rule-driven devices - general formulation and case study

João José Neto
Escola Politécnica da USP
e-mail: joao.jose@poli.usp.br

Abstract

A formal device is said to be adaptive whenever its behavior changes dynamically, in a direct response to its input stimuli, without interference of external agents, even its users. In order to achieve this feature, adaptive devices have to be self-modifiable. In other words, any possible changes in the device's behavior must be known at their full extent at any step of its operation in which the changes have to take place. Therefore, adaptive devices must be able to detect all situations causing possible modifications and to adequately react by imposing corresponding changes to the device's behavior. In this work, devices are considered whose behavior is based on the operation of subjacent non-adaptive devices that be fully described by some finite set of rules. An adaptive rule-driven device may be obtained by attaching adaptive actions to the rules of the subjacent formulation, so that whenever a rule is applied, the associated adaptive action is activated, causing the set of rules of the subjacent non-adaptive device to be correspondingly changed. In this paper a new general formulation is proposed that unifies the representation and manipulation of adaptive rule-driven devices and states a common framework for representing and manipulating them. The main feature of this formulation is that it fully preserves the nature of the underlying non-adaptive formalism, so that the adaptive resulting device be easily understood by people familiar to the subjacent device. For illustration purposes, a two-fold case-study is presented, describing adaptive decision tables as adaptive rule-driven devices, and using them for emulating the behavior of a very simple adaptive automaton, which is in turn another adaptive rule-driven device.

Keywords - adaptive devices, rule-driven formalisms, self-modifying machines, adaptive decision tables, adaptive automata.

1 Introduction

Despite many adaptive devices have been created and used in the last decade, notations had not been elaborated in a way that the represented adaptive formalism be as close as possible to the original non-adaptive underlying formulation. Among several other reasons, self-modifying formalisms have not been extensively employed, as a consequence of the complexity of the existing formulations, which make them difficult to use.

In this work we introduce a general proposal for the formulation of adaptive devices based on rule-driven subjacent non-adaptive ones. We expect that the proposed formulation be clear, intuitive and easy to learn, without adding complexity to reading, writing or interpreting the notations. We also expect that the proposal be general enough, in the sense that it be insensitive to the underlying non-adaptive formalism, so that all achieved results remain valid even possibly changing the nature of the subjacent rule-driven system.

2 (Non-adaptive) Rule-driven devices

Formal state machines are popular mathematical tools used for describing and modeling actual real life systems. At each stage of their operation, these devices assume some configuration, which usually comprehends the contents of the whole set of information-holding elements and the current status of the device. In conjunction with the input stream yet to be processed, despite how the current configuration was reached, the device's configuration fully determines its further behavior. A formal machine operates

by successively moving the device from one configuration to another, in response to stimuli consumed from their input stream.
Without loss of generality, we may state that such devices start their operation at some initial configuration that follows well-known fixed restrictions. After having processed the full input sequence of stimuli, the device reaches some configuration which may indicate that its whole input stream has been either accepted or rejected. Therefore, we may split the set of all possible configurations for the device into two partitions: one for the accepting configurations and the other for the rejecting ones.
A (*non-adaptive*) *rule-driven device* is any formal machine whose behavior depends exclusively on a finite set of rules which map each possible configuration of the device into a corresponding next configuration. The device is said to be *deterministic* if and only if, for any given initial or intermediate configuration and any input stimulus, its defining set of rules determines one and only one next configuration. The device is *non-deterministic* otherwise. Non-deterministic devices allow more than one valid move at each moment. So, their use requires that all possible next moves be tried as intermediate steps toward some final accepting configuration. Inefficiencies caused by such trial-and-error operation usually turn non-deterministic devices inadequate for sequential implementation. Therefore, in order to achieve efficiency, whenever possible, choosing deterministic equivalent devices is highly recommended. Let us define ND = (C, NR, S, c_0, A, NA) where:

- ND is some *rule-driven device*, whose operation is established by a set of rules NR.
- C is the set of all its possible *configurations*, and $c_0 \in C$ is its *initial configuration*.
- S as the (finite) set of all possible *events* that are valid *input stimuli* for ND, with $\varepsilon \in S$.
- $A \subseteq C$ (resp. $F = C - A$) is the subset of its *accepting* (resp. *failing*) *configurations*.
- ε denotes "*empty*", and represents the *null* element of the set to which it belongs.
- $w = w_1 w_2 \ldots w_n$ is a *stream of input stimuli*, where $w_k \in S - \{\varepsilon\}$, $k = 1, \ldots, n$ with $n \geq 0$.
- NA is a (finite) set, with $\varepsilon \in NA$, of all possible symbols to be output by ND as side effects to the application of the rules in NR. In practice, *output symbols* in NA may be mapped into procedure calls, so an output generated by applying any rule may be interpreted as a call to its corresponding procedure.
- NR is the set of *rules* defining ND by a relation $NR \subseteq C \times S \times C \times NA$. Rules $r \in NR$ have the form $r = (c_i, s, c_j, z)$, meaning that, in response to any *input stimulus* $s \in S$, r changes the *current configuration* c_i to c_j, consumes s and outputs $z \in NA$ as a side-effect.

A rule $r = (c_i, s, c_j, z)$, with $r \in NR$; $c_i, c_j \in C$; $s \in S$; $z \in NA$, is said to be *compatible* with the current configuration c if and only if $c_i = c$ and s is either empty or equal to the device's current input stimulus. In this case, the application of a single compatible rule moves the device to configuration c_j (denoted by $c_i \Rightarrow^{s} c_j$) and appends z to its output stream. Note that s, z or both may be empty. Let $c_i \Rightarrow^{\sim} c_m$, $m \geq 0$ denote $c_i \Rightarrow^{\varepsilon} c_1 \Rightarrow^{\varepsilon} c_2 \Rightarrow^{\varepsilon} \ldots \Rightarrow^{\varepsilon} c_m$, an optional sequence of empty moves. Let $c_i \Rightarrow^{\sim w_k} c_j$, denote $c_i \Rightarrow^{\sim} c_m \Rightarrow^{w_k} c_j$, an optional sequence of empty moves followed by a non-empty one consuming the symbol w_k. An input stream $w = w_1 w_2 \ldots w_n$ is said to be *accepted* by ND when $c_0 \Rightarrow^{\sim w_1} c_1 \Rightarrow^{\sim w_2} \ldots \Rightarrow^{\sim w_n} c_n \Rightarrow^{\sim} c$ (for short, $c_0 \Rightarrow^{w} c$, with $c \in A$). Complementarily, w is said to be *rejected* by ND when $c \in F$. The *language* described by ND is the set $L(ND) = \{ w \in S^* \mid c_0 \Rightarrow^{w} c, c \in A \}$ of all streams $w \in S^*$ that are accepted by ND.

3 Adaptive devices

Define T as a built-in time counter that starts at 0 and is automatically incremented by 1 when a non-null adaptive action is executed. Each value k assumed by T may subscript the names of time-varying sets. In this case, it selects AD's operation step k.

A device AD=(ND_0, AM) is said to be adaptive whenever, for all operation steps k≥0, AD follows the behavior of ND_k until the execution of some non-null adaptive action starts its operation step k+1 by changing its set of rules.

At any AD's operation step k≥0, being ND_k the corresponding subjacent device defined by NR_k, the execution of some non-null adaptive action evolves ND_k to ND_{k+1}. So, AD starts its operation step k+1 by creating the set NR_{k+1} as an edited version of NR_k. Thereafter, AD will follow the behavior of ND_{k+1} until a further non-null adaptive action causes another step to start. This procedure iterates until the input stream is fully processed.

AD starts its operation at c_0, with an initial shape AD_0=(C_0, AR_0, S, c_0, A, NA, BA, AA). At step k≥0, an input stimulus (always) moves AD to a next configuration and then starts its operation step k+1 if and only if a non-adaptive action is executed. So, being AD at step k, with shape AD_k=(C_k, AR_k, S, c_k, A, NA, BA, AA), the execution of a non-null adaptive action leads to shape AD_{k+1}=(C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA). In this formulation:

- AD=(ND_0, AM) is some adaptive device, given by an initial subjacent device ND_0 and an adaptive mechanism AM.
- ND_k is AD's subjacent non-adaptive device at some operation step k. ND_0 is AD's initial subjacent device, defined by a set NR_0 of non-adaptive rules. By definition, any non-adaptive rules in any NR_k mirror the corresponding adaptive ones in AR_k.
- C_k is the set of all possible configurations for ND at step k, and $c_k \in C_k$ is its starting configuration at step k. For k=0, we have respectively C_0, the initial set of valid configurations, and $c_0 \in C_0$, the initial configuration for both ND_0 and ND.
- ε ("empty") denotes the absence of any other valid element of the corresponding set.
- S is the (finite, fixed) set of all possible events that are valid input stimuli for AD ($\varepsilon \in S$).
- $A \subseteq C$ (resp. F = C−A) is the subset of its accepting (resp. failing) configurations.
- BA and AA are sets of adaptive actions, both containing the null action ($\varepsilon \in BA \cap AA$)
- $w = w_1 w_2 \ldots w_n$, k = 1, ... , n is a stream of input stimuli, where $w_k \in S - \{\varepsilon\}$.
- NA, with $\varepsilon \in NA$, is a (finite, fixed) set of all possible symbols to be output by AD as side effects to the application of adaptive rules. Just like in non-adaptive devices, the output stream may be interpreted as a corresponding sequence of procedure calls.
- AR_k is a set of adaptive rules, given by a relation $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$. In particular, AR_0 defines the initial behavior of AD. Adaptive actions map the current set of adaptive rules AR_k of AD into a new set AR_{k+1} by adding adaptive rules to AR_k. and/or deleting rules from it. Rules ar$\in AR_k$ have the form ar = (ba, c_i, s, c_j, z, aa), meaning that, in response to some input stimulus $s \in S$, ar initially executes the adaptive action $ba \in BA$; if the execution of ba eliminates ar from AR_k, the execution of ar is aborted; otherwise, it applies the subjacent non-adaptive rule nr = (c_i, s, c_j, z) $\in NR_k$, as described before; finally, it executes adaptive action $aa \in AA$.
- Define AR as the set of all possible adaptive rules for AD.
- Define NR as the set of all possible subjacent non-adaptive rules for AD.
- $AM \subseteq BA \times NR \times AA$, defined for a particular adaptive device AD, is an *adaptive mechanism* to be applied at any operation step k to each rule in $NR_k \subseteq NR$. AM must be such that it operates as a function when applied to any sub-domain $NR_k \subseteq NR$. That will determine a single pair of adaptive actions to be attached to each non-adaptive rule.

  The set $AR_k \subseteq AR$ may be synthesized by collecting all adaptive rules obtained by merging such pairs of adaptive actions to the corresponding non-adaptive rules in NR_k. Equivalently, we may build NR_k by removing all references to adaptive actions from the rules in AR_k.

The algorithm below sketches the overall operation of any rule-driven adaptive device.

1. Set the device at its initial configuration.
2. Set the input stream at its leftmost event.
3. If there are no more events to be processed, then go to step 8
   else feed the device by picking the next event to be processed.
4. Choose the next adaptive rule to be applied:
   Given the current configuration $c_T \in C_T$ and stimulus $s_T \in S$, extracted from the input stream not yet processed, search the set NR_T for compatible rules, i.e., a set of rules that may be applied under the current circumstances, and collect them in a set $CR_T=\{ar \in AR_T \mid ar = (ba, c_T, s, c, z, aa), s \in \{s_T,\varepsilon\}; c,c_T \in C_T; ba \in BA; aa \in AA; z \in NA\}$
   There are three cases to be considered:
   - if CR_T is empty, no moves at all are allowed for the device (because AR_T is incompletely specified), then the input stream is rejected by default.
   - if $CR_T = \{ ar^k \}$for some $ar^k = (ba^k, c_T, s, c^k, z^k, aa^k) \in AR_T$, then a single rule is available, for deterministic application. By doing so, the device will reach a well-defined next configuration $c_{T+1}=c$.
   - if $CR_T =\{ar^k=(ba^k, c_T, s, c^k, z^k, aa^k) \mid k=1,2,...,m\}$, then all these m rules are equally allowed for being non-deterministically applied to the current configuration, so all of them are applied in parallel to the current configuration, as usual in the operation of non-deterministic devices. Obviously, in non-parallel environments, such parallelism must be exhaustively simulated, e.g. by applying some backtracking strategy.
5. If adaptive action ba^k is the null adaptive action ε in the rule being applied, then proceed to step 6, otherwise, apply the adaptive action ba^k to the current set of rules, yielding a new intermediate configuration for the device AD. Note that, in some cases, even the adaptive rule ar^k being applied may erase itself by executing ba^k. In such an extreme case, the application of ar^p is aborted by returning to step 4.
6. Apply the rule $nr^k=(c_T, s, c^k, z^k)$, just as defined by the underlying non-adaptive device, to the current (intermediate) configuration of AD, yielding another (intermediate) configuration.
7. If the adaptive action aa^k in the adaptive rule ar^k being executed is the null adaptive action ε, proceed to step 3, otherwise apply aa^k to the current set of adaptive rules, finally yielding the next configuration for the device. As in step 5, the execution of the adaptive action may also erase ar^k. In this case, however, no further action is needed.
8. if the current configuration is an accepting configuration, then accept the input stream, otherwise reject it, and then stop.

4 Example

Decision tables are tools very popular among information systems programmers and software engineers. Adaptive decision tables constitute an interesting application of the concept of adaptive devices to the field of information systems. Adaptive decision tables may be defined as the class of adaptive devices that use the traditional (non-adaptive) decision tables as their underlying formalism.

4.1 (Non-adaptive) Decision tables

Decision tables may be viewed as tabular tools that encode a set of rules represented by conditions and corresponding actions to be executed when those conditions are matched. In typical decision tables (fig. 1) rules are represented as columns while rows are employed to encode conditions (condition rows) and actions (action rows). In each rule, marked cells corresponding to each condition row refer to relevant conditions to be tested, and indicate whether that condition is to be tested for true or false (non-marked conditions are not to be tested), while marked cells in action rows indicate that the corresponding action is expected to be performed as a response to a match in all marked conditions.

						Rules ↓				
Rule nº →		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Condition rows	**c_1**	T	T	**F**	-	-	F	T	T	-
	c_2	-	T	**T**	-	-	F	-	F	T
	...									
	c_n	-	-	**-**	-	T	-	F	F	-
Action rows	**r_1**	F	T	**T**	T	F	F	F	F	T
	r_2	F	F	**T**	T	F	T	T	F	F
	...									
	r_m	F	F	**F**	F	T	F	F	T	T

Fig. 1 - Structure of a typical non-adaptive decision table.

Rules in the decision table are encoded as follows:

- *Condition rows:* cells of the rule corresponding to conditions to be tested are filled with a boolean value (True or False) corresponding to the particular value to be tested for that condition. A special null mark (-) indicates that the associated condition is not to be tested.
- *Action rows:* cells of the rule corresponding to actions to be performed are filled with the value True. For all actions not to be executed, that cell is filled with False.

The operation of such non-adaptive decision tables is quite straightforward:

1. First, the status of the system is checked against the combinations of conditions stated in each of the rules encoded in the table.
2. If no rule matches the current status, then no action is executed at all.
3. If a single rule matches the current status, then we have a deterministic choice, so the matching rule is selected to be applied.
4. If more than one rule match the current status, then we face a non-deterministic situation. Consequently, all those rules are to be applied in parallel. In practice, parallelism may be simulated, e.g. by some exhaustive backtracking strategy.
5. The selected rule is then applied by executing the set of all actions indicated with a boolean value True in the cells of the rule corresponding to action rows.
6. Once the selected rule has been applied, the decision table gets ready to be used again.

For instance, let us assume that condition c_1 is False and condition c_2 is True. In our decision table, obviously only rule 3 (shaded in fig.1) matches such status. Therefore, in this case the decision table will activate actions ra_1 and ra_2 for execution, as we may easily observe by inspecting the action rows specified in rule 3. Note that if rule 1 had been selected instead, no actions would have been called at all, since all action rows in rule 1 are filled with False. It is obvious from this simple example that decision tables are very easy to design and use.

Unfortunately, these classical devices are static, in the sense that their individual rules are all predefined and never change along the whole operation of the device. Furthermore, the set of rules defining classical decision tables are not allowed to change, so in classical decision tables there is no dynamic inclusion or exclusion of rules. In order to provide more flexibility to this useful and popular tool, we may use it as the basic underlying non-adaptive formalism for building a far more powerful adaptive device.

4.2 Adaptive decision tables

Adaptive decision tables (fig. 2) are easily obtained from conventional ones by adding to them further rows encoding the adaptive actions to be performed before ("before-" adaptive action rows) and after the rule is applied ("after-" adaptive action rows).

		Rules ↓								
Rule nº →		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Condition rows	**c_1**	T	T	**F**	-	-	F	T	T	-
	c_2	-	T	**T**	-	-	F	-	F	T
	...									
	c_n	-	-	**-**	-	T	-	F	F	-
Action rows	**z_1**	F	T	**T**	T	F	F	F	F	T
	z_2	F	F	**T**	T	F	T	T	F	F
	...									
	z_m	F	F	**F**	F	T	F	F	T	T
"before-" Adaptive Action rows	**ba_1**	T	F	**F**	F	F	F	F	T	F
	ba_2	F	F	**T**	F	T	F	F	T	F
	...									
	ba_p	F	T	**F**	F	F	F	F	F	F
"after-" Adaptive Action rows	**aa_1**	F	T	**T**	F	F	F	F	T	F
	aa_2	F	F	**F**	T	F	T	F	T	F
	...									
	aa_q	F	F	**F**	F	F	T	F	T	F

Fig. 2 - Structure of an adaptive decision table based on the non-adaptive table in fig. 1 as its subjacent device.

When adaptive actions are executed, the table usually has its set of rules modified, therefore correspondingly changing the number of columns in the adaptive decision table. Note that with the chosen layout, however, the number of rows of the adaptive decision tables remains unchanged, since adaptive actions do not modify any of the items encoded in their rows. This property is truly valuable for implementation purposes.

For operating such an adaptive device, the subjacent non-adaptive decision table is first used for determining the rule(s) matching the current situation of the condition predicates. Then, the selected adaptive rule is performed by executing the indicated "before-" adaptive actions, then applying the subjacent non-adaptive rule, and finally executing the indicated "after-" adaptive actions.

In some cases, when executing its adaptive actions, some adaptive rule being applied may even exclude itself. Whenever the currently used rule happen to be eliminated by its own before- adaptive action, its application is aborted, and the next rule to be applied is elected from the resulting set of rules.

The aspect of an adaptive decision table is shown in the example depicted ahead. The upper half of the table in the figure corresponds to the corresponding underlying non-adaptive decision table, while its lower half represents the attached adaptive mechanism

Note that by associating adaptive actions in this way to the usual formulation of decision tables, no substantial changes are introduced for the user, since at first glance adaptive actions might be simply interpreted as additional standard actions to be executed in response to some particular matching of conditions. From a conceptual viewpoint, however, the execution of adaptive actions has significantly deeper implications, since their execution affects the decision table itself, allowing changes to be imposed to its own behavior.

4.3 Application

In the table illustrated in fig. 2 above, adaptive rule 3 is marked with a shaded background.

It is activated when condition c_1 is False and condition c_2 is True, regardless to the other conditions. Under this situation, the application of this adaptive rule operates as follows:

1. Execute adaptive action b_2 (which will probably change the rule set of the device) *before* the subjacent non-adaptive rule is applied.
2. Apply the underlying rule: actions r_1 and r_2 are performed just as they would be executed in the classical non-adaptive case.
3. Execute adaptive action a_1 (probably changing the decision table again) *after* the application of the underlying non-adaptive rule.

4.3.1 Adaptive functions

Adaptive actions may be defined as abstractions called adaptive functions, in a way correspondingly similar to that of function calls and function declarations in a usual programming language. Adaptive functions define generic abstractions while adaptive actions correspond to adaptive function specific calls. Adaptive actions customize the corresponding adaptive function abstraction by assigning arguments to their formal parameters according to each particular needs.

4.3.2 Specifying adaptive functions

In order to state exactly how each adaptive action is expected to operate, we need to provide some further information: the name of the corresponding adaptive function, the set of parameters to be used, the elementary adaptive actions to be applied and the exact way parameters, variables and generators are to be employed.

Thus, the specification of adaptive functions must include the following items:

- *name*: a symbolic name, used for referencing adaptive functions. When calling adaptive actions, the name of the corresponding adaptive function is used to select among available adaptive actions.
- *(formal) parameters*: a set of symbolic names that are used for referencing values passed as arguments to an adaptive function at the time it is called. All instances of the formal symbolic parameters, once replaced with the values associated to their corresponding arguments within the body of the adaptive function, may not be further modified throughout the execution of the adaptive function.
- *variables*: these are symbolic names used for holding values resulting from the application of some rule-searching elementary adaptive function. Variables are filled only once during the execution of an adaptive function, and its value remain unchanged during the execution of the adaptive function.
- *generators*: these elements are symbolic names that refer to new values each time they are used. Once generators are filled with some value, they do not change any more while the adaptive function is active.
- *body*: the body of an adaptive function encodes all the editing operations needed to make the desired changes to the current set of rules of the decision table.

  Elementary adaptive actions are editing primitives that allow either testing the rule set or specifying single modifications to the rules of an adaptive decision table. The body of an adaptive decision table consists essentially of a set of elementary adaptive actions.

  There are three kinds of elementary adaptive actions that may be combined within the body of an adaptive function in order to specify its operation:

  - *rule-searching elementary adaptive actions*: these actions do not modify the set of rules, but allow searching the rule set for rules matching a given pattern.
  - *rule-erasing elementary adaptive actions*: these actions remove rules that match a given pattern from the current set of rules.

- *rule-inserting elementary adaptive actions*: these actions allow adding a rule with a specified pattern to the current set of rules.

4.3.3 Encoding adaptive functions

In order to encode adaptive functions, a format must be chosen for each of the component items listed above. It should be convenient that the format be similar to that of adaptive decision tables. We chose to include the specification of adaptive functions as part of the adaptive decision table itself, since adaptive functions are meaningless outside the environment they act on. The format adopted for encoding adaptive functions within adaptive decision tables will be informally introduced through the following example.

4.3.4 Overall format for adaptive decision tables

Adaptive decision tables as defined here will be drawn in an extension of the notation already discussed for non-adaptive tables. So, the subjacent decision table is represented as usual and the adaptive mechanism is added by inserting the following elements in the rows:

- one heading row containing a tag for specifying the type of each column (H= header of the specification of an adaptive function; +, -, ? = including, excluding, inspecting elementary adaptive action; S= starting rule; R= normal rule; E= ending rule)
- one extra row for the names of each adaptive function used
- one extra row for the names of each parameter, variable or generator used by the adaptive functions
- in the example below, for better legibility of the table, assignments and comparisons referring to variables used by standard (non-adaptive) actions are denoted explicitly and not as function calls

Similarly, the following additions have been done to the columns of the table:

- one header column for each adaptive function (tag = H) starting the specification of the adaptive function. This header must include a tag B or A in the cell corresponding to the name of the before- or after- adaptive functions, respectively, a tag P in each cell corresponding to a formal parameter, a tag V in each cell corresponding to a variable and a tag G in each cell corresponding to a generator. Each header column is followed by a set of columns related to elementary adaptive actions. This set is finished when a starting rule or another header is found.
- in columns denoting elementary adaptive actions (tags +, - or ?) the cells corresponding to conditions are filled with the value the condition is to be tested against; cells corresponding to actions to be executed are marked; cells corresponding to assignments are filled with the value to be assigned. Required adaptive actions are marked, and the cells corresponding to their parameters are filled with a constant or the name of a variable, a parameter or a generator to be passed as an argument. Homonymous parameters must be avoided between adaptive functions called within the same rule.
- one column for the starting rule (tag = S) of the adaptive automaton, standing for the rule to be applied before any other. In this column, actions are activated in order to initialize all initial operating conditions for the device. A set of normal rules follow this column, ending with the ending rule, which closes the specification of the table.
- columns denoting normal rules (tag = R) specify all rules defining the decision table. Each normal rule is specified by filling condition cells to be tested with constants or names of variables, generators or formal parameters; actions cells and adaptive actions are specified just as described above for elementary adaptive actions.
- the single column denoting the ending rule (tag = E) serves as a delimiter for the set of current rules in the adaptive decision table, and represents only a logical marker.

4.3.5 Illustrative Example

Let us illustrate the encoding of adaptive functions by means of a low-complexity adaptive example. It is shown as a complete adaptive decision table in fig. 3. In this example, we define two adaptive functions: The first adaptive function is named X and has two formal parameters, p1 and p2, and uses one generator, g1, while the second one, Y, has one formal parameter, q1. Note that no variables are used in this example, but if there were variables, new corresponding rows would have been added, since they are denoted and used in the same way parameters and generators are.

		Tag→	**H**	+	+	+	+	-	**H**	-	+	**S**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**E**
subjacent decision table	**condition**	**state =**		***p1***	***g1***	***g1***	***g1***	***p1***		***q1***	***q1***		**"I"**	**"I"**	**"J"**	**"J"**	**"J"**	**"J"**	"K"	"L"	
		input =		***p2***	"λ"	"d"	"↵"	***p2***		"↵"	"↵"		"λ"		"λ"	"d"	"↵"				
	action	**state :=**		***g1***	**"J"**	**"J"**	"L"	**"J"**		"L"	"K"	**"I"**	**"J"**		**"J"**	**"J"**	"L"				
		get (input)										✓	✓		✓	✓					
		Accept :=												⌨				🖱	✓	✓	
adaptive functions	**function names**	***X***	**B**		✓	✓		✓					✓								
		Y					✓		**A**	✓							✓				
	other names	***p1***	**P**		***g1***	***g1***		***p1***					**"I"**								
		p2	**P**		"λ"	"d"		***p2***					"λ"								
		q1					***g1***		**P**	***q1***							**"J"**				
		g1	**G**																		

Figure 3 - An illustrative adaptive decision table emulating the behavior of a very simple adaptive automaton that accepts identifiers and memorizes them.

Other features that were not used in this example are the calls to before- and after-adaptive actions within the adaptive functions. If they were present, additional appropriately tagged columns would have been included for representing them.

The very simple adaptive decision table in fig. 3 illustrates the encoding of the information needed for implementing a simple adaptive device.

Not all possible options of the model have been explored, but the given illustration is complex enough to be used as a guide for developing other projects with this technique.

This example illustrates the use of adaptive decision tables as an alternative way to implement adaptive automata-based logic. Applications of adaptive automata have been shown in [Jos00]. The target of this decision table is to read an input sequence formed of letters and digits and to collect identifiers formed in the usual way, say, as a non-empty sequence of letters and digits starting with a letter, and ending with the special end-marker ↵ **.** Whenever a new identifier is found, the adaptive decision table is adequately modified so that the identifier be thereafter registered as an already known one, while already known identifiers are simply accepted by the table without modifying it. So, it operates as a purely syntactic name-collecting device for which all needed available information is permanently encoded in the adaptive decision table.

Further descriptions of the operation of similar adaptive devices are found in [Jos93].

4.3.6 A step-by-step operation of the illustrative device

5 Previous experience with adaptive techniques

Adaptive technology concerns to techniques, methods and disciplines referring to actual practical applications of adaptive devices. Historically, adaptive devices emerged from the field of formal languages and automata. Consequently, early applications of such devices have been in the area of rigorous definition of formal and computer languages. In this area, we may list the works by Burshtein [Bur90], Shutt, Cabasino [Cab92], Rubinstein [Rub95b], Neto [Jos94] and Iwai [Iwa00]. For example, adaptive automata have been proposed as a practical formalism for the representation of languages with context dependencies [Rub95b], [Jos94]. On another hand, adaptive grammars were also introduced as generative devices whose operation also allows their use in the rigorous definition of context-dependent languages [Rub95b], Iwai [Iwa00].

An early meta-system based on adaptive automata has been proposed and implemented which allowed many tests to be carried out in a comfortable form [Per97]. This work was a practical proof that adaptive engines might be useful and not so much difficult to design and implement. Burshtiein [Bur90], Shutt [Shu93] and Christiansen [Chr90] have proposed adaptive grammatical devices whose operation reminds two-level Von Wijngaarden grammars [Wij75], with enough power to express complex type-1 and type-0 languages. For use in the specification and analysis of real time reactive systems, we also find some works based on adaptive versions of classical statecharts [Alm95]. As an evolution of this work, in addition to reactive aspects, mechanisms based on Petri Nets have been added to adaptive statecharts for explicit analysis of synchronization aspects of concurrent adaptive systems [San97].

Another extremely interesting manifestation of the power and practicality of adaptive devices for the specification and implementation of complex systems has been the formulation and use of Adaptive Markov chains, which have been very successfully used in the design and physical implementation of a computer music-generating system [Bas99]. Adaptive devices are also being tested in the field of decision-making systems. An adaptive automata-powered system prototype was implemented as a tool to be used in the automatic generation and selection of solutions for problems with high computational complexity.

Artificial intelligence is a field that may be strongly benefited by adaptive technology, since adaptive devices have a built-in mechanism for acquiring, representing and manipulating knowledge. Although not been actually implemented, a proposal has been made for the application of adaptive automata in computer education, as a learning mechanism of a computer-aided tutoring system and as a model of the student's progress in such tutoring systems. So, learning is a natural feature shared by all adaptive devices.

In particular, a small experiment with regular languages has been reported that shows the potential of adaptive automata as a good device for constructing grammar inference systems [Jos98]. Another area in which adaptive devices showed their strength is the specification and processing of natural languages. One of the works in this field employed adaptive automata as the main mechanism of an automatic tagging system for texts in Portuguese language. Further works are being currently developed in this direction, with many good intermediate results in the representation of syntactical context-dependent features of natural languages. Simulation and modeling of intelligent systems are also concrete applications of adaptive formalisms, as it was illustrated in the description of the control mechanism of an intelligent autonomous vehicle that collects information from its environment and builds a map for easier navigation. Many other applications for adaptive devices are possible in several fields. They must be extensively explored through the well-succeeded search for simple and efficient alternative ways to perform complex computation tasks.

6 Conclusions

As a result of this work, we have achieved, for adaptive devices, a formulation in which nothing beside the formal adaptive mechanisms is introduced, giving the proposed formulation the character of a simple extension of the underlying non-adaptive device.

So, the integrity and even the intuition of the underlying formulation in which the adaptive device is based are preserved, as well as all their properties.

Consequently, after getting familiarized with the concept of adaptive devices, users have no extra need of learning further concepts and notations, so they are free for using already designed and tested non-adaptive devices as a basis for easily building adaptive versions.

From another viewpoint, with the proposed formulation users may directly identify the underlying non-adaptive device at any moment during the operation of an adaptive device, so simplifying its debugging effort and increasing the comprehension of the adaptive device by its designers. Our proposal has a very clean formulation, allowing the user to be permanently aware of all phenomena concerning the underlying mechanism.

No new notations or concepts are introduced, except those involving adaptive features, therefore, our proposal offers a formulation that is indeed intuitive and easy to learn. It is also general to a large extent, since it does not depend on the nature of the underlying non-adaptive formalism chosen.

We expect that this simple contribution encourage not only the revisiting and use of existent self-modifying formalisms, but also the formulation of new adaptive devices for solving problems that are hard to solve with usual non-adaptive tools.

7 References *(items marked with asterisk are available in Portuguese only)*

[Alm95]* ALMEIDA JUNIOR, J.R. STAD - Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos. São Paulo 1995, 202p. Doctoral Thesis. Escola Politécnica, Universidade de São Paulo.

[Bas99] BASSETO, B. A.; JOSÉ NETO, J. A stochastic musical composer based on adaptive algorithms. Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação. SBC-99 PUC-Rio, Vol 3, pp105-13, 19 a 23 de julho de 1999

[Bur90] BURSHTEYN, B. Generation and recognition of formal languages by modifiable grammars. ACM SIGPLAN Notices, v.25, n.12, p.45-53, 1990.

[Cab92] CABASINO, S.; PAOLUCCI, P.S.; TODESCO, G.M. Dynamic parsers and evolving grammars. ACM SIGPLAN Notices, v.27, n.11, p.39-48, 1992.

[Iwa00]* IWAI, M. K. Um formalismo gramatical adaptativo para linguagens dependentes de contexto. São Paulo 2000, 191p. Doctoral Thesis. Escola Politécnica, Universidade de São Paulo.

[Jos93]* JOSÉ NETO, J. Contribuição à metodologia de construção de compiladores. São Paulo, 1993, 272p. Thesis (Livre-Docência) Escola Politécnica, Universidade de São Paulo.

[Jos94] JOSÉ NETO, J. Adaptive automata for context-dependent languages. ACM SIGPLAN Notices, v.29, n.9, p.115-24, 1994.

[Jos98] JOSÉ NETO, J.; IWAI, M.K. Adaptive automata for syntax learning. XXIV Conferencia Latinoamericana de Informática CLEI'98, Quito - Ecuador, Centro Latinoamericano de Estudios em Informatica, Pontificia Universidad Católica Del Ecuador, tomo 1, pp.135-146. 19 a 23 de Outubro de 1998

[Jos00] JOSÉ NETO, J. Solving Complex Problems Efficiently with Adaptive Automata. CIAA 2000 - Fifth International Conference on Implementation and Application of Automata, July 2000 - London, Ontario, Canada

[Per97]* PEREIRA, J.C.D; JOSÉ NETO, J. Um ambiente de desenvolvimento de reconhecedores sintáticos baseados em autômatos adaptativos. II Brazilian Symposium on Programming Languages (SBLP'97), 3-5 September 1997, Institute of Computing, State University of Campinas, Campinas, SP, Brazil, pp.139-50

[Rub95b] RUBINSTEIN, R.S.; SHUTT. J.N. Self-modifying finite automata: An introduction, Information processing letters, v.56, n.4, 24, p.185-90, 1995.

[San97]* SANTOS, J.M.N. Um formalismo adaptativo com mecanismos de sincronização para aplicações concorrentes. São Paulo, 1997, 98p. M.Sc. Dissertation Escola Politécnica, Universidade de São Paulo.

[Shu93] SHUTT, J.N. Recursive adaptable grammar. M.S. Thesis, Computer Science Department, Worcester Polytecnic Institute, Worcester Massachusetts, 140p, 1993.

[Wij75] WIJNGAARDEN, A.V., et al, Revised report on the Algorithmic Language Algol 68, Acta Informatica, v.5, n.1-3, p.1-236, 1975.