$\mathcal{AI}$ IN SYSTEM THEORY: AN OBSERVABILITY VIEW

David A. Maluf *

ABSTRACT

The problem of the construction of a Logic Observer, which generates a sequence of propositions that correctly describe system state properties, is mainly intended to solve the state estimate of finite state machines. Of particular interest is the convergence of the logic-based dynamical system statements (in an appropriate sense) to true characterizations of the system state. This paper demonstrates that this observer can be built within a logic-based framework. It starts with an informal discussion of the nature of logic – focusing on the predicate calculus – and discusses the fundamental concepts associated with these formal systems. Later in the paper, the formal definition of the concepts of the logic-based dynamical observer is presented. An arbitrary automaton is selected as an example to understand the simulation and the expert system presented to accomplish the task. In this paper, the assumption is that the automata representation in state space systems is expressed by the quintuple set of states, set of inputs, set of outputs, transition functions and output functions. An automaton is taken as a deterministic input-state-output (finite) system.

Key Words— discrete event system, automata, predicate calculus, logic system, dynamical logic observer, observability, artificial intelligence.

INTRODUCTION

The *Simulation of Dynamical Logic Observers for Finite Automata* began with the introduction of the first series of publications on Dynamical Logic Observers in 1988 [1][2]. The concept of involving *Artificial Intelligence* in *Systems and Control Theory* enabled the application of the methodology and foundational aspects of mathematical logic and computer science to problems involving deterministic input-state-output dynamical systems [8]. The formulation of dynamical systems with the framework of *Knowledge Representation* may be viewed as a challenge from Artificial Intelligence (AI) to Systems and Control Theory (SCT). Partly in response to this, the concept of *Observability* in Systems and Control Theory has been studied within Artificial Intelligence methodologies.

The problem was originally formulated by the construction of a *classical dynamical system* which generates a sequence of state estimates and a *logical dynamical system* which generates a sequence of propositions that correctly describe the properties of the automaton's state. A focus of interest of this paper is the particular case where classical observer estimates converge to the correct current state and where the logic observer also converges to the correct characterizations of the current state.

The simulation mentioned in the title is mainly based on the dynamical logic observer. The classical dynamical observer is discussed in order to understand the shift from classical systems to logic systems [7]. The simulation was written in *Lisp* and is acceptable to any *Common Lisp* listener.

The beginning of this paper provides a review of the common concepts introduced in the field of *finite state machines* [6]. Basic definitions related to the terminology are presented in detail in the first part of the paper, and this section introduces the reader to both types of dynamical observers: namely, the classical observer [1], and the logic observer which we simulate; the latter constitutes the second part of the paper. In the last section, attempts at finding rules concerning the automata dynamics using logic observers are discussed.

Dynamical Observers for Finite Automata

In this section of the paper, a few basic definitions will be stated as a reference and review for the reader. Still, some basic knowledge of finite automata and control theory is necessary for a complete understanding of the problem.

An *input-state-output finite automaton* is defined as a quintuple $\mathcal{M} = (X, U, Y, \Phi, \eta)$ where X is a finite set of states, U, a finite set of inputs, Y, a finite set of outputs, Φ, a transition function defined as $X \times U \rightarrow X$ and η is output function defined as $X \rightarrow Y$.

One of the definitions of the observer problem which is phrased as the *dynamic initial*

*Department of Computer Science, Stanford University, Stanford CA 94305. maluf@db.stanford.edu, Fax: (415) 725-2588

state observer for a certain automaton is the problem of estimating the automaton initial state over a sequence of finite observations, where normally the observations consist of pairs of inputs and outputs [9]. Similarly, the *dynamic current state observer* is the problem of estimating the automaton current state over a sequence of observations. In such cases, the finite length of observations plays a crucial role in the estimation of the initial state since the observability is related to convergence of the estimation.

The nature of the output function η is such that it is not necessarily a one-to-one mapping (which would transform the partial observer problem into a complete forward problem). However, the output function is defined for all the states of the automaton which generate subsets of states. The automata under study are also often referred to as *partially observed automata* and without further mention we shall take the automata to be deterministic in both transition and output functions. It should be noted that the type of finite automata studied in this paper are also often referred to as *finite machines*.

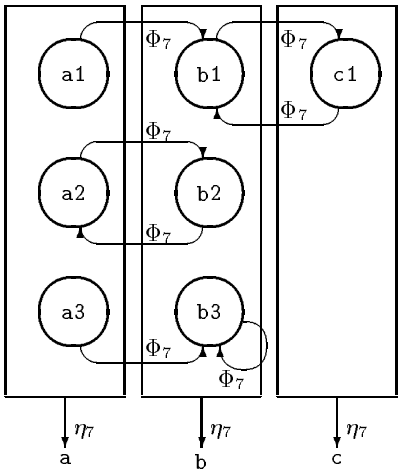

Figure 1: The $\mathcal{M}_7$ automaton. States, transitions and output functions.

A commonly used example of a finite state machine is chosen and defines the transition function as the same for all controls where the input set contains a single element (e.g, the tick of a clock). The assumption taken of the transition function simplifies the understanding of the classical and logic observer. Figure 1 shows the automaton known as the $\mathcal{M}_7$ *automaton*. This automaton contains seven states labeled `(a1,a2,..,c1)` and grouped under three distinct output functions η_7 labeled `a,b,c`. Φ_7 is the transition function which is considered constant as mentioned earlier in this paragraph. In the simulation section of this paper, the transition function will no longer be considered unique and the input set will be considered as containing a finite number of controls.

Classical Observers

The classical observer is considered an important step in the understanding of the logic observer. This observer introduces the idea of and need for an efficient computational implementation of logic concepts. What is required of the observer is to produce the estimates in the proper format. For the $\mathcal{M}_7$ automaton, a question could be asked such as in which state would the machine be after observing an output of `(a)`. Eventually the observer would provide the estimate `(a1,a2,a3)`. Adding a next observation `(b)`, the observer would provide the estimate of `(b1,b2,b3)`. In such a case it takes a third observation to make the observer lock on one state. What was stated before defines one branch out of many of the automaton tree as shown in Figure 2.

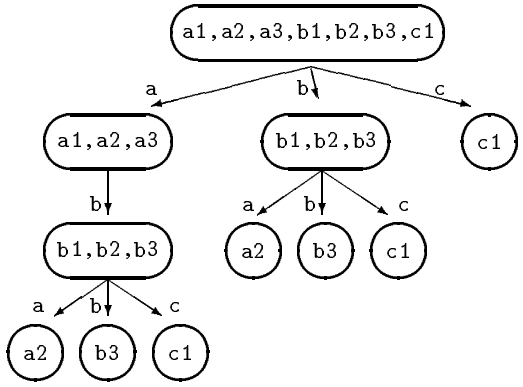

Figure 2: $\mathcal{M}_7$'s current state observer estimation tree.

Looking again at the $\mathcal{M}_7$ automaton, it is obvious that in some other combinations of observations the observer would have locked on to the correct state earlier if the initial observation were a `(c)`. Figure 2 shows all possible estimations the classical observer will return.

The initial simulation of the classical dynamical observer was performed in [2] and was written in Fortran. Two algorithms were written for the computation of the current state observer and initial state observer.

The simulation of the classical observer is performed by a *classical* computation (e.g., nested iterations) for which efficiency cannot be claimed. Repetitive calculation is frequently found which makes the simulation time-consuming for large state machines. With the emergent computer technology, the observer was rewritten and consists of additional enhancement of its predecessor classical observer. The classical observer was written in

Lisp and showed nested triple iterations; the programming of the classical observer starts with the whole set of space states and generates a tree based on every combined pair of observations created from the transition function and output function. The search continues for every non-singleton created. Such a search can run indefinitely: that is, there is no convergence of the tree which implies the automaton is not observable. Such behavior is proven by theorems and provides bounds to the depth of the tree in the case where all the branches converge to singletons.

Logic Observers

In the original reference [2], the *Logic-Based Dynamical Observer* of an automaton $\mathcal{M}$ is defined as a tree of families of first order theories where the automaton is $\mathcal{M} = (X, U, Y, \Phi, \eta)$ as defined in the beginning of the paper.

A family of the first order indexed by a set of steps is denoted by the formation rules specifying the well-formed formulas, by the logical axioms of predicate calculus, by the dynamical axioms which describe the state transition function, and by a set of observation axioms specifying a sequence of input-output pairs which belong to the sets X and Y.

Similarly, with the classical observer, the logic observer is required to produce estimates of the current (or initial) state from the dynamics of the automaton which are described by the transition and output functions and by the set of observations. Since the automaton can be fully expressed by propositional calculus, it becomes necessary to present the automaton by the corresponding predicates [5].

Encoding of Input-State-Output Machines

In this subsection, a description of the predicates that define an automaton and the necessary rules to achieve the simulation will be discussed. Looking at the definition and structure of an automaton [4], it is easy to see the necessity for two independent collections of axioms to represent the transition function and the output function. Based on the previous statement, such a requirement can be presented in two independent databases. Again, from the definition of the automaton, the transition function $\Phi : X \times U \rightarrow X$ which combines the set of states X with the input set U where every state $x \in X$ accepts every input $u \in U$. Or, written in a list mode:

`(NextState x u x')` .

`NextState` is an arbitrary description of the transition function database and is considered as an atom and used as a key-pad string [1]. Notice that it is possible that `x'` and `x` could be the same. The expansion of the predicate mentioned for the $\mathcal{M}_7$ automaton is shown later in this section.

Again, from the definition of the automaton, the output function $\eta : X \rightarrow Y$ also forces the state to produce an output y, where $y \in Y$. The encoding of the output function is of the form:

`(Emit x y)`

where `Emit` is also a description of the database. The concept of providing a description for the databases is useful when different dynamics of the transition and output functions are involved.

The encoding of the input-state-output automaton is achieved in a sequence of lists which eventually are kept and handled in such a way as to avoid duplication of the same axioms. Such encoding fits the requirement of both classical and logic observers.

T1:	`(NextState a1 u b1)`
T2:	`(NextState a2 u b2)`
T3:	`(NextState a3 u b3)`
T4:	`(NextState b1 u c1)`
T5:	`(NextState b2 u a2)`
T6:	`(NextState b3 u b3)`
T7:	`(NextState c1 u b1)`
F1:	`(Emit a1 a)`
F2:	`(Emit a2 a)`
F3:	`(Emit a3 a)`
F4:	`(Emit b1 b)`
F5:	`(Emit b1 b)`
F6:	`(Emit b1 b)`
F7:	`(Emit c1 c)`

Table 1: The $\mathcal{M}_7$ transition and output functions. The selection of `NextState` and `Emit` strings is arbitrary and provides a key-pad in handling the Databases.

SIMULATION OF DYNAMICAL LOGIC OBSERVERS

The basic challenge in the simulation of the logic observer was to generate the environment that would be able to handle the automaton and the observer efficiently. In the previous section, the encoding of the input-state-output machines showed the necessity of having two independent databases. For the logic observer in question, a set of rules can be defined to specify its goals [2] which are grouped in separate databases. Particular attention should

[1] Assuming the transition functions $\Phi_1 : X \times U \rightarrow X$, $\Phi_2 : X \times U^* \rightarrow X$ where $U \neq U^*$, it is necessary to group the functions separately, Φ_1:`(NextState-1 x` u `x')`, Φ_2: `(NextState-2 x` u^* `x')`.

[2] Usually it is expected from the logic observer to return a current estimate (or initial estimate) given the transition and output function database and a set of

be paid to the *Current Dynamic Observer* or `CDO-Next` facts which are generated as consequences defined by the rules of the logic observer. Another aspect of the logic observer is that the facts generated by the logic simulation are expressed within the framework of propositional calculus. This will enable the Logic Observer to use the previous expressions of the `CDO-Next` facts in the estimation problem. This aspect will be be discussed later in the building of the logic environment section.

Building a Dynamical Logic System Environment

This section is dedicated to the simulation of automata which is the basic concern of the current project. *Common Lisp* was chosen as the programming language. Lisp is considered to be the language of artificial intelligence and provides a powerful tool for symbolic problems which satisfies the requirements for the simulation of the logic observer.

Once the two collections of axioms representing the transition function and the output function are grouped, efficient methods for handling the databases are required. In the implementation of the simulation, functionalities common to the design of an expert system database are found. The major intent is to maintain different databases of facts and rules. A fact or facts are entered using the function `(db-add-facts form &optional (db database))`. These functions have the potential to be used in a dynamical form (changing the dynamics of the automaton). To preserve the deterministic property and one-to-one mapping, deletion of facts must be possible, which is achieved by the function `(db-delete-facts form &optional (db database))`. Upon the addition of facts, forward chaining is feasible in case the fact is new. The answer to that requirement is achieved by the use of the function `(add-fact fact (facts-db) (rules-db))` which has the potential of invoking the forward chaining on the fact added. In general, it is wise to complete deletions prior to the addition of the new facts.

Since expert systems are potentially capable of processing the symbolic representation defined above, the need for rules about how to process the knowledge base axioms becomes essential. The objectives of the logic observer are to assess convergence to singletons through unification, and to provide an estimation when given the observations. The current dynamical observer can be represented as a query of *what is the estimate of the current state given the sequence of observations*, and answering the query will solve the problem of the logic observer, but what is expected of the dynamical logic observer is more than providing an estimate. In such a case, the dynamical classical observer showed some efficiency, as for example the classical observer would estimate the observability question from the automaton dynamics without having the observations which are achieved by generating the tree and without applying the theorems that relate the observability [3] to the depth of the tree (see [2]). Such capabilities of the classical observer became themselves minimum requirements for the logic observer to generate information on the observability status of the automaton once the knowledge base of the dynamics of the machine were defined.

Since the axioms that define the automaton transition and output functions can be identified by key-pad descriptions within their specific databases, we can use an inference engine with no preset goal which requires a forward chaining rather than backward chaining. In such a case, comparison with the classical observer becomes reasonable. However, artificial intelligence can contribute to solving more complex questions. Since it is a matter of rules that interact in forward chaining, the output of the inference can be expressed and preserved as a set of axioms and can define the `CDO-Next` database as mentioned in the previous section. Thus, new features such as updating the inference knowledge database become powerful when changes occur in the dynamical structure of the automaton.

Much attention and care should be taken in the definition of rules that would control the forward chaining. The idea is to keep two distinct databases of rules where the first set would be used in case the inference is done for the first time, and the second set would be used for the dynamics updates of the machine. The forward chaining rules are entered using a form such as the following:

```
(defrule  rule-name
   (<= condition &rest consequences)).
```

In the next section, *Simulating an Observer*, descriptions of the rules that generate the `CDO-Next` axioms are presented as they concern the simulation of the observer rather than the preparation of the simulation. The latter consists of a permanent base for the complete simulation and hence should be kept separated.

Upon the simulation of the logic observer as will be shown later, the concept of having transformation in the dynamics of the automaton will influence the design of the logic system environment. Rules which are already defined can be added, modified or removed. Hence the databases (transition, output and rules) are verified. Using the given assertion pattern as a key, a changed fact searches through all of the

observations.

[3] Observability of an automaton is linked to the convergence of the tree branches to singletons.

forward chaining rules for a set whose conditions might resolve with the new fact (changed fact). For all of those which return valid binding frames [4] when the condition is treated as a query into the database, assertion of each of the facts in the consequences occurs after substituting for the variable bindings imposed by the conditions. Since this process is needed for every assertion in the database and potentially results in further assertions, it may generate a chain of such inferences.

Different functions were written specifically to provide an adequate support for the major routines. Those functions which are considered general purpose functions also facilitated the design of the classical observer.

Simulating an Observer

Since the encoding of the automaton depends on writing the transition and output functions in axioms which are entered as facts in specific format, and handled by a database, and since the environment built up provides an inference engine based on forward chaining which applies rules to the existing axioms of facts, the problem of simulating the observer becomes a question of formulating proper rules.

In the previous section, the format of defining rules (entering) was presented. The function "`defrule`" requires initially a name for the rule followed by the binding conditions and finally by the consequences. For example:

```
(defrule CDO-Next
 (<= (and
      (Emit ?state ?output)
      (NextStates ?initial ?input ?state)))
 (CDO-Next ?initial ?input ?output ?state))
```

`CDO-Next` was chosen as a name for the rule that basically generates the CDO-Next facts. The `and` is used to test the binding conditions. `(Emit ?state ?output)` [5] will return all the output function axioms.

This simple rule is considered the most important rule in representing the logic observer. It defines the search path of the forward propagation by (i) taking all the possible observation pairs of `input` and output and then (ii) providing all the Current Dynamical Observer Next (CDO-Next) states. In the above rule, `Emit` and `NextStates` appear as a pair of the knowledge base. The facts `CDO-Next` are defined as the consequence of the inference and Table 2 shows the output of such an inference in case the $\mathcal{M}_7$ is considered.

This example of a rule shows the flexibility of the logic observer and is partially controlled by the definition of the rule. For example if the query is required for one specific output e.g. `b`, the binding list `(Emit ?state b)` will be used and the table above would shrink to numbers 2 and 8.

1	(CDO (a1 a2 a3 b1 b2 b3 c1)	a	(a1 a2 a3))
2	(CDO (a1 a2 a3 b1 b2 b3,c1)	b	(b1 b2 b3))
3	(CDO (a1 a2 a3 b1 b2 b3 c1)	c	(c1))
$4_{(1)}$	(CDO (a1 a2 a3)	u,a	(-))
$5_{(1)}$	(CDO (a1 a2 a3)	u,b	(b1 b2 b3))
$6_{(1)}$	(CDO (a1 a2 a3)	u,c	(-))
$7_{(2)}$	(CDO (b1 b2 b3)	u,a	(a2))
$8_{(2)}$	(CDO (b1 b2 b3)	u,b	(b3))
$9_{(2)}$	(CDO (b1 b2 b3)	u,c	(c1))

Table 2: Simulation output of the $\mathcal{M}_7$ machine.

From the practical side, the question of efficiency in the simulation can become important. The application of large numbers of states and complex dynamics in the transition and output functions would attract the attention to the run-time cost in the estimation. This approach becomes necessary in the section *Applications of Dynamical Logic Observers* where the dynamical behavior is involved and the online estimation is required.

Returning to the definition of an automaton given at the beginning of this paper, let $\mathcal{M} = (X, U, Y, \Phi_i, \eta_i)$ be an automaton and let i be the index of the discrete event corresponding to the transformation of the transition and output function. In other words, let the automaton start with an initial transition function Φ_0 and output function η_0. Assuming that the automaton dynamics change to Φ_1 and η_1, the existing current and initial state estimation become invalid and thus a new estimation procedure is required employing the appropriate new system model.

Looking at the $\mathcal{M}_7$ automaton introduced in figure 1, the choice of different transformations of the automaton transition function which retain observability is wide. Changing the `(NextStates b2` Φ_7 `a2)` transition in the $\mathcal{M}_7$ to `(NextStates b2` Φ_7 `b1)` would provide an adequate example. Such a procedure can be applied differently using the environment introduced above and it could be made operational via the assignment procedure:

```
(defun db-change-fact (old-fact new-fact)
  (db-delete-fact old-fact transition-db)
  (add-fact new-fact transition-db rule-db))

(db-change-fact
  '(NextStates b2 $\Phi_7$ a2)
  '(NextStates b2 $\Phi_7$ b1))
```

Updating the facts of the databases, `add-fact` will automatically forward chain on the new fact. `db-add-fact` can be used to avoid the query.

Modifications of the automaton dynamics might lead to changes in the propagation tree. The re-estimation of the current (or initial)

[4] New facts generated.

[5] For example, the query `(Emit ?state a)` in the $\mathcal{M}_7$ automaton will return `a1 a2 a3` states.

state of the modified automaton requires a new **CDO-Next** database from which the propagation tree is deduced. Since the inference is based on forward propagation as mentioned previously, it becomes advantageous to use the forward chaining separately on the changed facts without generating a new **CDO-Next** database. For the modification on the $\mathcal{M}_7$ automaton mentioned above, forward chaining on the deleted fact provides the list of axioms to be deleted from the **CDO-Next** database and forward chaining again on new facts provides the list of axioms to be added[6]. The following table presents the modified **CDO-Next** axioms compared to the previous table.

	1	(CDO UNIVERSE	a	(a1 a2 a3))
	2	(CDO UNIVERSE	b	(b1 b2 b3))→7,8,9
	3	(CDO UNIVERSE	c	(c1))
	$4_{(1)}$	(CDO (a1 a2 a3)	u,a	(-))
	$5_{(1)}$	(CDO (a1 a2 a3)	u,b	(b1 b2 b3))→7,8,9
	$6_{(1)}$	(CDO (a1 a2 a3)	u,c	(-))
2→	$7_{(2)}$	(CDO (b1 b2 b3)	u,a	(a2))
2→	$8_{(2)}$	(CDO (b1 b2 b3)	u,b	(b1,b3))→10,11,12
2→	$9_{(2)}$	(CDO (b1 b2 b3)	u,c	(c1))
8→	$10_{(8)}$	(CDO (b1,b3)	u,a	(-))
8→	$11_{(8)}$	(CDO (b1,b3)	u,a	(b3))
8→	$12_{(8)}$	(CDO (b1,b3)	u,a	(c1))

Table 3: Partial modification of the **CDO-Next** database. Notice that, practically, $4_{(1)}$, $6_{(1)}$ and $8_{(10)}$ are discarded. **UNIVERSE** stands for the list (a1 a2 a3 b1 b2 b3 c1).

Since the modification in the dynamics is related to the state (b2), the inference starts with line 2 from the table above and generates lines $7, 8, 9$ and since line 8 is not a singleton, the forward chaining continues and generates lines $10, 11, 12$. As expected the process stops until singletons are generated.

The difference between the logic observer and the classical observer is seen with large numbers of states, assuming the variation of the automaton is minor. In fact the comparison should be achieved with the creation of the set estimate propagation tree, which is what the classical observer generates, and that requires the logic observer to be assembled from the generation of the Current Dynamical Observer database. The deduction of the tree from the CDO database is straightforward and is achieved in one iteration.

OPTIMIZATION USING THE LOGIC OBSERVER

One aspect related to the simulation of the observer is the association of the number of controls in the transition function with the maximum width of the propagation tree. Another aspect relates the depth of the automaton to its observability [5], and demonstrates that the automaton converges in less than N^2 levels in observable cases where N is the number of controls.

Limiting the Automaton Functions

In [5], the authors refer to the maximum number of controls for an automaton as of the order of 2^N. The following is a supportive example to show the exact number of the possible controls for a specific automaton.

Recall the definition of the transition function Φ to be $X \times U \rightarrow X$ where X is a finite set of states. The combination of singletons, doubles, etc. will form a new finite set. The mapping of every element of the new set to all the elements will construct a table. Since the output function and the transition functions are independent, it is arbitrary to choose the output function in such a way as to show all the controls in the table introduced above. Assuming, for example, a three state (a1 a2 a3) automaton, and choosing the output function with no initial input to the machine, the observer returns the whole set (a1 a2 a3). Denoting the transition u with a subscript number as shown in table 1.1., the problem can then be stated as how many *distinct* controls can the table accept.

Table 1.1 relates the controls of the transition function to the possible subset combination (excluding the empty set) of the state space (a1 a2 a3). What this table shows is the propagation possibility of every set in every row. For example, the first line, interpreted as (a1 a2 a3), will generate the sub-set pair (a1 a2) under the control u_1, (a1 a3) under u_2 etc. until all possible combinations of $(N-1, ...1)$ sets. Notice that N is not included for the reason that observability is violated when a set (not singletons) generates itself. These facts are shown in the table and marked with a dash when necessary.

	(a1 a2 a3)	(a1 a2)	(a1 a3)	(a2 a3)
(a1 a2)	u_1	-		
(a1 a3)	u_2		-	
(a2 a3)	u_3			-
(a1)	u_4			
(a2)	u_5			
(a3)	u_6			

Table 4: Mapping of a three state automaton assuming one possible output.

From Table 4, the maximum distinct controls that could be allocated for the first row is six and in general $2^N - 2$ (see footnote [7]).

Such allocation defines the transition function mapping in a particular fashion and any

[6] Practically, there is no need to forward chain on the old fact as an overwrite procedure handles the deletion.

[7] The general case can be deduced using the same mapping argument for N states. Output function with more than one possibility can be displayed in parallel tables, one table per output.

additional distinct controls claimed for the second, third, and last rows will violate the uniqueness of the existing controls. For example in Table 1.1, the set `(a1,a3)` generates the singleton `(a1)` under the control u_7. Since the set `(a1,a3)` is included in `(a1,a2,a3)`, this will make the latter set generate the same singleton under the control u_7 which violates the uniqueness of the existing control u_6.

	(a1 a2 a3)	(a1 a2)	(a1 a3)	(a2 a3)
(a1 a2)	u_1	-		
(a1 a3)	u_2		-	
(a2 a3)	u_3			-
(a1)		u_4		
(a2)			u_5	
(a3)				u_6

Table 5: Mapping of a three state automaton with one possible output.

Table 5 shows a different assignment of the controls, still with the purpose of finding the table with maximum unique controls. And as stated before, any additional control will violate the uniqueness. The allocation of controls u_4, u_5, u_6 is done arbitrarily for the singletons and can be related to the control of the tree propagation expansion.

Propagation Width for the $\mathcal{M}_7$ Automaton

Different approaches were tried with the purpose of solving the question of binding the propagation width. Starting by considering the maximum number of controls in an automaton to be of the order of 2^N, it is logical to believe that, in the worst case, the maximum width is of the order of N^N where again N is the number of states.

This section lists the reasoning for finding the maximum width of the $\mathcal{M}_7$ with seven states, three outputs and two transitions. Since the question is that of searching for the maximum width w^* in the propagation tree, it is obvious that the transition and output functions constitute complete information for the determination of w^*. Initially, the automaton is considered to have the same output function mapping as stated in the beginning of this document which follows that `(a1 a2 a3)` will emit `a`, `(b1 b2 b3)` will emit `b` and `(c1)` will emit `c`. However, one can assume a new transition function where α and β are the two operating controls. Looking at the dynamical nature of the automaton, some knowledge and rules can be deduced from the output function dynamics before considering the controls. (e.g. if the dimension of the output set equals the dimension of the state set, the estimation question can rely only on the observer output). Initially, *level 0* is always constant and is the set of all states `(a1 a2 a3 b1 b2 b3 c1)`. Knowing that at no input does the automaton emit an output before absorbing a control, then *level 1* is deduced in the form:

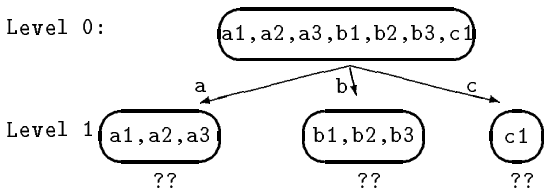

Figure 3: The $\mathcal{M}_7$ automaton. States, transitions and output

What is shown above is *independent* of the transition functions and defines a part of the characteristic of the automaton with respect to the output function and always constitutes the first set of facts of the `CDO-Next` database. The question is raised as to what is the maximum width on the second level of the propagation tree, or, that is to say, the sum of the sets of singletons, doubles and triples forming level 2.

Since the number of available controls and outputs are two and three respectively, the possible combinations of observations is limited to six (α `a`, α `b`,α `c`,β `a`,β `b`, β `c`). Looking at level 1, it becomes obvious that the maximum possible width of the second level would be thirteen. The last few statements mentioned before lead to the concept of the use of a backward chaining inference engine to find the dynamics of an automaton that suits the goals of the levels required. To complete the success of this search, the observability status of the proposed automaton that would generate the mentioned levels has to be evaluated through the use of the logic observer. This technique was carried out on this seven states automaton (with two controls and three outputs) and the maximum width found for the second level was eleven.

A similar exhaustive search should be achieved on higher levels in the tree which will lead to a higher number of transition function maps. This search should provide an answer to the problem of finding a limit to the width of the tree.

CONCLUSIONS

To briefly summarize, in this paper the simulation of the Dynamical Logic Observer has been proposed to contrast with the classical observer which was introduced in the beginning of the paper. The *logic observer* in this work started by the encoding of the input-state-output machine into sets of axioms which modify a given automaton in real time by handling the axioms in a systematic fashion to preserve the required

dynamics. Once the encoding is completed as an initial step, the logic observer forward propagates on the existing axioms and generates a permanent database of facts from which the tree is deduced.

The last section of this paper tries to set an approach for optimizing the behavior of the functions of the machines to acquire pre-set knowledge cases. Upon finding the maximum width of the tree for a given machine, the idea suggested was to define, on any given level, a certain width and then to propagate backward in search of the possible function that satisfies that goal.

REFERENCES

[1] Peter E. Caines, Russell Greiner, and Suning Wang. Dynamical Logic Observers for Finite Automata. In *Proceeding of the 27th IEEE Conference on Decision and Control*, pages 226-233, Austin, Texas, December 1988. (Also presented at the NASA-Ames workshop on Artificial Intelligence and Discrete Event Control Systems, June, 1988)

[2] Peter E. Caines, Suning Wang, and Russell Greiner. Dynamical Logic Observers for Finite Automata. In *Proceeding of the 1988 Conference on Sciences and Systems*, pages 50-56, Princeton University, Princeton NJ, March 1988

[3] Peter E. Caines and Suning Wang. Classical and Logic-Based Regulators for Partially Observed Automata: Dynamic Programming Formulation. In *Proceeding of the 1989 Conference on Information Sciences and Systems*, John Hopkins University, Baltimore, MA, March 1989.

[4] Peter E. Caines and Suning Wang. Classical and Logic-Based Regulators Design and Its Complexity. In *Proceedings of the 28th IEEE Conference on Decision and Control*, Tampa, Florida, December 1989.

[5] Michael R. Genesereth and Nils J. Nilsson. Logical Foundations of Artificial Intelligence. *Morgan Kaufmann Publishers, Inc., Los Altos, CA,* 1987.

[6] Arthur Gill. Introduction to the Theory of Finite State Machines. *New-York, McGraw Hill*, 1962.

[7] Jonathan S. Ostroff. Real Time Computer Control of Discrete Systems Modeled by Extended Machines: A Temporal Logic Approach. *Ph.D. thesis.* University of Toronto, 1987.

[8] Peter Ramadge and Murray Wonham. The control of Discrete Event Systems. In *Proceedings IEEE, Vol 77, No. 1*, pp 81-98, 1989.

[9] Jonathan S. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM, 12 (1)*, pp 23-24, 1965.