

Searching for Optimal Homing Sequences for Testing Timed Communication Protocols

Ariel Stulman

Jerusalem College of Technology / Dept. of Computer Science, Jerusalem, Israel

Email: stulman@jct.ac.il

Abstract – With the eruption of communication technology and its supportive protocols, the field of testing of timed-IUT (Implementation Under Test with time constraints) was quick to follow. New and extended test generation methods using timed automata were in the need. Many papers were written on a broad range of timed automata testing. Few papers, however, dealt with generation of state identification sequences problems. Those who did, keyed their algorithms towards transformation of a timed-IUT into a regular, un-timed IUT; thus, avoiding the need of redefining well known generation algorithms. This method, however, only functions if we search for optimal sequences w.r.t. their lengths. Optimality w.r.t. execution time was not considered in this context. In this paper we present a search algorithm for the direct generation of timed homing sequences optimal w.r.t. to execution time. This is accomplished by expanding and modifying the search tree used in previous algorithms such that it contains the capability of satisfying our goal. The solution found is proven to be optimal with respect to execution time. A comparison with other related algorithms is presented.

Index Terms – homing sequence, state identification, timed automata, testing

I. INTRODUCTION

Motivated mainly by automata theory, the field of program testing was heavily studied many years ago. Kohavi's book [1] gives a good exposition of the major results. During the 80's the topic mostly died down, only to come up again due to its application in the field of communication protocol testing. The abundant and diverse research in the area motivated many algorithms and methods that attempted to optimize the testing procedure (status messages [2], separating family of sequences, distinguishing sequences [3], UIO sequences [4], [5], characterizing sequences [6], [1], [7], and identifying sequences [1]). A survey of the main methods can be found in [8].

Due to the wide use of finite state machines (FSM) as the modeling technique for an implementation under test (IUT), its inherent flaw was automatically projected into system testing. Standard FSMs do not take into account temporal constraints that may influence the IUT; and as such, most methods developed for system testing were not applicable to real-time systems. With the proposal of Alur and Dill's 'Timed automata' [9] in 1994, an entirely new field of

system testing for real-time (reactive) systems emerged. Quickly, it became a thriving field with many papers published or currently published on a wide variety of sub-topics ([10], [11], [12], [13], [14], [15], [16], [17] and many others).

Regardless of the many topics under scrutiny, none of them dealt with some of the more famous problems that exist within the field of protocol testing; specifically, the homing problem, a sub-problem of the state identification class. Even the papers that did consider this class of problems, their main objective was in demonstrating how a t-IUT can be transformed into a regular, un-timed IUT, for which all necessary algorithms pre-exist (see [10], [18], [19], [13] and others). A few papers did provide algorithms for direct generation of timed homing sequences ([20], [21]). There, however, as is done in many other papers (see for example [14]) the notion of defining optimality as a function of the sequence length (as was done with un-timed machines) is retained. It would seem that when real-time systems are in question, optimality of sequences should be defined in terms of execution time.

As we are un-aware of an algorithm for the generation of optimal homing sequences w.r.t. execution time, the main contribution of this paper is in presentation of such an algorithm while conforming to the more intuitive definition for optimality. We do so while retaining the direct generation algorithm given in [20]. In a nutshell, the type of search algorithm given in [1] is changed to accommodate for execution time optimality.

Section II introduces the basic definitions that pertain to this discussion, and formulates the problem needing a solution. Section III provides the crux of the current work, with a toy example to demonstrate the solution. Assessment and concluding remarks are given in sections IV and V, respectively.

II. PRELIMINARIES

A. I/O Automata

The majority of the testing techniques discussed in the literature are based on a variant FSM: the famous *Mealy machine* [22]. This fact is based upon the ability of the Mealy machine to exchange messages (input and output) with its environment. A deterministic transition is stimulated by input from the environment, and as a

consequence, the machine can return an output message back to the environment (a *reactive machine*). Since a *black box* testing model – where one cannot see the internal structure of the IUT but has access to its input and output ports – is the basic premise of checking experiments (see [23],[14]), the Mealy machine is a sufficient candidate for representing the internal logic of an IUT.

Definition 1 (Mealy Machine). A *Mealy machine*, M , is a 6-tuple $\langle S, s_0, I, O, \lambda, \delta \rangle$, where:

- S is a finite set of states.
- $s_0 \in S$ is the initial state of the system.
- I is a finite set of input events ($I = \{t_1, t_2, \dots, t_p\}$).
- O is a finite set of output events ($O = \{o_1, o_2, \dots, o_r\}$).
- λ is the state transfer-function ($\lambda: S \times I \rightarrow S$).
- δ is the output function ($\delta: S \times I \rightarrow O$).

For simplicity we extend the state transfer function, λ , from single input symbols to input strings as follows: for some initial state s_1 , let the input sequence $\sigma = \alpha_1 \alpha_2 \dots \alpha_k$ take the machine successively through the states $s_{j+1} = \lambda(s_j, \alpha_j)$, where $j = 1, 2, \dots, k$, such that the final state of $\lambda(s_1, \sigma) = s_{k+1}$. In the same manner we extend the output function, δ , from a single output to an output string such that: $\delta(s_1, \sigma) = \beta_1 \beta_2 \dots \beta_k$, where $\beta_j = \delta(s_j, \alpha_j)$ with $j = 1, 2, \dots, k$. Obviously, $s_i \in S$, $\alpha_j \in I$ and $\beta_j \in O$.

B. State Uncertainty

It may be the case that some state information of the IUT is missing. The *state uncertainty* of the IUT is defined as a multi-set of states that may adequately complete the missing state information [1]¹. If the initial state of the IUT is unknown, we speak of an *initial state uncertainty*; a set containing all possible states that may constitute the IUT's initial state. For example, consider the automata in Fig. 1 that represents the internal logic of an IUT implementing the alternating bit protocol (sender side). If the machine can begin in any of the four states, we say that the *initial state uncertainty* is $\{ABCD\}$.

When the current state of the IUT is unknown, we speak of a *current state uncertainty* multi-set. Suppose, for example, that $\sigma = \text{ack1}$. After inputting of σ into the IUT in Fig. 1, $\{\{AC\}_q, \{B\}_{m0}\}$ ² is the *current state uncertainty*.

A *homogenous state uncertainty* is defined as an uncertainty multi-set in which all sub-sets contain a singleton.

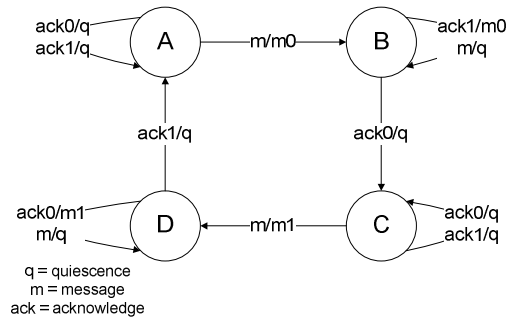


Figure 1. FSM representation of the Alternating Bit Protocol (sender side)

C. Successor Tree

A *successor tree* is a tree representing the states that can be reached by the IUT based on all possible input combinations. The purpose of the tree is to graphically display the n^{th} successors of the root; aiding the experimenter in the selection of the most suitable input sequence to meet his needs.

The root of the tree contains the initial state of the IUT. When it is not known, we associate with the root an *initial state uncertainty* vector. Tree edges represent input to the IUT. Every node is associated with a *current state uncertainty* vector representing accumulated state uncertainty knowledge until that point; with σ being the concatenation of labels on the edges that form the path from the root to the node. For example, a partially extended successor tree for the IUT in Fig. 1 with an initial state uncertainty of $\{ABCD\}$ is shown in Fig. 2².

Since the *degree* of the tree is $|I|$, the number of acceptable inputs in the language of the IUT, at level j ($0 \leq j \leq \infty$) we may have $|I|^j$ nodes. It is quite obvious that in order to reduce the size of the successor tree, some restrictions must be placed (avoid redundancy, etc.).

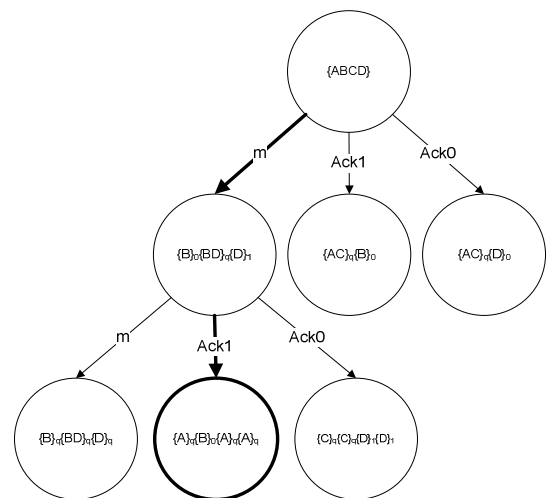


Figure 2. Partially extended successor tree of the IUT in Fig. 1

¹ In the field of AI it is termed "belief state" (see [24] page 84).

² We split the current state uncertainty into multiple sub-sets since we can determine which sub-set must contain the actual current state based on the IUT's output. The output itself is shown as a subscript after the set; using 'q' to depict quiescence.

D. Timed I/O Automata

A *timed automaton* is a Mealy machine with the addition of temporal constraints on the transition function λ . A transition cannot fire, and hence no output or internal state change can occur, if the clock guard on the transition is not satisfied. A clock or set of clocks must be included within the system to allow for the definition of time.

Definition 2 (Clock Constraint). Let a *clock constraint*, Δ , over a set of clocks, C , be defined as a Boolean expression of the form $x \text{ op } z$, where $x \in C$, op is a classical relational operator ($=, \leq, \geq, >, <, \neq$), and z is a mathematical expression composed of integer constants and/or clock values and evaluates to an integer.

Definition 3 (Clock Guard). Let a *clock guard* over C be a conjunction of clock constraints over C : ($\text{clock_guard} = \Delta_1 \wedge \Delta_2 \wedge \dots \wedge \Delta_m$). Let ψ be the set of possible clock guards pertaining to the system.

Definition 4 (Timed I/O Automata). A *timed i/o automata*, TA, is a 7-tuple $\langle S, s_0, I, O, C, \lambda, \delta \rangle$, where:

- S is a finite set of states.
- $s_0 \in S$ is the initial state of the system.
- I is a finite set of input events ($I = \{i_1, i_2, \dots, i_p\}$).
- O is a finite set of output events ($O = \{o_1, o_2, \dots, o_r\}$).
- C is a finite set of clocks.
- λ is the state transfer function ($\lambda: S \times I \times \psi \rightarrow S$).
- δ is the output function ($\delta: S \times I \times \psi \rightarrow O$).

Assumption 1. For simplicity, we assume that if TA receives $i_k \in I$ at instant t , it will also output the corresponding $o_b \in O$ at that exact instant. [Justification: Within the class of protocol testing, in essence, output is produced on a transition generated by the input. Reducing transition time to be infinitely small, we may consider the input and output as occurring at the same instant.³]

Assumption 2. For explanation purposes, we assume that the TA contains a single clock, and the *clock guard* on a transition is composed of at most one *clock constraint*. (This assumption will be lifted later).

E. Testing Environment

The need for testing arises due to the lack of information about an IUT. Normally, testing is performed using a *black box model* in which we do not have access to the internal structure of the IUT. This requires that the deduction of missing information be accomplished by means of inputting messages to the machine (testing sequence). Extracting a homing sequence requires the information be inferred from the outputs the machine generated in response to the input.

³ This is a classical assumption in the field, see for example [16].

Assumption 3. In the current context, testing is performed on a *fully specified*⁴, *strongly connected*⁵ and *reduced*⁶ IUT or t-IUT.

F. Homing Sequence – Problem and Solution

In order to begin execution of any test sequence (specifically conformance testing), we must bring the IUT to some identified state from which we know how to proceed. The state must be unequivocally identified, regardless of the current state of the IUT. Based upon the output produced in response to a specific input sequence, we wish to reach a definite conclusion as to meet the goal at hand. It is this sequence that allows for a solution to the homing sequence problem.

Definition 5 (Homing Sequence). An input sequence, σ_{hs_k} , is said to be a *homing sequence* (HS) of machine M, if the final state of M can be determined uniquely from M's response to σ_{hs_k} , regardless of the initial state [1]. Let σ_{hs} be the set of all HSS accepted by M as a solution to the homing problem; thus, $\sigma_{hs_k} \in \sigma_{hs}$.

Definition 6 (Optimal Homing Sequence). We classify an HS for machine M as *optimal*, $\sigma_{hs_{op}}$, if for all HSS, σ_{hs_k} , accepted by M, $\sigma_{hs_{op}}$ is shortest. i.e., $\left\{ \left| \sigma_{hs_{op}} \right| \leq \left| \sigma_{hs_k} \right| \forall \sigma_{hs_k} \in \sigma_{hs} \right\}$.

F.1 Homing Tree (Optimal Homing Sequence)

The homing sequence problem was completely solved in [1]. A search is conducted on the successor tree with an initial uncertainty containing all states using the famous BFS while avoiding repeated states (see [24] section 3.5):

Definition 7 (Homing Tree). A *homing tree* (HT) is a successor tree in which we deem a node as terminal (leaf) when one of the following occurs:

1. **The loop rule:** The *current uncertainty* associated with the node was already associated with a node in a preceding level.
2. **The redundancy rule:** The *current uncertainty* associated with a node is also associated with other nodes on the same level. One is chosen as a non-terminal candidate for future expansion of the tree; the others deemed terminal.

⁴ There exists a definition for each state, $s_j \in S$, and every input, $\alpha_k \in I$; i.e.: $\lambda(s_j, \alpha_k)$ and $\delta(s_j, \alpha_k)$ are defined for s_j in S and α_k in I.

⁵ For every pair of states, $s_i, s_j \in S$, there exists an input sequence, σ_{ij} , which takes the IUT from s_i to s_j ; i.e. $\lambda(s_i, \sigma_{ij}) = s_j$.

⁶ For every pair of states, $s_i, s_j \in S$, there exists an input sequence, σ , which distinguishes them; i.e.: $\lambda(s_i, \sigma) \neq \lambda(s_j, \sigma)$ or $\delta(s_i, \sigma) \neq \delta(s_j, \sigma)$ for some σ .

The HS is constructed by concatenating the labels on the edges of the HT path leading from the initial uncertainty (root) to the *first* node found satisfying the search criterion.

Intuitively, the *length* of the HS, $|\sigma_{hs_k}|$, is equal to the depth of the solution node within the HT. It has been proven that using a BFS when all step costs are equal is optimal and complete [24]. Since time isn't a factor, and only a goal node is required, $|\sigma_{hs_k}| = |\sigma_{hs_{op}}|$.

F.2 Examples for HT

Suppose the successor tree of Fig. 2 is a HT for the automaton in Fig. 1. There are two nodes at the second level of the HT that are homogenous uncertainties. We may stop our search; we found two HSS ($\sigma_{hs_1} = 00$ or $\sigma_{hs_2} = 01$) that uniquely identify a final state based on the output regardless of the initial state. In Table 1 we show the output and deduced final state for $\sigma_{hs_1} = 00$, the first optimal HS found.

TABLE 1. IUT output for $\sigma_{hs_1} = 00$

Initial State	IUT response to $\sigma_{hs_1} = 00$	Final State
A	01	A
B	01	A
C	10	C
D	00	C

G. Problem Formulation

The problem we face when dealing with an IUT with timing constraints (t-IUT) is essentially the same; mainly, to bring the t-IUT to some positively identified state, optimally. Here too, we must accomplish the task merely by means of inputting messages to the t-IUT and examining its output (*black box model*).

Dealing with a t-IUT, however, introduces some complexity. We must take into account the timing constraints that restrict transitions for two reasons. Firstly, some transitions may be applicable only at specific times and inapplicable at others. Secondly, we must not introduce non-determinism into the model. Consider Fig. 3 in which a portion of a t-IUT is shown. If the uncertainty associated with a node in the HT contains both A and B (as an *initial state uncertainty* certainly will), simply labeling an edge in the HT with 0 is insufficient. If the current node is A, we must also consider compliance with the constraint $[c < 6]$. Adding the constraint $0[c < 6]$ to the edge, however, will not suffice. Since the actual current node might be B, such a transfer constraint will be un-deterministic (if $c=2$ we will go to node C, while if $c=4$ we will go to node D).

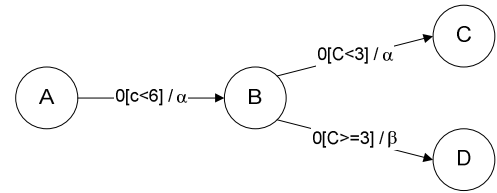


Figure 3. Portion of a t-IUT

III. HOMING SEQUENCE FOR TIMED-IUT – PROBLEM AND SOLUTION

A. Previous Work

As mentioned earlier, previous work dealt mainly with the transformation of t-IUTs into regular IUTs for which solutions pre-exist ([10], [18], [19], [13] and others). Only [20] provided an algorithm for the direct generation of homing sequences for timed-IUTs. The main idea was of an adaptation of the HT used for the extraction of optimal homing sequences to a timed-homing tree which introduced the timing parameters into the tree itself.

The drawback with the above algorithm, however, was the use of minimal sequence length as the definition of optimality. The basic premise was "let us solve the original problem, but this time for t-IUTs". It seems, however, that optimality for t-IUTs should be defined in terms of execution time. Due to time constraint attached to inputs, equal length sequences do not necessarily complete within the same amount of time. It is quite possible that longer sequences complete faster than shorter ones. Does it really matter if we have fewer inputs if the total time for execution is elongated?

B. Optimal t-HS w.r.t. Execution Time

Definition 8 (Timed- Homing Sequence). A timed-homing sequence (t-HS), σ_{t-hs_m} , is a HS that includes timing of input. σ_{t-hs_m} is of the form $\alpha_1[\psi_1].\alpha_2[\psi_2].....\alpha_k[\psi_k]$; where, $\alpha_j \in I$ and $\psi_q \in \Psi$, the timing constraint for that input. Let σ_{t-hs} be the set of all timed HSS accepted by a t-IUT as a solution to the homing problem; thus, $\sigma_{t-hs_m} \in \sigma_{t-hs}$.

Definition 9 (Sequence Execution time). The execution time of a sequence, π_σ , is determined by the *minimum*⁷ time that must elapse before a sequence can be inputted in its entirety.

⁷ We used the term *minimum* because constraints can be composed of inequalities that allow for flexibility of input synchronization. When calculating the execution time, we attempt to enter input syllabi as quickly as possible; hence, *minimum* time.

Definition 10 (Optimal timed-homing sequence). A timed homing sequence, $\sigma_{t-hs_{op}}$, is considered optimal w.r.t. execution time if $\left\{ \pi_{\sigma_{t-hs_{op}}} \leq \pi_{\sigma_{t-hs_m}} \mid \forall \sigma_{t-hs_m} \in \sigma_{t-hs} \right\}$.

For example, the homing sequence $\sigma_{t-hs_1} = \alpha_1[c < 2]\alpha_2[c \geq 7]$ must allow for $c \in C$ to equal 7 before it can terminate ($\pi_{\sigma_{t-hs_1}} = 7$). A second solution sequence, $\sigma_{t-hs_2} = \alpha_1[c = 1]\alpha_2[c < 3]\alpha_3[c \geq 3]\alpha_4[c = 6]$, must terminate when $c \in C$ reaches 6 ($\pi_{\sigma_{t-hs_2}} = 6$). Under the current definition of optimality, σ_{t-hs_2} is optimal even-though $|\sigma_{t-hs_2}| > |\sigma_{t-hs_1}|$.

B.1 Timed Homing Tree (extraction of optimal homing sequences)

Unlike homing sequences deemed optimal based on their lengths, where a node's height within the HT represented shorter (hence, optimal) sequences (see section II.F.1 and [20]), optimality based on execution time requires major adaptation of the uncertainty tree and extraction algorithm. It is no longer a tree with equal cost on its branches. Rather, we must use a uniform cost search, expanding the node with the lowest path (time) cost. In addition, the rules governing efficiency vis-à-vis the tree pruning mechanism must also change to tolerate future possible run-time developments:

Definition 11 (timed successor tree). A timed successor tree is a successor tree with the addition of time constraints on its edges [20].

Similarly to their task in regular successor trees, nodes in a timed successor tree represent the state of the t-IUT. Edges, representing the stimulation of the t-IUT through input, are marked with an input literal and paired with an associated time constraint. To achieve t-IUT stimulation, literals must be inserted in compliance with the associated time constraint. For example, an edge labeled $0[c=3]$ would represent the input of 0 when clock $c=3$, while a label of $1[c<17]$ would allow for the input of 1 anytime before $c=17$.

In order to allow for the comparison of homing sequences represented by two nodes in the tree vis-à-vis their execution time, we must allow them knowledge of their cumulative run-time information. Let us denote the amount of time that we must allow for the descending of a timed successor tree from node j to node k while conforming to the timing constraints on the tree's edges as $\tau_{j \rightarrow k}$. We will classify a node as having higher selection priority value, $\rho_j > \rho_k$, iff $\tau_{root \rightarrow j} < \tau_{root \rightarrow k}$.

Now we can develop the timed homing tree suited for our needs:

Definition 12 (timed-homing tree). A timed homing tree tuned for the extraction of execution time optimal sequences (t-HT), is a timed successor tree pruned for efficiency by one of two rules:

1. **The loop rule:** the *current state uncertainty* associated with a node is also associated with a node with a higher *priority* value⁸.
2. **The redundancy rule:** the *current state uncertainty* and *priority* associated with a node are associated with other non-terminal nodes as well (regardless of their level in the tree). Arbitrarily⁹, one is selected for future tree extension; the others terminated.

For expansion of the t-HT until a possible solution, we use the algorithm described in the following pseudo-code (comments are inserted for better explanation):

```

/* The function TimedHSExtraction returns a timed
homing sequence, timedHS, of the form
 $\alpha_1[\Psi_1], \alpha_2[\Psi_2], \dots, \alpha_k[\Psi_k]$ , for the t-IUT passed as
its first parameter. The second parameter is a
set containing the initial state uncertainty -
does not necessarily contain all states of the t-
IUT.
Note: the function is written in pseudo-C++ code
style. */
timedHS TimedHSExtraction (const directedGraph t-
IUT, set initialUncertainty)
{
    /* Create a search tree based upon a
SuccessorTree. Object constructor also extracts
ranges from the t-IUT and places them within
ranges, a vector data member. */
    SuccessorTree rt_homing_tree (t-IUT,
initialUncertainty);
    node *currentNode;
    do
    {
        /* expand is a SuccessorTree member function
that returns the next node for inspection
based on the priority of the node. Its
definition follows */
        currentNode = rt_homing_tree.expand();
    }
    /*end do/while loop when currentNode==NULL (no
nodes left for inspection) or when homogenous
uncertainty found*/
    while (currentNode && (currentNode->uncertainty
= HomogenousUncertainty));
    if (currentNode->uncertainty =
HomogenousUncertainty) //solution node found
        return rt_homing_tree.path(*currentNode);
    else
        return NULL; //sequence un-available
    } //end function TimedHSExtraction

/* The SuccessorTree member function expand
returns the next non-terminal node; where next is
defined by a highest priority first mechanism. If

```

⁸ It is possible that a node will become terminal even though when created it was not deemed so. This can occur the search finds a node on another sub-branch of the tree with the same *current state uncertainty* having a higher priority.

⁹ It is plausible that the wisest selection would be the node located highest in the tree. This would allow for the solution, given that it passes through one of these nodes, to be optimal with respect to its sequence length as well.

all nodes where deemed terminal, the function returns NULL. It is well known that a priority queue is the data structure of choice for achieving such a behavior. Thus, data member **priorityNodesQueue**, initialized in the object constructor to the root of the tree, is used within the function.

Note: the function is written in *pseudo-C++* code style. */

```
node* SuccessorTree::expand ( )
{
    /* An empty priorityNodesQueue implies that there aren't any nodes left in the tree that are non-terminal. priorityNodesQueue is a queue data member within SuccessorTree.*/
    if (priorityNodesQueue.isEmpty())
        return NULL;
    /* localRoot is a local variable used to point to the node within the tree that will next be inspected and expanded. Its value is extracted from the queue of non-terminal nodes waiting in priorityNodesQueue */
    node* localRoot=priorityNodesQueue.dequeue();
    /* data member ranges contains the time ranges extracted from the t-IUT to be placed on the edges to descendant nodes of localRoot. */
    for (int count=0; count<ranges.size();++count)
    {
        /* descendant is an array of pointers to descendant nodes within class node. For each range in ranges we create a new node, attach it to its parent node, and tell it, via its constructor, who the parent node is (this is useful when we need to extract the path of the node) and what the time constraint on its edge was - knowledge needed for deciding its priority.*/
        localRoot -> descendant [count] = new node (localRoot, ranges[count]);
        /* Member function goodCurrentUncertainty associates the correct uncertainty to the node passed as its parameter, and returns false if the node should be terminated; true otherwise.*/
        if ( goodCurrentUncertainty ( localRoot -> descendant [count] ) )
            /* Termination of a node simply requires it not be placed within the non-terminal nodes waiting for inspection. If termination is not required, the node is enqueued into priorityNodesQueue.*/
            priorityNodesQueue.enqueue(localRoot->descendant [count]);
    } // end for
    return localRoot;
} //end SuccessorTree member function expand
```

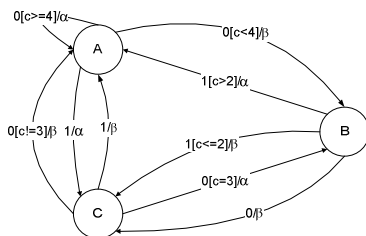


Figure 4. Description of a single clock IUT using an automaton

B.2 Example of t-HT

In Fig. 5 we show the t-HT¹⁰ for the timed automaton of Fig. 4. The t-HS extracted from the tree using the above algorithm is $\sigma_{t-hs_{op}} = 1[c \leq 2].1[c \leq 2]$ ¹¹; a t-HS clearly optimal.

In order to deduce the final state of the t-IUT after the sequence is completed, we must use Table 2.

TABLE 2. Final state of Fig. 4 after insertion of $\sigma_{t-hs_{op}}$

Initial State	t-IUTs output ¹² in response to $\sigma_{t-hs_{op}} = 1[c = 0].1[c = 1]$	Final State
A	a . β	A
B	β .a	C
C	β . β	A

B.3 Proof of optimality

In order to prove the timed homing sequence extracted using the above algorithm is optimal, we must first present two sub-theorems:

Theorem I: A descendant node necessarily has a lower priority than all of its ancestors ($\rho_{des} < \rho_{ans}$).

Proof: Let us assume that the path to node N_{des} (descendant node) passes through node N_{ans} (ancestor node). Time *must* elapse until we can reach N_{des} from N_{ans} . Namely, $\tau_{ans \rightarrow des} > 0$.¹³ If the time that must elapse in order to reach N_{ans} is $\tau_{root \rightarrow ans} > 0$, the total time needed to reach node N_{des} ($\tau_{root \rightarrow des}$) is $\tau_{root \rightarrow ans} + \tau_{ans \rightarrow des}$, a value obviously greater than

¹⁰ We inserted into the tree two nodes that would have been applicable if a dense-time model were used. Based on Assumption 2, however, the range $3 < c < 4$ is an impossibility; thus, we labeled them as not relevant (NR).

¹¹ Examination of Fig. 5 might bring one to the illusion that there is an additional possibility for tree expansion; namely, $1[c \leq 2].0[c < 3].0[c < 3]$. This, however, is a misconception. The priority of a node was derived from the "minimum time that must elapse before the node can be reached, $\tau_{root \rightarrow k}$." Hence, for two sequences that allow for input of all syllabi without delay, $|\sigma_1| < |\sigma_2| \rightarrow \pi_{\sigma_1} < \pi_{\sigma_2}$ holds true. Therefore,

$$\pi_{1[c \leq 2].1[c \leq 2]} < \pi_{1[c \leq 2].0[c < 3].0[c < 3]}$$

¹² The first output literal was bolded to demonstrate the *shortest timed homing prefixes*. If the t-IUT outputs a in response to $1[c=0]$, we can already deduce that the initial state was A, and that t-IUT is currently at node C. An initial state of A is the only state where such an output can legally occur for such specific input. Only an output of β will require us to continue with the t-HS until a final verdict is reached.

¹³ It is inconsequential if N_{des} is a direct descendant of N_{ans} or an indirect one. In either case we denoted the time that elapses to reach N_{des} from N_{ans} by $\tau_{N_{ans} \rightarrow N_{des}}$.

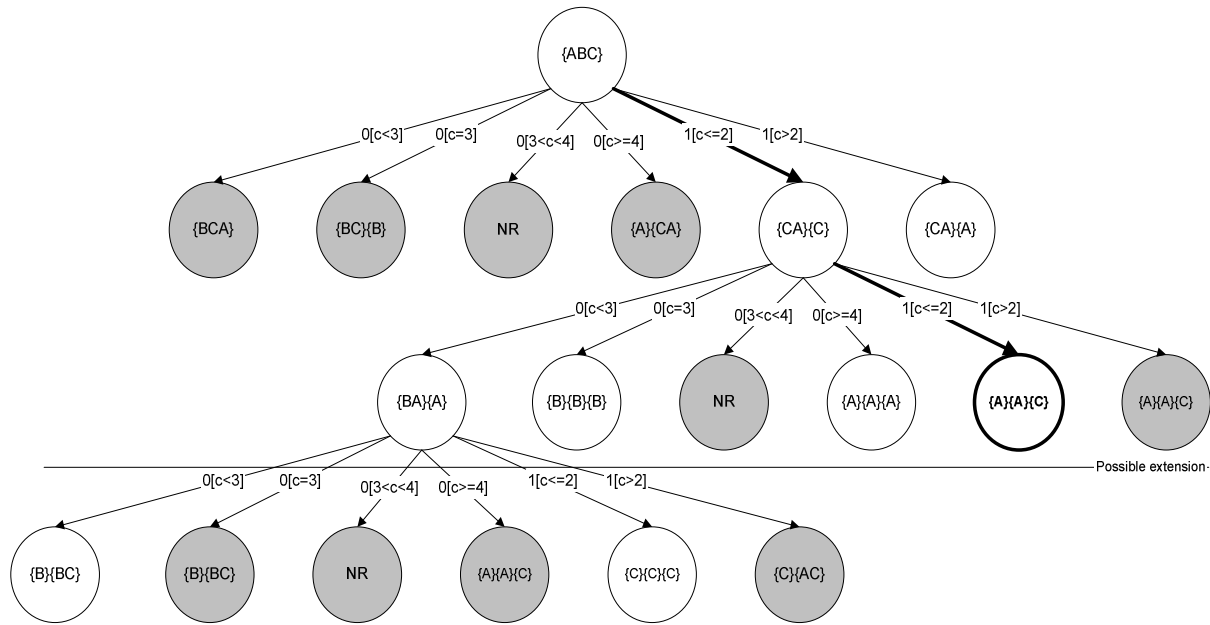


Figure 5. Timed-HT for machine shown in Fig. 4

$\tau_{root \Rightarrow ans}$. By the classification of a node's priority, N_{des} must have a lower priority than N_{ans} ($\rho_{des} < \rho_{ans}$). \square

Theorem II: The length of time that must elapse before we can reach a node j in the t-HT, $\tau_{root \Rightarrow j}$, is equal to the execution time of the [sub-]sequence the node represents, π_{σ_j} .

This theorem evolves directly from the method used for timed homing sequence extraction and construction. \square

Now the proof of execution time optimality is direct:

Theorem III: A t-HS constructed from a highest priority non-terminal leaf node in a t-HT is optimal, $\sigma_{t-hs_{op}}$, as defined in Definition 10.

Proof (by negation): Let us assume the existence of a timed homing sequence, $\sigma_{t-hs_{possible}}$, that executes faster than the sequence found, $\sigma_{t-hs_{found}}$. Specifically, $\pi_{\sigma_{t-hs_{possible}}} < \pi_{\sigma_{t-hs_{found}}}$. By Theorem II, $\tau_{root \Rightarrow possible} < \tau_{root \Rightarrow found}$ for nodes $N_{possible}$ and N_{found} representing $\sigma_{t-hs_{possible}}$ and $\sigma_{t-hs_{found}}$, respectively. This dictates that $\rho_{possible} > \rho_{found}$.

Since until the first acceptable solution N_{found} was found only the highest priority non-terminal leaf nodes were selected for inspection, $N_{possible}$ must either be another non-terminal leaf node such that

$\rho_{possible} \leq \rho_{found}$ or one of their future descendants satisfying the same in-equality.

By Theorem II and the inherit definition of node priority, at any point in time a highest priority non-terminal leaf node represents a [sub-] sequence that executes faster than [sub-]sequences represented by the other non-terminal leaf nodes and all of their future descendants; thus, $N_{possible}$ cannot represent a faster executing timed homing sequence than N_{found} ($\pi_{\sigma_{t-hs_{possible}}} \geq \pi_{\sigma_{t-hs_{found}}}$). \square

C. Multi-clock t-IUT

Based on Assumption 2, the discussion so far only contained constraints based on a single clock. We can, however, relax that assumption and still use the t-HT to find an optimal t-HS, $\sigma_{t-hs_{op}}$.

In [25] and [13], an algorithm for grouping common regions of the famous region graph [9] into zones is provided. Both the extension and its basic ancestor do not limit the timed automaton to a single clock. Rather, the graph is augmented with an additional dimension for every clock that is added to the system. This implies that a zone's dimension increases linearly with the addition of clock into the system. In a single clock system we were able to model zone/region using simple segments. In a 2-clock system we require that zones/regions be modeled using areas. It follows, that an n -clock system would require polyhedrons of order n for modeling the zones/regions that group common behavior of the t-IUT.

The algorithm we developed can easily be built upon the notion of time zones¹⁴. We group time values that behave similarly on specific input into a single group, and treat them all as one. We can then use these time zones as the time constraints on the t-HT's edges. As such, the order of the polyhedrons is inconsequential. Once zones are identified, they can be used as time constraints and inserted into the tree. The remainder of the algorithm functions in the similar manner to what is described above.

IV. ASSESSMENT

To quantify the contribution of a new algorithm, one must provide some sort of comparison factor for which the proposed algorithm performs better than an alternative. Obviously, if an alternative does not exist, that in itself constitutes the greatest contribution measure needed. One could, however, compare the new method to some previously existing method even though it was not meant to solve the exact problem set.

As the author is unaware of an existing algorithm for the generation of optimal timed homing sequences with respect to run time, a choice was made to compare the new algorithm to some previously proposed algorithm for the generation of timed HS. We chose to use the same example in [20], as it gives an algorithm for the generation of timed HS optimal w.r.t. sequence length. In addition, a comparison is done to a transformation method proposed in [25] and [13].

The solution reached in [20] stated that $0[c=3].1[c>2]$ is an optimal sequence for the automaton shown in Fig. 4. This would dictate that $c > 3$ must exist for the completion of its execution. Section B.2 showed that for the same automaton $c \leq 2$ is sufficient for the completion of an equivalent sequence. When it comes to run-time, the current solution is noticeably superior.

It is true, that [20] claimed that the algorithm they presented would allow for a solution optimal w.r.t execution time. It, however, would require "the nodes should be ordered with the lower time constrain placed first". Our method, however, removes this requirement. The ordering is built into the tree expansion algorithm, and the need to consciously place tree nodes in specific order isn't required.

V. CONCLUSION

As discussed and demonstrated above, the advantage of using our direct t-HS generation method is two-fold. Firstly, assuming the existence of a HS, our method will *always* find an optimal t-HS. Secondly, it is all accomplished without

the high transformation costs described in [25], [13] and [20]. This would be an improvement to conformance testing of timed protocols, were it to be implemented within tools.

This algorithm can further be expanded to include other problems within the state identification problem class, such as synchronizing sequences, distinguishing sequence, etc. It can also possibly be expanded to include additional heuristics as to decide which nodes are more promising for expansion.

In addition, we hope to conduct experimentation to quantify the savings gained by our method. To develop a tool that will allow for quick test generation using the above algorithm. We hope the tool will be incorporated into existing testing tools, allowing for better algorithms to be used in labs and industry.

REFERENCES

- [1] Z. Kohavi, *Switching and Finite Automata Theory*. 2nd ed., McGraw-Hill, 1978.
- [2] W.Y.L. Chan, S.T. Voung, and M.R. Ito, "An Improved Protocol Test Generation Procedure based on UIO's," *Proc SIGCOM*, pp. 283-278, 1992.
- [3] G. Gonec, "A Method for the Desing of Fault Detection Experiment," *IEEE Trans. on Comp.*, C-19, pp. 551-558, 1980.
- [4] A.V. Aho, A.T. Dahbura, D. Lee, and M.U. Uyar, "An Optimization Technique for Protocol Conformance Test Generation based on UIO Sequences and Rural Chinese Postman Tours," *IEEE Trans. Commun.*, vol. 39, pp. 1604-1615, 1991.
- [5] K.K. Sabnani, A.T. Dahbura, "A Protocol Test Generation Procedure," *Comp. Networks ISDN Sys.*, vol. 15, pp. 285-297, 1998.
- [6] T.S. Chow, "Testing Software Design Modeled by Finite State Machines," *IEEE Trans. Software Eng.*, SE-4(3), pp. 178-187, 1978.
- [7] G. Luo, G. von Bochmann, and A.F. Petrenko, "Test Selection based on Communicating Nondeterministic Finite State Machines using a Generalized Wp-method," *IEEE Trans. Software. Eng.*, 20(2), pp. 149-162, 1994.
- [8] D. Lee, M. Yannakakis, "Principles and Methods of Testing Finite State Machines – a Survey," *Proc. of the IEEE*, 84(8), pp. 1090-1126, 1996.
- [9] R. Alur, D. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, pp. 183-235, 1994.
- [10] S. Bloch, H. Fouchal, E. Petitjean, and S. Salva, "Some Issues on Testing Real-time Systems," *Int. J. of Comp. and Info. Science*, 2(4), pp. 230, 2001.
- [11] R. Cardell-Oliver, T. Glover, "A Practical and Complete Algorithm for Testing Real-Time Systems," in *Proceedings of the 5th international Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, A. P. Ravn and H. Rischel, Eds. Lecture

¹⁴ The segments used in the example of section B.2 are the realization of this theoretical fact. The time constraints we actually used in the t-HT are the time zones (segments) that were extracted from the single clock automata.

- Notes In Computer Science, vol. 1486. London: Springer-Verlag, 1998, pp. 251-261.
- [12] K. Naik, B. Sarikaya, "Protocol Conformance Test Case Verification using Timed – Transition," in *Proceedings of the 14th International Symposium on Protocol Specification, Testing and Verification*, Vancouver, Canada, 1994.
- [13] S. Salva, H. Fouchal, and S. Bloch, "Metrics for Timed Systems Testing," *4th OPODIS International Conference on Distributed Systems*, Paris, 2000, pp. 177.
- [14] M.U. Uyar, M.A. Fecko, A.S. Sethi, and P.D Amer, "Testing Protocols Modeled as FSM with Timing Parametes," *Computer Networks*, 31(18), pp. 1967-1988, 1999.
- [15] L. Briones, E. Brinksma, "Testing real-time multi input-output systems," in *7th Int. Conf. on Formal Engineering Methods*, ICFEM'05, LNCS vol. 3785, Springer, 2005, pp. 264-279.
- [16] M.G. Merayo, M. Nunez, and I. Rodriguez, "Generation of optimal finite test suites for timed systems," in *Proceedings of the First Joint IEEE/IFIP Symposium on theoretical Aspects of Software Engineering*, TASE, IEEE Computer Society, Washington, DC, 2007, pp. 149-158.
- [17] M.G. Merayo, M. Núñez, and I. Rodríguez, "Formal testing from timed finite state machines," *Comput. Networks*, 52(2), 2008, pp. 432-460.
- [18] A. Khumsi, L. Ouedraogo, "A new method for transforming timed automata," in *Brazilian Symposium on Formal Methods (SBMF)*, Recife, Brazil, 2004, pp. 101-128.
- [19] M. Krichen, S. Tripakis, "State identification problems for timed automata," in *The 17th IFIP Intl. Conf. on Testing of Communicating Systems (TestCom'05)*, LNCS vol. 3502, Springer, 2005, pp. 175-191.
- [20] A. Stulman, S. Bloch, and H.G. Mendelbaum, "Optimal homing sequences for machines with timing constraints," in *WSEAS 4th International Conference on Systems*, Venice, Italy, 2004.
- [21] A. Stulman, S. Bloch, and H.G. Mendelbaum, "Optimal Synchronizing Sequences for Machines with Timing Constraints," in *ESA '05 International Conference on Embedded Systems and Applications*, Las Vegas, NV, USA, 2005.
- [22] J.E. Hopcroft, J.D. Ullmann, *Intoduction to Automata Theory, Languages, and Computation*. Reading Massachusetts, USA: Addison-Wesley, 1979.
- [23] E.P. Hsieh, "Optimal Checking Experiments for Sequential Machines," *IEEE Trans. on Computers*, C-20, pp. 1152-1166, 1971.
- [24] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*. 2nd Ed., Prentice Hall, 2003.
- [25] E. Petitjean, H. Fouchal, "From Timed Automata to Testable Untimed Automata," in *24th IFAC/IFIP International Workshop on Real-Time Programming*, Schloss Dagstuhl, Germany, 1999.

Ariel Stulman received his B.Tech and Applied Sciences from the Jerusalem College of Technology, Israel, in 1997. In 2001 he earned the M.Sc. degree from Bar-Ilan University, Israel, and in 2005 a Ph.D. from the University of Reims Chanpagne-Ardenne, France. As of 2006 he is a faculty member at the Jerusalem College of Technology, Jerusalem, Israel. His research interests include testing of synchronous and concurrent software and protocols. He is a member of the ACM.