# Studying Synchronization Issues for Extended Automata[*]

Natalia Kushik[1] and Nina Yevtushenko[2]

[1]*SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Palaiseau, France*

[2]*Ivannikov Institute for System Programming of the Russian Academy of Sciences, Moscow, Russia*

Keywords: Extended Automata, Synchronizing Sequence, Model based Testing and Monitoring.

Abstract: The paper presents a study of synchronization issues for one of non-classical state models, i.e., a state identification problem widely used in the area of Model based Testing (MBT) and run-time verification / monitoring. We consider Finite Automata (FA) augmented with the context variables and their related updates when the transitions are executed. For such Extended Automata (EA) we define the notions of merging and synchronizing sequences that serve as reset words in MBT, and show that under certain conditions and when every context variable is defined over a ring, it is possible for the extended automata of the studied class to 'repeat' the necessary and sufficient conditions established for the classical automata. Otherwise, in a general case, the problem can be reduced to deriving reset words for classical FA that represent corresponding EA slices.

## 1 INTRODUCTION

Finite state models are widely used as formal specifications in the testing and verification area of discrete and hybrid systems. When deriving test suites with the guaranteed fault coverage, in MBT, one of typical well known problems for finite automata or finite state machines concerns their state identification (Lee and Yannakakis, 1994; Lee and Yannakakis, 1996). Final state identification in some cases can be solved via generation and application of homing and synchronizing sequences (Sandberg, 2004) to the machine under experiment. Such sequences can serve as reset words or checking sequence preambles, when it comes to *active* testing of non-initialized implementations (Hennie, 1964). At the same time, both sequences can minimize the monitoring efforts when testing or verifying a system behavior in a *passive* mode (Kushik et al., 2016). State identification problems are well studied for classical finite automata and finite state machines (FSMs), however when the corresponding state model is augmented with additional parameters / variables, such as for example, timeouts, predicates, input / output parameters, to the best of our knowledge, the problem has not been largely investigated.

Synchronizing sequences bring a machine to a unique final state and are usually considered for machines without outputs (Sandberg, 2004), i.e., for classical automata. For deterministic complete automata the length of such sequence is polynomial and it exists whenever each state pair has a merging sequence.

Note also that when it comes to testing and verification of a discrete event system, be that software or hardware component of a communicating system, it is rather hard to obtain its formal specification as a finite automaton or a finite state machine. Sometimes it is more convenient to consider an *extended* model augmented with parameters listed above. In this paper, we state and solve a problem of the existence check and derivation of a synchronizing sequence for an extended automaton, which looks like a classical FA augmented with context variables that update their values when certain transitions are executed, as well as *special* predicates guarding some transitions which depend on context variables.

When the behavior of an Implementation Under Test (IUT) is described by an extended machine it can well happen that for simplifying the run-time verification or monitoring, not only a reached state is important but rather a state together with the context. As a motivating example, we consider a *Simple Connection Protocol (SCP)* which is designed to 'connect' two entities, negotiating the quality of service at the connection establishment (Alcalde et al., 2004). The SCP allows connecting an entity called the *upper layer* to an entity called the *lower layer*. The upper layer dialogues with the SCP for fixing the quality

[*]This work is partially supported by RSF project N 22-29-01189.

of service (QoS) desirable for the future connection. Later on, the upper layer comes to the lower layer requesting the establishment of a connection. The lower layer accepts or refuses this connection request. If the lower layer accepts the request, then it informs the upper layer that the connection has been established and the upper layer can start transmitting data which is followed by a corresponding acknowledgment. The reader can find an FSM describing the SCP behavior in (Kushik et al., 2016), where a possibility of minimizing the monitoring efforts through the observation of the SCP homing sequences was discussed. In the example below, we abstract from the negotiation step and data transmission, i.e., when monitoring the behavior of the SCP implementation, our observations will be taken at the inputs, i.e., requests, and the following implementation actions. The corresponding extended automaton, describing the SCP connection establishment, is shown in Figure 1.

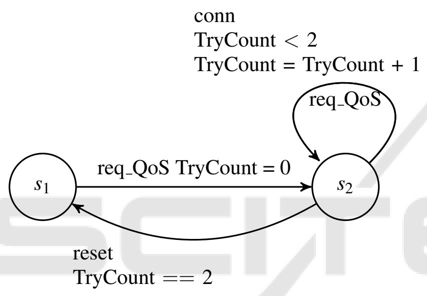

Figure 1: Extended automaton for the connection establishment in the SCP.

In the original FSM in (Kushik et al., 2016), the upper layer tried to establish the connection two times, before receiving the *abort* signal from the lower layer, i.e., in fact, the context variable $TryCount$ in the EA in Figure 1 is defined over the group $(\{0,1,2\}, +mod3)$. Assume, that during the protocol monitoring, one of the properties to be checked is the *safety* of the *abort* signal. Indeed, we would like to make sure not only that *abort* follows the connection request but that there were at least two unsuccessful attempts before.

That is, we would like to observe the output *abort*, when the protocol reaches not only state $state = s_2$ but the configuration $(state = s_2, < TryCount = 2 >)$ (and not $(state = s_2, < TryCount = 1 >)$, for example). In other words, differently from the (Kushik et al., 2016) result, we not only want to know the current state of the protocol implementation when verifying certain properties but even more precisely, we would like to make sure that the configuration of interest has been reached. The latter could allow minimizing the number of properties to be checked, as not all the properties are relevant at different configurations, even for the same state.

In the literature, there have been proposed various definitions of extended FAs and FSMs, see for example (Petrenko et al., 2004; Holzmann, 2004). In (El-Fakih et al., 2016; Petrenko et al., 1999; Petrenko et al., 2004), the distinguishability notions for an EA are considered. However, for the machines of the studied classes, for identifying a final (current) configuration of the machine, to the best of our knowledge, there exist few papers where a homing sequence (HS) is derived for an FSM with timed guards (Tvardovskii and Yevtushenko, 2020) and a synchronizing sequence (SS) is derived for a Timed Automaton (Doyen et al., 2014). In the latter paper, the authors also consider the SS problem for a Weighted Automaton (WA), that is considered as an EA where the weight is a context variable. However, the weight cannot be directly assigned to some integer, and due to this fact, the authors show that in their case, an SS never exists for a non-initialized WA, as two configurations with the same location and different initial weight values cannot be synchronized. In a general case of EA, it is not the case. The reachability problem of WA (Bouyer-Decitre, 2016) is also relevant to our studies, but on the one hand, it is different from the SS problem, and on the other, weights themselves do not affect the behavior of the machine (Droste et al., 2009) which is not the case for context variables of an EA considered in the paper.

We hereafter investigate a specific class of EA where the values of context variables belong to a ring and thus, the update functions are defined accordingly using ring multiplication and addition; predicates are used to verify if a context variable value belongs to a certain ring subset. The provided formal definition of such EA allows establishing the conditions for existence check and derivation of an SS.

The main contribution of the paper is a method for the existence check and derivation of a synchronizing sequence for an EA with the context variables which values belong to a ring, as well as with the appropriate predicates. For a special class of configurations, when the context variables' values belong to an ideal of the ring, to have an appropriate SS, it is necessary and sufficient that the corresponding underlying automaton (context-free slice) has an SS, along with having proper transitions from a state reached by the SS. The same result applies to an EA with mutually exclusive predicates at each state, that verify that a context variable value belongs to an ideal of a ring. Given a set of configurations with the same state and context variables which values belong to an ideal, we also discuss an issue of merging the configurations of the set into a single configuration. If such a sequence exists then it is used for deriving an SS for the given extended automaton.

The structure of the paper is as follows. Section 2 contains preliminaries as well as the problem statement. The existence check and derivation of a merging sequence for two sets of configurations for an extended automaton is discussed in Section 3. Correspondingly, a method for the existence check and derivation of a transfer sequence and of an SS for an extended automaton is presented in Section 4. Section 5 is devoted to EA with mutually exclusive predicates and the related SS derivation problem. Section 6 concludes the paper.

## 2 BACKGROUND AND PROBLEM STATEMENT

In this paper, we consider one of the classical state identification problems, namely we focus on the existence check and derivation of a *synchronizing* sequence for finite extended automata. As usual, a *finite automaton*, simply an *automaton* throughout this paper, is a 3-tuple $\mathbf{A} = (S, M, \delta)$ where $S$ is a finite nonempty set of states, $M$ is a finite nonempty set of actions, $\delta \subseteq S \times M \times S$ is a set of transitions. Note that, similar to (Ito and Shikishima-Tsuji, 2004; Volkov, 2008), we consider automata without the non-observable action. Moreover, in this paper, we focus on complete deterministic automata, i.e., for each pair $(s, m)$, $s \in S$, $m \in M$, there exists exactly one transition $(s, m, s') \in \delta$. Given a sequence / trace $\alpha \in M^\star$ and a state $s$, $\alpha$ takes the automaton to the $\alpha$-*successor* of $s$. The $\alpha$-successor of the subset $S'$ of states is the set of $\alpha$-successors for all states of $S'$. A sequence / trace $\alpha \in M^\star$ is an *SS* for $\mathbf{A}$ if the $\alpha$-successor of the set of states $S$ is a singleton. If the automaton has an SS then the automaton is *synchronizing*. If the automaton has the designated subset $S' \subset S$ of initial states, i.e., is weakly initialized, then this automaton is *synchronizing* if there exists a trace $\alpha$ such that the $\alpha$-successor of the set $S'$ is a singleton. In this paper, we consider non-initialized automata if the converse is not explicitly stated.

A sequence $\alpha \in M^\star$ is a *merging* sequence for two different states $s$ and $p$ of $\mathbf{A}$ if the $\alpha$-successors of $s$ and $p$ coincide, i.e., are the same. It is known (Eppstein, 1990; Natarajan, 1986) that a complete and deterministic automaton is synchronizing if and only if every pair of different states has a merging sequence. In the SCP example, given in Section 1, a merging sequence is a synchronizing sequence for two states $s_1$ and $s_2$, for example, it can be a single input *req_QoS*.

However, if we would like to take into account the values of the context variables when synchronizing the automaton, we need to restrict the corresponding definition and in fact, merge and synchronize not the states, but rather the configurations. That is the reason why in this paper, we consider a special class of extended automata and define the notion of an SS for this class of machines. For the sake of simplicity, we first, abstract from the predicates (or guards) that can potentially label the transitions, only keeping the context variables that can be updated when a transition is executed. Therefore, in this paper, an *extended* automaton is augmented with a finite set of context variables and each transition is labeled with update functions for these variables. To formally define the possible update functions, we furthermore turn to the relevant algebraic structures, and consider that every context variable is defined over a ring.

An extended automaton $\mathbf{A}$ is a 4-tuple $\mathbf{A} = (S, M, T, \delta)$ where $S$ is a finite nonempty set of states, $M$ is a finite nonempty set of actions, $T = \{t_1, \ldots, t_k\}$ is a finite set of context variables which are defined over a finite ring $\mathbf{R} = (R, +, *)$, and $\delta$ is a set of transitions between states from $S$ such that each transition in $\delta$ is a tuple $(s, a, up, s')$, where $s, s' \in S$ are the initial and final states of a transition, $a \in M$ is an (input) action, $up =< f_1, \ldots, f_k >$ is a context update function such that $\forall j = 1, \ldots, k$, the function $f_j(t_j) : R \longrightarrow R$ is a linear combination $h * t_j + b$ where $h, b \in R$. By default, for identity function $f_j$ the context variable $t_j$ does not change its value after the transition is executed and we will simply omit these functions when defining the transitions[1]. An EA is *complete* and *deterministic* when at every state, there exists exactly one transition under each input.

As an example of an abstract EA, consider an automaton $\mathbf{A}$ in Figure 2. This automaton has three states and two context variables $t_1$ and $t_2$ defined over the ring $\mathbf{R} = \mathbf{Z}_{10} = (\{0, \ldots, 9\}, +mod10, *mod10)$; $h_1 = 3$, $h_2 = 1$ while $b_1 = 4$, $b_2 = 0$. All the transitions in the automaton contain the update of context variables, except one, namely $(1, a, 2)$ which is only labeled by a letter $a$, i.e., all the context variables preserve their values when the transition is executed.

As usual, a configuration is a pair $(s, \mathbf{v})$ where $s$ is a state and $\mathbf{v}$ is the context, i.e., $\mathbf{v}$ is a vector of values of context variables. We also consider a finite set of configurations $C_s = (s, V)$ where $V$ is a finite non-empty set of contexts. Moreover, given two configurations $(s, \mathbf{v_1})$ and $(s, \mathbf{v_2})$ and a trace $\sigma$, the $\sigma$-successors of $(s, \mathbf{v_1})$ and $(s, \mathbf{v_2})$ are $(p, \mathbf{u_1})$ and $(p, \mathbf{u_2})$

[1]Here we notice that in a WA in (Doyen et al., 2014), the weight values are also defined over an infinite Abel group $\mathbf{R} = (R, +)$.

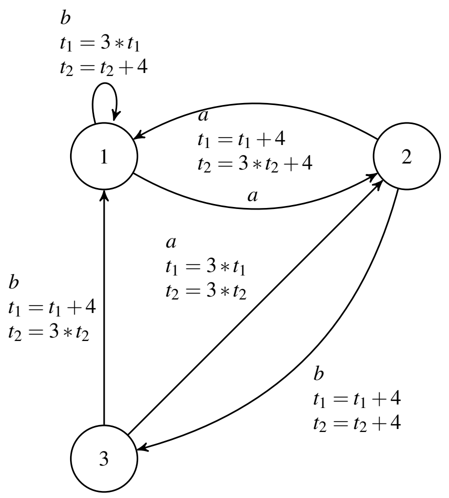

Figure 2: An extended automaton **A**.

for some state $p$.

Given an automaton $\mathbf{A} = (S, M, T, \delta)$ with the set $T = \{t_1, t_2, \dots, t_k\}$ of context variables with the values in $R$, we further consider the context-free slice (El-Fakih et al., 2008) $\mathbf{A}^{aut}$ that is the underlying classical automaton without the context variables, while $\mathbf{A}^{sim}$ denotes the classical automaton that is obtained by the simulation of $\mathbf{A}$. By definition, both, $\mathbf{A}^{aut}$ and $\mathbf{A}^{sim}$, are complete and deterministic automata if an initial EA is complete and deterministic.

Given the set $A = A_1 \times A_2 \times \cdots \times A_k$, $A_j \subseteq R$, two sets of configurations $C_s = (s, A)$ and $C_p = (p, A)$ and a set $B = B_1 \times B_2 \times \cdots \times B_k$, $B_j \subseteq R$, of contexts, we would like to check if there exist a sequence $\sigma$ of actions and a state $q$ such that from each configuration $(s, \mathbf{v}) \in C_s$ and from each configuration $(p, \mathbf{v}) \in C_p$ the sequence $\sigma$ takes the extended automaton $\mathbf{A}$ to some configuration of the set $C_q = (q, B)$. If the trace $\sigma$ exists then we further refer to it as a $(q, B)$-*merging* sequence for the sets $C_s$ and $C_p$. A sequence which $(q, B)$-merges $n$ sets of configurations $C_1 = (s_1, A), \dots, C_n = (s_n, A)$, is a $(q, B)$-*merging* sequence for the set of these $n$ subsets of configurations.

A sequence which $(q, R^k)$-merges $n$ sets of configurations $C_1 = (s_1, R^k), \dots, C_n = (s_n, R^k)$ is a $q$-*synchronizing sequence* for the automaton $\mathbf{A}$. A sequence which merges $n$ sets of configurations $C_1 = (s_1, R^k), \dots, C_n = (s_n, R^k)$ into a singleton $(s, \mathbf{v})$ is a *synchronizing sequence* for the automaton $\mathbf{A}$.

Given a set of configurations $C_s = (s, A)$, if there exist a singleton $(q, \mathbf{v})$ and a sequence that takes the automaton from each configuration of the set to $(q, \mathbf{v})$, then this sequence is a *transfer* sequence from $C_s$ to $(q, \mathbf{v})$ or a *synchronizing* sequence for $C_s$.

In this paper, we tackle the following problems:

1. Existence check of a $(q, B)$-merging sequence for two sets of configurations of a given automaton;
2. Derivation of a $(q, B)$-merging sequence for two sets of configurations, whenever exists;
3. Existence check and derivation of a $(q, B)$-merging sequence for an extended automaton;
4. Existence check and derivation of a transfer sequence for a subset $C_q = (q, B)$ of an extended automaton;
5. Derivation of a synchronizing sequence for an extended automaton.

## 3 EXISTENCE CHECK AND DERIVATION OF A MERGING SEQUENCE FOR TWO SETS OF CONFIGURATIONS IN AN EXTENDED AUTOMATON

Note that a $(q, B)$-merging sequence for a pair of states $\{s, p\}$ only exists if in the context-free slice $\mathbf{A}^{aut}$ of the extended automaton $\mathbf{A}$ there exists a sequence merging states $s$ and $p$ into state $q$.

**Proposition 1.** *1. If for states s and p there is no merging sequence in the slice* $\mathbf{A}^{aut}$ *then there is no* $(q, B)$*-merging sequence for any two sets of configurations* $C_s = (s, A)$ *and* $C_p = (p, A)$, $A \subseteq R^k$.
*2. A* $(q, R^k)$*-merging sequence exists for the sets* $C_s = (s, A)$ *and* $C_p = (p, A)$ *if and only if in the slice* $\mathbf{A}^{aut}$*, there exists a sequence merging states s and p into state q.*

The first statement of the proposition establishes the necessary conditions for the existence of a $(q, B)$-merging sequence for two sets of configurations for an arbitrary $B \subseteq R^k$. However, according to the second statement of the proposition, if $B = R^k$ then the conditions become necessary and sufficient.

If $B \subset R^k$, then the sufficient conditions for the existence of a $(q, B)$-merging sequence can be obtained similar to ‘classical’ synchronizing / homing tree approaches (Sandberg, 2004). For that matter we adapt the notion of a successor tree for such an extended automaton and later on propose the corresponding truncating rules that allow deriving a $(q, B)$-merging sequence or to conclude that such a sequence does not exist.

Given the set $A = A_1 \times A_2 \times \cdots \times A_k$, $A_j \subseteq R$, the set $B = B_1 \times B_2 \times \cdots \times B_k$, $B_j \subseteq R$, two sets of configurations $C_s = (s, A)$ and $C_p = (p, A)$, the root of the tree is labeled by the pair $\{C_s, C_p\}$. Edges of the tree are labeled by possible (input) actions. Given a current node labeled by a pair $\{C_x = (x, A'_1 \times A'_2 \times \cdots \times$

$A'_k), C_y = (y, A''_1 \times A''_2 \times \cdots \times A''_k)\}$, this node is adjacent to a node labeled by $\{C_q = (q, L'_1 \times L'_2 \times \cdots \times L'_k), C_z = (z, L''_1 \times L''_2 \times \cdots \times L''_k)\}$ through an arc labeled by $m$, if **A** contains the following transitions: $(x, m, up, q)$, $(y, m, up, z)$ and $L'_j$ is obtained from $A'_j$ through the application of related update function $f_j$ for the variable $t_j$ for the transition $(x, m, up, q)$, while $L''_j$ is obtained from $A''_j$ through the application of related update function $f_j$ for $t_j$ for the transition $(y, m, up, z)$, $j \in \{1, 2, \ldots, k\}$.

Truncating rules are defined as follows.

**Rule 1:** A node labeled by a pair $\{C_q = (q, A' = A'_1 \times A'_2 \times \cdots \times A'_k), C_z = (z, A'' = A''_1 \times A''_2 \times \cdots \times A''_k)\}$ is terminal if at the same level or upper in the tree there exists a node labeled by a pair $\{C_x = (q, L' = L'_1 \times L'_2 \times \cdots \times L'_k), C_y = (z, L'' = L''_1 \times L''_2 \times \cdots \times L''_k)\}$ such that $L'_j \subseteq A'_j$ and $L''_j \subseteq A''_j$, $j \in \{1, 2, \ldots, k\}$.

**Rule 2:** A node labeled by a pair $\{C_q = (q, A'), C_z = (z, A'')\}$ is terminal if $q = z$, and $A'_j \subseteq B_j$, $A''_j \subseteq B_j$, $j \in \{1, 2, \ldots, k\}$.

**Proposition 2.** *A sequence $\alpha$ is a $(q, B)$-merging sequence for sets $C_s = (s, A)$ and $C_p = (p, A)$ of configurations of the extended complete deterministic automaton* **A** *if and only if it labels a path to a node truncated using Rule 2. If all the nodes in the tree are truncated using Rule 1 then there no SS for the automaton* **A**.

Note that by definition, $\alpha$ is a $(q, B)$-merging sequence for the sets $(C_s, A)$ and $(C_p, A)$ if and only if the automaton **A** is taken by $\alpha$ from any configuration of the set $(C_s, A)$ to a configuration of the set $(q, B)$ and the same holds for any configuration of $(C_p, A)$. In the successor tree, it is exactly the case when $\alpha$ labels a path to a node that is terminal due to Rule 2.

Note also that rules 1 and 2 provide an estimation of the length of a shortest $(q, B)$-merging sequence for the sets $C_s = (s, A)$ and $C_p = (p, A)$. Indeed, it is limited by the number of pairs $\{C_q = (q, A' = A'_1 \times A'_2 \times \cdots \times A'_k), C_z = (z, A'' = A''_1 \times A''_2 \times \cdots \times A''_k)\}$ and thus can be estimated as $O(n^2 |R|^{2k})$ but in reality is much shorter when it exists. We would like to highlight the fact that the $(q, B)$-merging sequence derivation strategy can be also applied in the case of infinite ring **R**, however another truncating rule should be then added; the latter should define the maximal desirable length of a merging sequence in question.

**Proposition 3.** *A sequence $\alpha$ is a $(q, B)$-merging sequence for the complete deterministic extended automaton* **A** *if and only if $\alpha$ is a $(q, B)$-merging sequence for each pair of different sets of configurations $C_s = (s, R^k)$ and $C_p = (p, R^k)$. If there is no $(q, R^k)$-merging sequence for the extended automaton* **A** *then there is no SS for the automaton* **A**.

Consider a slightly modified automaton in Figure 2 when the update functions at the transition from state 1 to state 2 under input $a$ are not identities but $t_1 = 2$ and $t_2 = 2$. By direct inspection one can assure that there is an SS *bba* that takes the automaton from any configuration to the configuration $(2, < 2, 2 >)$.

## 4 EXISTENCE CHECK AND DERIVATION OF A TRANSFER AND A SYNCHRONIZING SEQUENCE FOR EXTENDED AUTOMATA

Note that, differently from classical automata, for the existence check of an SS it is not sufficient to have the merging sequences for all pairs of states of the underlying context-free automaton; nor it is sufficient to have the merging sequences for all pairs of sets of configurations. The reason is that the configurations of the obtained sets should be brought into the set of configurations for which there exists a sequence that transfers this set to a single configuration.

### 4.1 $(q, B)$-Merging Sequence Derivation

Under certain conditions over the automaton **A**, existing necessary and sufficient conditions for classical automata can be somehow 'repeated'. Below, as before, we consider that the EA **A** is complete and deterministic.

**Proposition 4.** *Given an ideal $I$ of the ring* **R**, *let for every context variable $t_j$ and its update function $h_j * t_j + b_j$, it holds that $b_j$ is in $I$. Then $\{C_s = (s, I^k), C_p = (p, I^k)\}$ has a $(q, I^k)$-merging sequence if and only if $\{s, p\}$ has a merging sequence in the related context-free slice* $\mathbf{A}^{aut}$.

Indeed, by definition of update functions, after updating the value of any context variable, it still belongs to $I$, and thus the context-free slice $\mathbf{A}^{aut}$ defines the existence of the $I^k$-merging sequence.

**Corollary 1.** *Given an ideal $I$ of the ring* **R**, *let for every context variable $t_j$ and its update function $h_j * t_j + b_j$, it holds that $b_j$ is in $I$. Then the automaton* **A** *with the initial set of configurations $(s_1, I^k), \ldots, (s_n, I^k)$ has a $(s_j, I^k)$-merging sequence for some $j \in \{1, \ldots, n\}$, if and only if each state pair $\{s, p\}$ in its context-free slice has a merging sequence.*

As an example, consider again the automaton in Figure 2 and an ideal $I = \{0, 2, 4, 6, 8\}$.

A sequence *ba* is the $(2, I^2)$-merging sequence in this case. Note that the two sets $(1, I^2)$ and $(3, I^2)$ are

$(2,I^2)$-merged by a single input $a$, and $ba$ is a $(2,I^2)$-merging sequence for the whole automaton $\mathbf{A}$.

## 4.2 Deriving an SS for a Set $(q,B)$ of an Extended Automaton

We now study whether given a pair $(q,I^k)$, there exists a configuration $(p,\mathbf{v})$ and an input sequence $\beta$ such that $\beta$ takes the automaton from each configuration of the set $(q,I^k)$ to $(p,\mathbf{v})$. We refer to such sequence as a *transfer* sequence from $(q,I^k)$ to $(p,\mathbf{v})$. Suppose that a transfer sequence $\beta = x_1 \dots x_n$ exists and for a context variable $t_j$ of the configuration we have the following updates: $h_1 * t_j + b_1, \dots, h_n * t_j + b_n$ when applying this input sequence. Consider now two configurations of the set with the initial value of context variable $t_j$ equal to $z_1$ and $z_2$. In order to get the same value of this variable after applying the sequence $\beta$ it has to be held that $h_1 * \cdots * h_n * z_1 = h_1 * \cdots * h_n * z_2$. To prove this, consider the formulas $h_n * (h_{n-1} * (\dots z_1) + b_{n-1}) + b_n$ and $h_n * (h_{n-1} \cdot (\dots z_2) + b_{n-1}) + b_n$. If the results of the corresponding functions are equal then $b_n$ can be deleted as well as all the products of the type $h * b$ as they belong to the ideal $I$. The results of two functions are equal if and only if $h_n * h_{n-1} * \cdots * h_1 * z_1 = h_n * h_{n-1} \cdots * h_1 * z_2$.

Therefore, there exists $z' \in I$ such that for any item $z \in I$, the product $h_n * h_{n-1} * \cdots * h_1 * z$ is $z'$. Thus, this $z'$ can be only 0.

Correspondingly, given a ring $\mathbf{R}$ without zero divisors, a transfer sequence exists if and only if there is a path to some state such that for each context variable $t_j$ there is a transition of the path with the update function $t_j = b_j$. If the ring has zero divisors then the conditions become only sufficient, since in this case, the above product has to be a proper zero divisor. For instance, in the above example (Figure 2) it can happen when the product equals 5.

**Proposition 5.** *1. Given a ring $\mathbf{R}$ without zero divisors and a set $(s,I^k)$ of configurations, there exists a transfer sequence for $(s,I^k)$ if and only if there exist a state $p$ and a path from state $s$ to $p$ such that for each context variable $t_j$ there is a transition of the path with the update function $t_j = b_j$.*
*2. Given an arbitrary ring $\mathbf{R}$ and a set $(s,I^k)$ of configurations, let there exist a state $p$ and a path from state $s$ to $p$ such that for each context variable $t_j$ there is a transition of the path with the update function $t_j = b_j$. Then the sequence labeling the path is a transfer sequence for the set $(s,I^k)$.*

Indeed, consider two configurations $(s,t'_1 \dots t'_k)$ and $(s,t''_1 \dots t''_k)$ of the set $(s,I^k)$. For a transfer sequence $\beta$ and $j = 1,\dots,k$, we have the update function $k_j * t_j + b_j$. Therefore, $h_n * (h_{n-1} * (\dots t'_j) + b_{n-1}) + b_n = h_n * (h_{n-1} * (\dots t''_j) + b_{n-1}) + b_n$. Then $b_n$ can be deleted as well as all the products of the type $h * b$ as they belong to the ideal $I$. Correspondingly, by induction, the results of two functions are equal if and only if $h_n * h_{n-1} * \cdots * h_1 * t'_j = h_n * h_{n-1} \cdots * h_1 * t''_j$. Since the same holds for $t'_j = 0$, the latter means that $h_n * h_{n-1} * \cdots * h_1 = 0$. At the same time, once in the path there exists an update function $t_j = b_j$, in the postfix of the path we get the same $t_j$ value independently of the initial value of this variable.

Here we note that the conditions of part 2 of Proposition 5 can be modified when the ring has zero divisors. In this case, the product $h_n * h_{n-1} * \cdots * h_1 = h$ can have only non-zero items but at least one of them is a zero divisor.

**SS Derivation.** The process of deriving an SS for a complete deterministic extended automaton described above, i.e., where the values of context variables belong to an ideal $I$ of a finite ring $\mathbf{R}$, can be performed in two steps.

**Step 1:** To check whether the underlying automaton (context-free slice) $\mathbf{A}^{aut}$ has an SS. If there is no SS then the extended automaton $\mathbf{A}$ has no SS. Otherwise, derive the set of all states $\{s_1,\dots,s_l\}$ such that there exists an SS to these states.

**Step 2:** Let the automaton have an $(q,I^k)$-merging sequence to the set $(p,I^k)$ of configurations. If there exist a state $s_j \in \{s_1,\dots,s_l\}$, a state $p$ and a path from state $s_j$ to $p$ such that for each context variable $t_j$ there is a transition of the path with the update function $t_j = b_j$, then there exists a transfer sequence for $(s_j,I^k)$ and thus, there exists an SS for the extended automaton with the initial set of $(s_1,I^k),\dots,(s_n,I^k)$ of configurations. This SS is obtained by prolonging an $(q,I^k)$-merging sequence to state $s_j$ by a transfer sequence from $(s_j,I^k)$.

Note that for the EA in Fig. 2 the conditions of Proposition 5 do not hold and by direct inspection, one can assure that the EA does not possess an SS. However, if we change an update function at state 1 for input $b$ for $t_2$ as $t_2 = 4$ and an update function at state 2 for input $b$ as $t_1 = 4$ then the EA has an SS *baab*.

It is also important to underline that if the ring has no zero divisors and there are no such states $s_j$ and $p$ at Step 2, there is no guarantee that the extended automaton has no SS. The reason is that at Step 1, there can exist an $(q,(I')^k)$-merging sequence where $I'$ is a proper subset of $I$ for which a corresponding transfer sequence can exist.

We also notice that every context variable $t_j$ can be defined over a proper ring $\mathbf{R}_j$ and correspondingly, the context $\mathbf{v}$ will be defined not over $\mathbf{R}^k$ but over the ring that is the Cartesian product of $\mathbf{R}_j$. In this case, the statements of the paper should be slightly modified.

## 5 SYNCHRONIZATION ISSUES FOR AUTOMATA WITH PREDICATES

We now add simple predicates to an EA and show that some results of the previous section still hold. We assume that an extended automaton has predicates where a predicate $P_j$ is a function defined over the context variable $t_j$; $P_j$ is a mapping $P_j : R \longrightarrow \{True, False\}$ of the type $t_j \in B$, $B \subseteq R$ or its negation.

The transition is *unconditional* if the predicate $P_j$ is $True$ for any value of $t_j$; then by default, we do not associate any predicate with such a transition. Due to the definition of predicates, every two predicates are mutually exclusive, i.e., the automaton $\mathbf{A}^{sim}$ again is complete and deterministic. However, if an EA $\mathbf{A}$ has predicates then the context-free slice $\mathbf{A}^{aut}$ of the deterministic EA can be non-deterministic. Nevertheless, the definition of merging sequences and a synchronizing sequence stay the same for an automaton with the above predicates.

Fig. 3 contains an example automaton $\mathbf{A}$ augmented with predicates. Note that for this augmented automaton, $ba$ is not a $(2, I^2)$-SS anymore. In fact, a $(q, I^2)$-SS cannot start with input $b$ due to the non-determinism of the slice $\mathbf{A}^{aut}$. There is however a longer $(q, I^2)$-synchronizing sequence, for example $aba$.

**Proposition 6.** *Given a set of configurations $C_s = (s, W)$ and a sequence $\sigma$, if the $\sigma$-successor of $s$ in the context-free slice $\mathbf{A}^{aut}$ of the automaton $\mathbf{A}$ is a set $Q$ of states then the $\sigma$-successor of $C_s$ in $\mathbf{A}$ is contained in the union of some sets $C_q$ over all $q \in Q$.*

**Corollary 2.** *If the context-free slice $\mathbf{A}^{aut}$ is synchronizing then the automaton $\mathbf{A}$ is $(q, R^k)$-synchronizing.*

The corollary establishes the sufficient condition for the existence of a $(q, R^k)$-synchronizing sequence for an extended automaton. Note that this condition is not necessary even for a complete and deterministic EA. However, there is a proper case of an extended automaton with predicates when the conditions of the corollary become necessary and sufficient.

Similar to the results of the previous section, let $I$ be an ideal of the ring $\mathbf{R}$. Consider a complete extended automaton $\mathbf{A}$ with the following features. A transition of the EA can have a predicate $P(t_j)$ which

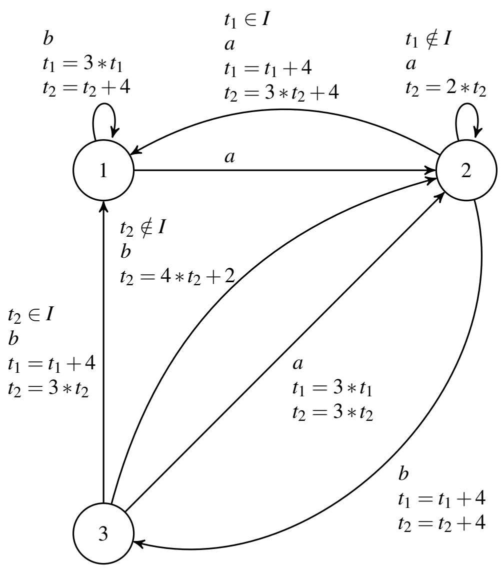

Figure 3: An extended automaton $\mathbf{A}$ augmented with predicates.

is $True$ if $t_j$ is in $I$ or its negation. Moreover, there are the update functions of the kind $h * x + b$ where $h, b$ are in $I$.

**Proposition 7.** *Given a set of configurations $C_s = (s, R^k)$ and an action $m$, the set where the automaton $\mathbf{A}$ is taken from any configuration of the set $C_s$ by action $m$ is a subset of a set $(s, I^k)$.*

Due to the above proposition, after applying any input (action) at the initial configuration we will reach a configuration where the context is in $I^k$. Therefore, the problem should be solved for a submachine with subsets of such configurations from which there exist only unconditional transitions. That is, the results of the previous section can be directly applied.

## 6 CONCLUSION

In this paper, we studied a problem of the existence check and derivation of synchronizing sequences for extended finite automata that are widely used in MBT and monitoring. We investigated a particular class of those when the context variables are defined over a finite ring and in this case, the conditions for the existence check of an SS can be established. In fact, when the updates are represented by linear functions for which the coefficients belong to an ideal, an SS can be derived based on merging sequences for pairs of sets of configurations combined with a corresponding transfer sequence. We established the conditions for the existence of such a transfer sequence. The same results hold for a particular class of the extended automata with predicates, which we also described in the paper.

As a future work, we plan to extend the studied EA classes, by adding input/output parameters, and considering other update functions and predicates. Synchronizing sequences with appropriate features can also be studied, similar to safe synchronizing sequences in (Doyen et al., 2014) when an SS does not traverse appropriate (unsafe) states.

Finally, all the fundamental results presented in the paper need a thorough experimental evaluation, concerning their performance when it comes synchronization issues in MBT and monitoring. We plan to perform such experimental study with various (distributed) networking systems in the future.

## REFERENCES

Alcalde, B., Cavalli, A. R., Chen, D., Khuu, D., and Lee, D. (2004). Network protocol system passive testing for fault management: A backward checking approach. In de Frutos-Escrig, D. and Núñez, M., editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2004, 24th IFIP WG 6.1 International Conference, Madrid Spain, September 27-30, 2004, Proceedings*, volume 3235 of *Lecture Notes in Computer Science*, pages 150–166. Springer.

Bouyer-Decitre, P. (2016). Optimal reachability in weighted timed automata and games. In Faliszewski, P., Muscholl, A., and Niedermeier, R., editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 3:1–3:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Doyen, L., Juhl, L., Larsen, K. G., Markey, N., and Shirmohammadi, M. (2014). Synchronizing words for weighted and timed automata. In Raman, V. and Suresh, S. P., editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Droste, M., Kuich, W., and Vogler, H. (2009). *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition.

El-Fakih, K., Kolomeez, A., Prokopenko, S., and Yevtushenko, N. (2008). Extended finite state machine based test derivation driven by user defined faults. In *First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008*, pages 308–317. IEEE Computer Society.

El-Fakih, K., Yevtushenko, N., Bozga, M., and Bensalem, S. (2016). Distinguishing extended finite state machine configurations using predicate abstraction. *J. Softw. Eng. Res. Dev.*, 4:1.

Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3):500–510.

Hennie, F. C. (1964). Fault detecting experiments for sequential circuits. In *5th Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, USA, November 11-13, 1964*, pages 95–110.

Holzmann, G. J. (2004). *The SPIN Model Checker - primer and reference manual*. Addison-Wesley.

Ito, M. and Shikishima-Tsuji, K. (2004). Some results on directable automata. In *Theory Is Forever, Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*, pages 125–133.

Kushik, N., López, J., Cavalli, A. R., and Yevtushenko, N. (2016). Improving protocol passive testing through "gedanken" experiments with finite state machines. In *2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016, Vienna, Austria, August 1-3, 2016*, pages 315–322. IEEE.

Lee, D. and Yannakakis, M. (1994). Testing finite-state machines: State identification and verification. *IEEE Trans. Computers*, 43(3):306–320.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84:1090–1123.

Natarajan, B. K. (1986). An algorithmic approach to the automated design of parts orienters. In *Proceedings of Symposium on Foundations of Computer Science (SFCS)*, pages 132–142.

Petrenko, A., Boroday, S., and Groz, R. (1999). Confirming configurations in EFSM. In Wu, J., Chanson, S. T., and Gao, Q., editors, *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China*, volume 156 of *IFIP Conference Proceedings*, pages 5–24. Kluwer.

Petrenko, A., Boroday, S., and Groz, R. (2004). Confirming configurations in EFSM testing. *IEEE Trans. Software Eng.*, 30(1):29–42.

Sandberg, S. (2004). Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, pages 5–33.

Tvardovskii, A. S. and Yevtushenko, N. V. (2020). Deriving homing sequences for finite state machines with timed guards. *Model. Anal. Inform. Sist.*, 27(4):376–395.

Volkov, M. V. (2008). Synchronizing automata and the černý conjecture. In *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, pages 11–27.