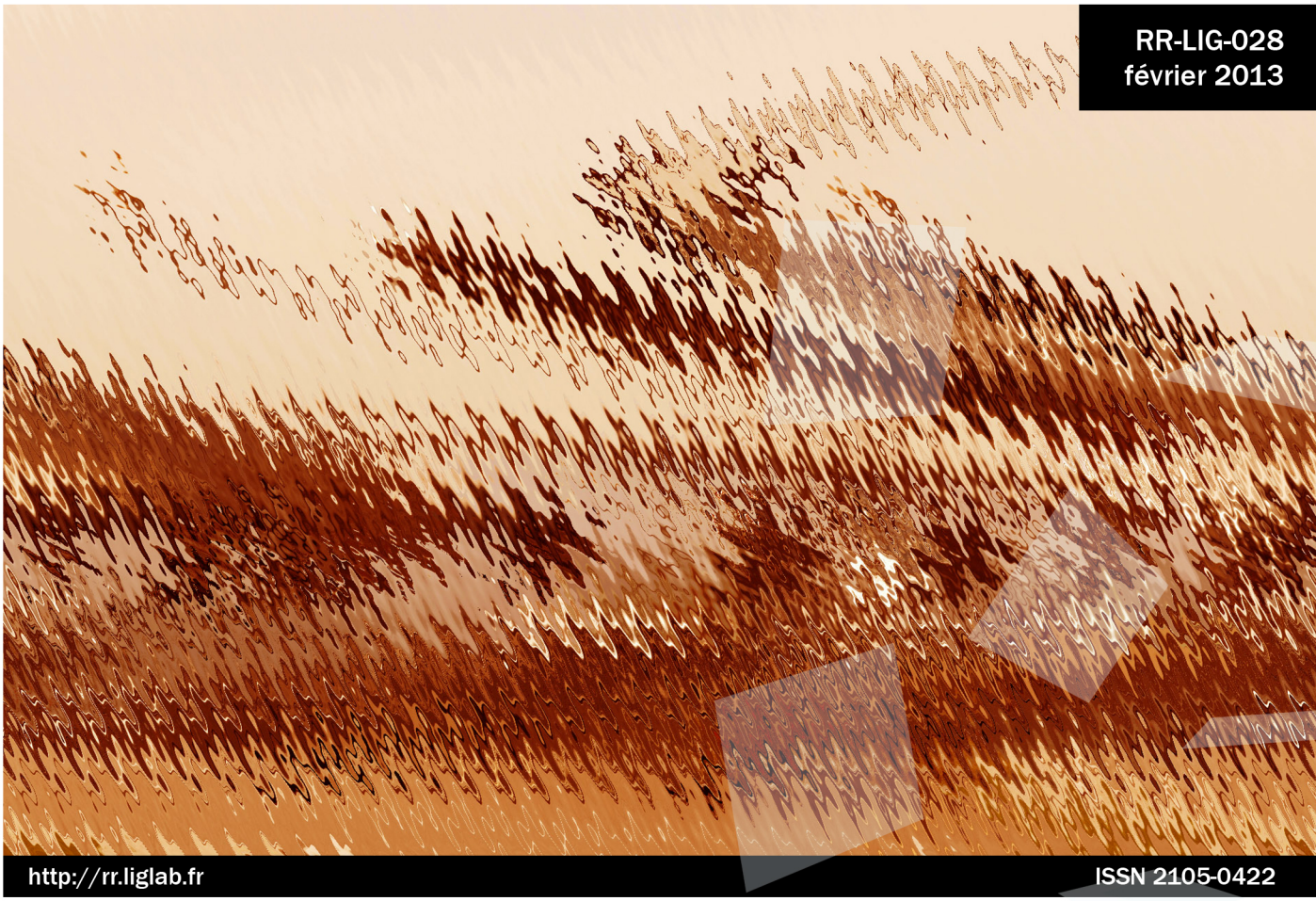
Verification of Modular Systems with Unknown Components Combining Testing and Inference

Roland Groz, Keqin Li, Alexandre Petrenko

Les rapports de recherche du LIG

Verification of Modular Systems with Unknown Components Combining Testing and Inference

Roland GROZ, Professor, LIG, University of Grenoble (Grenoble-INP), France
Keqin LI, SAP Research, France
Alexandre PETRENKO, Lead Researcher, CRIM, Montréal, Canada



RR-LIG-028
février 2013

http://rr.liglab.fr
ISSN 2105-0422

L I G

Verification of Modular Systems with Unknown Components Combining Testing and Inference

Roland Groz [1], Keqin Li [2], Alexandre Petrenko [3]

1 Université de Grenoble, LIG Lab, France. Roland.Groz@imag.fr

2 SAP Research, France. Keqin.Li@sap.com

3 CRIM, Canada. Alexandre.Petrenko@crim.ca

Abstract. Verification of a modular system composed of communicating components is a difficult problem, especially when the formal specifications, i.e., models of the components are not available. Conventional testing techniques are not efficient in detecting erroneous interactions of components because interleavings of internal events are difficult to reproduce in a modular system. The problem of detecting intermittent errors and other compositional problems in the absence of components' models is addressed in this paper. A method to infer a controllable approximation of communicating components through testing is elaborated. The inferred finite state models of components are used to detect compositional problems in the system through reachability analysis. To confirm a flaw in a particular component, a witness trace is used to construct a test applied to the component in isolation. The models are refined at each analysis step thus making the approach iterative.

Keywords*: Model Inference, Testing, Verification, Input/Output Transition Systems, Finite State Machines, Intermittent Errors.*

1. Introduction

Integration of components is now a major mode of software development. Very often, components coming from outside sources (such as COTS) have to be connected to build a system. In most cases, the components do not come with a formal model, just with executable or in some cases source code. At the same time, the interaction of components may lead to integration bugs that may be hard to find and trace, especially in the absence of any model or other development information. In this paper, we are targeting compositional problems in the behaviours of a system composed of communicating components. Specifically, we aim at identifying intermittent (sporadic) errors occurring in event interleavings that are difficult to reproduce in an integrated system. Generally speaking, the system may produce several event interleavings in response to a given external input sequence, if that sequence is applied several times. This occurs in particular when the execution order of the components changes over time from one experiment to another, typically because of different scheduling, varying load and communication jitters. However, it is unrealistic to expect to be able to enforce all the interleavings during testing. Therefore, the intermittent errors are hard to elicit and to reproduce in functional testing.

On the other hand, all potential interleavings can be checked for potential errors by instrumenting the modular system and executing it in a controlled environment to observe all the possible executions and interactions of all its components, see, e.g., Verisoft [God97]. For components without source code, this approach may not be applicable and a model-based approach can be attempted. Indeed, if components' models are available, the global model state space can be exhaustively searched (reachability analysis) to look for compositional problems, using, for instance, a model checker. As stated earlier, the models usually do not exist. One possibility would be to reverse engineer them from the code, but reconstruction based on static analysis has a number of limitations. The main alternative is to infer them from executions. However, given the complexity of typical software components, it is unrealistic to assume that components could be modelled with a perfect abstraction in a finite, compact representation. Inferring approximated models of components in a given modular system appears to be more realistic.

In this paper, we are developing an approach to verify a modular system by inferring tuneable approximated models of its components through testing and performing reachability analysis to detect intermittent errors and other compositional problems in the system. This approach possesses two main advantages regarding the models that are inferred. First, it allows derivation of models of the components describing the behaviours that can actually be exhibited in the integrated system.

Typically, components bundle a number of functions, but it is often the case that only a subset of those functions are used in the system, so inferring them in isolation is hard if not impossible; whereas our approach delivers models which omit behaviours unused in the integrated system. Second, the models are determined with a controllable precision to balance between the level of abstraction and the amount of efforts needed to obtain the models.

We make the following assumptions about a given system:

- The system interacts with a slow environment which submits external inputs only when the system stabilizes.
- Components are black boxes that behave as finite state (Mealy) machines and interact asynchronously. In response to an input, a component can produce several outputs to other components or the environment of the system.
- The models of some components are unknown.
- Each component is deterministic; however, due to possible jitter in communication delays, or scheduling of components, the system might not be.
- A set of controllable external input actions of the system is known. Those are the actions that the tester can provide to the system under test.
- A set of observable actions which includes all output actions of the components is given. Those are the actions that the tester can observe in the system under test.

No additional information about the system, such as the number of states, a priori given positive or negative samples of its behaviour or teacher [KV94], often used in traditional model learning, is available for model inference. The components are modelled using Finite State Machine (FSM) with multiple outputs, or equivalently Input/Output Transition System (IOTS) (with restrictions). A modular system is composed of IOTS components that communicate asynchronously through queues modelled by IOTS. We define an approximation of an FSM, called Z-quotient, based on state distinguishability achieved by the set of input sequences Z. The precision of this approximation can be controlled by the parameter Z, which can be seen as a partial state-characterization set. A full characterization set, as used in the W-method [Vas73], might not be known, since identification of a state machine is in general infeasible without knowing the number of its states.

The first contribution of this paper is an algorithm that computes a Z-quotient for a system treated as a black box (thus, for a whole system) , by testing in the two following steps: behaviour exploration bounded by the parameter Z and “folding” of the observed behaviour by state merging using trace inclusion relation. More precisely, the inference efforts are scoped to a part of the system, called a testable model, defined by the controllable and testable actions. This allows one to focus on interesting features of the systems, while abstracting the others. We then elaborate an approach to infer a Z-quotient of a modular system. The Z-quotient of the system with observable internal actions in the form of an IOTS is used to infer initial models of unknown components by projecting the quotient. Reachability analysis of the models is next performed to identify witness (diagnostic) traces of a composition problem, such as unspecified receptions, livelocks, and races. The identified witness needs to be tested in the real system, to check whether it is an artefact coming from our approximations or the system indeed has this problem. However, since we cannot control the delays in the integrated system, each unknown component involved in a questionable execution is tested in isolation on a projection of the witness trace. A witness refuted by testing the components yields new observations. The models are then refined using new observations and the process iterates until the obtained models are well-formed or a composition problem is confirmed. The obtained component models are consistent with all observations made on the real system. Once composed, they are at least as accurate as the Z-quotient model of the system.

The problems coming from delays and jitters are often hard to elicit and to reproduce in functional testing; they often appear as a side effect in stress testing, but are harder to analyse in that stage. The crux of our approach is precisely to make it possible to identify such intermittent problems that occur only under specific circumstances in integrated systems. This is achieved through the proposed combination of inference, reachability analysis and testing.

The paper is organized as follows. Section 2 provides basic definitions on state models used in the rest of the paper: FSM and Input/Output Transition System. Section 3 defines a Z-quotient of those models and presents an algorithm for its inference. Inference of modular systems and a verification approach are elaborated in Section 4. The notion of a slow asynchronous product is defined to formalize the interactions of components in a slow environment and several known compositional problems along with the witness traces are formally defined. The approach is illustrated on a small example. Section 5 discusses the related work. Section 6 concludes the paper.

2. Basic Definitions

In this section, we first give some basic definitions of the two state models, Finite State Machine (FSM) and Input/Output Transition System (IOTS) used to model a System Under Test (SUT) and then formulate the conditions, under which trace equivalence of the two models, FSM and IOTS, can be established, allowing to use them interchangeably.

2.1. FSM Model

We use a slightly generalized definition of finite state machines which allows multiple outputs for transitions.

A *Finite State Machine with multiple outputs* (FSM) A is a 6-tuple (S, s_0, I, O, E, h), where

- S is a finite set of states with the initial state s_0;
- I and O are finite non-empty disjoint sets of inputs and outputs, respectively;
- E is a finite set of finite sequences of outputs in O (may include the empty sequence ε);
- h is a behaviour function h: $S \times I \rightarrow 2^{S \times E}$, where $2^{S \times E}$ is the powerset of $S \times E$.

Depending on the properties of the behaviour function, a number of various types of FSM can be defined as follows.

FSM $A = (S, s_0, I, O, E, h)$ is

- *trivial* if $h(s, a) = \varnothing$ for all $(s, a) \in S \times I$;
- *fully specified* (a complete FSM) if $h(s, a) \neq \varnothing$ for all $(s, a) \in S \times I$;
- *partially specified* (a partial FSM) if $h(s, a) = \varnothing$ for some $(s, a) \in S \times I$;
- *deterministic* if $|h(s, a)| \leq 1$ for all $(s, a) \in S \times I$;
- *nondeterministic* if $|h(s, a)| > 1$ for some $(s, a) \in S \times I$;
- *observable* if the automaton $A_d = (S, s_0, I \times E, \delta)$, where $\delta(s, a\beta) \ni s'$ iff $(s', \beta) \in h(s, a)$, is deterministic.

We consider only observable machines; one could employ a standard procedure for automata defeminisation to transform a given FSM into an observable one. Moreover, all the machines are assumed to be initially connected, i.e., each state is reachable from the initial state. We use a, b, c for input symbols, α, β, γ for input and output sequences, s, t, p, q for states, and u, v, w for traces.

Given FSM $A = (S, s_0, I, O, E, h)$, $(s_1, a\beta, s_2)$ is a *transition* if $s_1, s_2 \in S$ and $(s_2, \beta) \in h(s_1, a)$. A *path* from state s_1 to s_{n+1} is a sequence of transitions $(s_1, a_1\beta_1, s_2)(s_2, a_2\beta_2, s_3)\ldots(s_n, a_n\beta_n, s_{n+1})$ such that $(s_{i+1}, \beta_i) \in h(s_i, a_i)$, where $1 \leq i \leq n$ and n is the length of the path. A sequence $u \in (I \times E)^*$ is called a *trace* of FSM A in state $s_1 \in S$, if there exists a path $(s_1, a_1\beta_1, s_2)(s_2, a_2\beta_2, s_3)\ldots(s_n, a_n\beta_n, s_{n+1})$ such that $u = a_1\beta_1a_2\beta_2\ldots a_n\beta_n$. Note that a trace of A in state s_0 is a word of the automaton A_d. Let $Tr(s)$ denote the set of all traces of A in state s and $Tr(A)$ denote the set of traces of A in the initial state.

The *projection operator* $\downarrow B$, which projects sequences in $(I \times E)^*$ onto the set $B \subseteq I \cup O$, is recursively defined as $\varepsilon_{\downarrow B} = \varepsilon$, $(ua)_{\downarrow B} = u_{\downarrow B}a$ if $a \in B$, and $(ua)_{\downarrow B} = u_{\downarrow B}$ otherwise, where $u \in (I \times E)^*$ and $a \in I \cup O$. Given a sequence $u \in (I \times E)^*$, the sequence $u_{\downarrow I}$ is the *input projection* of u. Input sequence $\alpha \in I^*$ is a *defined* input sequence in state s of A if there exists $u \in Tr(s)$ such that $\alpha = u_{\downarrow I}$. We use $\Omega(s)$ to denote the set of all defined input sequences for state s.

Given two states $s, t \in S$ of FSM A and a set of input sequences $Z \subseteq \Omega(s) \cap \Omega(t)$, s and t are Z-*equivalent*, if for all $\alpha \in Z$ it holds that $\{u \in Tr(s) \mid u_{\downarrow I} = \alpha\} = \{u \in Tr(t) \mid u_{\downarrow I} = \alpha\}$. Z-equivalent states are k-*equivalent*, if Z is the set of all input sequences of length k. States s and t are *equivalent* if they are Z-equivalent and $Z = \Omega(s) = \Omega(t)$, i.e., $Tr(s) = Tr(t)$. If $Tr(s) \subseteq Tr(t)$ then s is *trace-included* in t. States s and t that are not Z-equivalent are Z-*distinguishable*. An input sequence $\alpha \in \Omega(s) \cap \Omega(t)$ such that $\{u \in Tr(s) \mid u_{\downarrow I} = \alpha\} \neq \{u \in Tr(t) \mid u_{\downarrow I} = \alpha\}$ is called a sequence *distinguishing* s and t. States s and t are (k-)*distinguishable*, if there exists a sequence distinguishing them (of length k). A set of input sequences Z such that each pair of distinguishable states is Z-distinguishable is called a *characterization* set of FSM A. A complete FSM which has no equivalent states is called *minimal*.

The introduced equivalence and distinguishability relations over states are extended to states of different machines.

2.2. IOTS Model

In this paper, the labelled transition system model is used along with the FSM model, since certain operations on FSMs, such as composition, are simpler to formulate using their transition system counterparts.

A *labelled transition system* (LTS) L is a quadruple (S, s_0, A, λ), where

- S is a set of states with the initial state s_0;
- A is a non-empty set of actions;
- $\lambda \subseteq S \times (A \cup \{\tau\}) \times S$ is the transition relation, with the symbol τ denoting internal actions.

If the set A is partitioned into disjoint sets of input and output actions I and O, then the LTS L is an *input/output transition system* (IOTS).

Given IOTS $L = (S, s_0, I, O, \lambda)$, $(s_1, a, s_2) \in \lambda$ is called a *transition*; (s_1, a, s_2) is *input*, *output* or *internal* transition, if $a \in I$, $a \in O$ or $a = \tau$, respectively. A *path* from state s_1 to state s_{n+1} is a sequence of transitions $(s_1, a_1, s_2)(s_2, a_2, s_3)\ldots(s_n, a_n, s_{n+1})$, such that $(s_i, a_i, s_{i+1}) \in \lambda$, where $1 \le i \le n$.

The projection operator $\downarrow B$, defined for FSM sequences, applies to IOTS sequences in a similar way. It projects sequences of actions in $(I \cup O \cup \{\tau\})^*$ onto the set $B \subseteq I \cup O \cup \{\tau\}$. We also lift the projection operator to IOTS, i.e., given IOTS $L = (S, s_0, I, O, \lambda)$ and set $A \subseteq I \cup O$, the IOTS $L_{\downarrow A}$ is obtained by first replacing each transition $(s_1, a, s_2) \in \lambda$ such that $a \notin A$ by internal transition (s_1, τ, s_2) and then defeminising the obtained IOTS by τ-reduction.

We use $en(s)$ to denote the set of actions enabled in state s, i.e., $en(s) = \{a \in (I \cup O \cup \{\tau\}) \mid \exists t \in S ((s, a, t) \in \lambda)\}$.

Depending on the properties of the transition relation, a number of various types of IOTS can be defined as follows.

IOTS $L = (S, s_0, I, O, \lambda)$ is

- *trivial* if $en(s) = \varnothing$ for each $s \in S$;
- *deterministic* if it has no internal transitions and λ is a function from $S \times (I \cup O)$ to S;
- *conflict-free* if input actions are only enabled in stable states; state $s \in S$ is *stable* if no output or internal actions are enabled in s, i.e., $en(s) \cap (O \cup \{\tau\}) = \varnothing$, otherwise it is *unstable*.
- A conflict-free IOTS is *fully specified* if all input actions are enabled in each stable state and *partially specified* otherwise.
- A deterministic conflict-free IOTS $L = (S, s_0, I, O, \lambda)$ is *output-deterministic* if for all $s \in S$, $en(s)$ contains at most one output action; otherwise it is *output-nondeterministic*.

State $s \in S$ is a *deadlock* if no action is enabled in it, i.e., $en(s) = \varnothing$. State $s \in S$ is a *livelock* if there is a cycling path of output or internal transitions that includes s; if the path includes only internal transitions then livelock is *internal*, otherwise it is an *output* livelock. IOTS L is *deadlock-free* or *livelock-free*, if there is no deadlock or livelock state reachable from the starting state, respectively.

Hereafter we consider a subclass of transition systems, denoted $IOTS_{\text{basic}}(I, O)$, which includes finite, deterministic, fully specified, conflict-, deadlock- and livelock-free IOTSs over the input and output action sets I and O, each of which has a stable initial state.

A sequence $u \in (I \cup O)^*$ is called a *trace* of IOTS L in state $s_1 \in S$ if there exists a path $(s_1, a_1, s_2)(s_2, a_2, s_3)\ldots(s_n, a_n, s_{n+1})$, such that $u = (a_1\ldots a_n)_{\downarrow(I \cup O)}$. Similar to FSM, we use $Tr(s)$ to denote the set of traces in state $s \in S$, while $Tr(L)$ to denote the set of traces of L in the initial state.

For the IOTS class $IOTS_{\text{basic}}(I, O)$ we introduce additional definitions. Let $L \in IOTS_{\text{basic}}(I, O)$. A path $(s_1, a_1, s_2)(s_2, a_2, s_3)\ldots(s_n, a_n, s_{n+1})$ between two stable states s_1, and s_{n+1} of L is called *simple* if all the intermediate states $s_2, s_3, \ldots, s_n$ are unstable, thus a_1 is input action, while $a_2, \ldots a_n$, if $n > 1$, are output actions. We use $(s, a\beta, t)$ to denote a simple path from stable state s to stable state t. Then an arbitrary path from stable state s_1 to stable state s_{n+1} can be represented as a sequence of simple paths $(s_1, a_1\beta_1, s_2)(s_2, a_2\beta_2, s_3)\ldots(s_n, a_n\beta_n, s_{n+1})$, it defines a *stable* trace $a_1\beta_1a_2\beta_2\ldots a_n\beta_n$. Stable traces are, in fact, suspension traces [Tre96], though in this paper, we do not use explicit quiescence output action. We use $STr(s)$ to denote the set of stable traces of L in s and $STr^k(s)$ to denote the set of stable traces, each of which has at most k input actions, i.e., $STr^k(s) = \{u \mid u \in STr(s) \wedge |u_{\downarrow I}| \le k\}$.

Given two stable states $s, t \in S$ of IOTS $L \in IOTS_{\text{basic}}(I, O)$ and a set of input sequences $Z \subseteq I^*$, s and t are *Z-equivalent*, if for all $\alpha \in Z$ it holds that $\{u \in STr(s) \mid u_{\downarrow I} = \alpha\} = \{u \in STr(t) \mid u_{\downarrow I} = \alpha\}$. Z-equivalent states are *k-equivalent*, if Z is the set of all input sequences of length k. Stable states s and t are *equivalent* if they are I^*-equivalent, i.e., $STr(s) = STr(t)$, otherwise they are distinguishable. Stable states that are not Z-equivalent are *Z-distinguishable*.

2.3. Relating FSM and IOTS

Comparison of the two models immediately reveals that trivial FSM and trivial IOTS do not differ; so now we consider models where inputs label at least one transition. Any transition system in $IOTS_{\text{basic}}(I, O)$ behaves as an FSM; it produces all due outputs in response to input before the environment offers a next input, similar to FSM, where input along with an output sequence produced in response to it constitute an atomic action, thus a single transition. The set of its stable traces coincides with the set of traces of a corresponding FSM. On the other hand, any fully specified FSM can be converted into an IOTS in $IOTS_{\text{basic}}(I, O)$, once each of its transitions is unfolded into input transition, followed, if the output of the transition is not the empty sequence, by output transitions; their number is defined by the length of the output sequence.

The above discussion leads to the following statement.

Theorem 1. Let I be input set and O be output set, then

- For any fully specified FSM A over I and O with the set of traces $Tr(A)$, there exists an IOTS L in $IOTS_{\text{basic}}(I, O)$ such that $Tr(A) = STr(L)$.
- For any IOTS L in $IOTS_{\text{basic}}(I, O)$ with the set of stable traces $STr(L)$, there exists FSM A over I and O such that $Tr(A) = STr(L)$.

Theorem 1 allows one to convert a given FSM into an IOTS and vice versa, preserving the traces. This makes the existing state minimization methods developed for FSM, see, e.g., [KVB97] fully applicable to the class of transition systems $IOTS_{\text{basic}}(I, O)$. In this paper, we use both models; though for technical reasons, we will use the FSM model for inferring a single model of an SUT and the IOTS model for inferring and verifying its components.

3. Inferring a State Model of a System Under Test

In this section, we introduce the concept of an initial Z-quotient of a given FSM and give the algorithm for its inference by testing. Finally, we discuss how the assumptions about an SUT can be relaxed to reflect more realistic situations when the SUT does not necessarily behave as an FSM.

3.1. Initial Z-Quotient

Assume we are given an SUT which behaves as an FSM (thus as an IOTS in the above defined class) and we can perform experiments with it by applying inputs and observing outputs to infer an FSM model. To test the SUT we need to know at least its input alphabet I. The inference can easily be performed assuming that the black box behaves as a deterministic machine and the number of distinct states n as well as a characterization set is known. Since each input can be applied to the SUT if needed, the machine is also fully specified. To construct an FSM model it is sufficient to adapt the W-method [Vas73] as follows. The set $I^{<n}$ of input sequences of length at most $n - 1$ allows us to reach all of the n states. The state identification is achieved with the help of the characterization set; all the states are identified as soon the characterization set applied after n different traces produced by the SUT in response to input sequences from $I^{<n}$ yields n different sets of traces. The transitions between the states are inferred using again the characterization set to identify the tail states of transitions, exactly as it is done according to the W-method. The actual machine may have equivalent states, but the inferred machine does not, it is its minimal form, i.e., the quotient implied by the state equivalence, which is confirmed by the known characterization set. The existence of a characterization set greatly simplifies the inference process; however, the problem is that a characterization set may seldom be given for inference.

Therefore, when no characterization set is known for testing, we need to find a way of inferring an approximated model with a controllable precision. To infer not the minimal form of the FSM but some approximation in the form of a quotient, we can use instead of a characterization set an arbitrary set Z of input sequences. This set defines the Z-equivalence relation between states of the unknown FSM, so states of this machine could be identified modulo Z-equivalence. The idea leads to the following definition and eventually to an inference algorithm.

Given a complete FSM $A = (S, s_0, I, O, E, h)$ and a finite non-empty set of input sequences $Z \subseteq I^*$, let π_z be the partition on the set of states S induced by the Z-equivalence relation. For state s, all of the states that are Z-equivalent to state s constitute the equivalence class $\pi_z(s)$. It is known that a state equivalence relation induces a quotient model of the original machine, see, e.g., quotient model of Kripke structure [CGP99]. The idea, which can be traced back to work in [Ner58] and [BF72] is to collapse all Z-equivalent states (k-equivalent in [BF72]) while preserving all

transitions. The obtained model preserves all the traces of the original machine, but contains additional traces.

Formally, a *Z-quotient model of* a complete FSM $A = (S, s_0, I, O, E, h)$ is an FSM $K = (Q, q_0, I, O, E, k)$, where each state q_i is an equivalence class π_{z_i} of the partition π_z, $q_0 = \pi_z(s_0)$, and for any $\pi_{z_i} \in Q$ and $a \in I$, $k(\pi_{z_i}, a) = \{(\pi_z(t), \beta) \mid (t, \beta) \in h(s, a) \wedge s \in \pi_{z_i}\}$.

Consider the example of an FSM A in Figure 1. Assume $Z = \{a\}$. Then, in A, s_1 and s_2 are the only Z-equivalent states. The FSM K is the Z-quotient of A, in which $q_1 = \{s_1, s_2\}$. We can see in this example that although A is deterministic, the Z-quotient is nondeterministic, if some states are Z-equivalent but nevertheless distinguishable.

(a) FSM A (b) FSM K as $\{a\}$-quotient of A

Figure 1: FSM A and its $\{a\}$-quotient

Quotient models are widely used in model checking, however, for inference purposes we are constrained to a single representative of each equivalence class π_z. The reason is that once a distinct Z-distinguishable state is identified it should be included into the inferred model. This constraint leads to the following definition.

Definition 1. Given a complete FSM $A = (S, s_0, I, O, E, h)$ and a set of input sequences Z, an FSM $K = (Q, q_0, I, O, E, k)$ is an *initial*[1] *Z-quotient model of* A, if there exists an injection f from Q to S such that

- $f(q_0) = s_0$
- for any two distinct states $q_1, q_2 \in Q$, $f(q_1)$ and $f(q_2)$ are Z-distinguishable
- for any $q \in Q$ there exists a path $(s_0, a_1\beta_1, s_1)\ldots(s_{n-1}, a_n\beta_n, s_n)$, such that $s_i = f(q_i)$, $q_i \in Q$, $1 \leq i \leq n$ and $s_n = f(q)$
- for any $q \in Q$ and $a \in I$, $\beta \in O^*$, $(p, \beta) \in k(q, a)$ iff there exists $s \in S$, such that $(s, \beta) \in h(f(q), a)$ and s and $f(p)$ are Z-equivalent.

In Figure 2, an initial Z-quotient of FSM A is depicted, in which $Z = \{a\}$, $f(q_0) = s_0$ and $f(q_1) = s_1$. Note that the initial Z-quotient is deterministic.

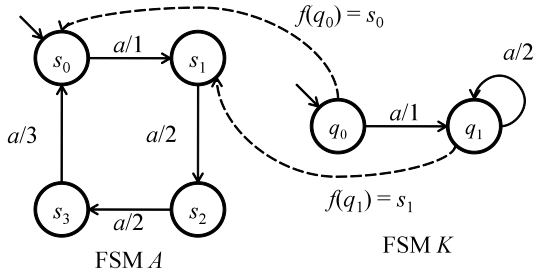


Figure 2: Initial Z-quotient of FSM A

In Figure 3, an FSM L is provided, we want to check whether L is an initial Z-quotient of A, assuming as before $Z = \{a\}$. Suppose there is an injection f from Q to S, $f(q_0)$ must be s_0. $f(q_2)$ can only be equal to s_3, since $k(q_2, a) = \{(q_0, 3)\}$. Now, we need to determine $f(q_1)$. If $f(q_1) = s_1$, according to the last clause of the definition, s_2 and s_3 should be Z-equivalent, which is not the case, they produce different outputs. If $f(q_1) = s_2$, the third clause is not satisfied. We conclude that there is no injection f from Q to S with the required properties, thus, L is not an initial Z-quotient of A.

[1] We use "initial" in the introduced term of initial Z-quotient to emphasize the fact that the latter represents a part reachable from the initial state of a given FSM modulo Z-equivalence. It is not related to the usage of "initial" such as "initial object" in category theory.

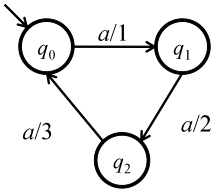


Figure 3: An FSM L which is not initial Z-quotient of FSM A

Comparing the two models, Z-quotient and initial Z-quotient, we may identify several differences. One difference is that given an FSM, an equivalence class is not represented in the initial Z-quotient if each path leading the FSM to any state in that class traverses several Z-equivalent states. Another difference is that any FSM has a unique Z-quotient, but may have several initial Z-quotients. As Figure 1 and Figure 2 illustrate, Z-quotient of a deterministic FSM could be nondeterministic, while its initial Z-quotients always remain deterministic.

There is special case of a Z-quotient, when the set Z is a characterization set of a given FSM A. In this case, as any two states of A could be distinguished by a string in Z, as a result, each state of the given machine is represented by a separate state in the quotient, which thus becomes equivalent to A. Recall that in the case of Figure 2, where $Z = \{a\}$ is not a characterization set of A, the FSM K and A are not equivalent. This observation leads to the following theorem about properties of initial Z-quotients.

Theorem 2. Given a Z-quotient K of a complete FSM A, if Z is a characterization set of A, then FSM A and K are equivalent; otherwise, if A has distinguishable but Z-equivalent states, then A and K are distinguishable.

The proof is given in Appendix A.

The more sequences of a characterization set are included into the set Z the more precise the approximation. The precision of Z-quotient of an FSM can be controlled by the parameter Z, henceforth called the *inference* parameter.

Given a natural k, a Z-quotient is called a *k-quotient* if $Z = I^k$. It is known that the set I^k contains distinguishing sequences for any pair of states in any FSM over the input alphabet I with at most $n = k + 1$ states, hence it is a characterization set of such machines.

The case of $k = n - 1$ corresponds actually to worst case situations, which occur in special pathological machines, such as Moore locks. For other types of machines k-equivalence of states becomes state equivalence for much lower value of k and thus it may be more appropriate to consider instead of upper bounds asymptotic characterization of FSM parameters for "almost all FSMs". The monograph [TB73] indicates that for complete FSM with n states, m inputs, and l outputs, the length of input sequences reaching all n states is asymptotically equal to $\log_m n$ and distinguishing states just $\log_m \log_l n$. These results suggest that even when the value of the parameter k is well below the actual number of states of a given FSM the approximation of the FSM in the form of a k-quotient might be sufficiently precise and thus acceptable for practical applications. This also means that choosing a set Z, one is not obliged to focus on long input sequences, but rather on those which can discriminate various operational modes of an SUT and thus its internal states.

For completeness we also provide the definition of an initial Z-quotient for an IOTS in $IOTS_{\text{basic}}(I, O)$ which is derived from that for FSM and Theorem 1.

Definition 2. Given an IOTS $L = (S, s_0, I, O, \lambda)$ with the set of stable states S_{stable}, $L \in IOTS_{\text{basic}}(I, O)$ and a set of input sequences Z, an IOTS $K = (Q, q_0, I, O, \mu)$, $K \in IOTS_{\text{basic}}(I, O)$ with the set of stable states Q_{stable} is an *initial Z-quotient model of L*, if there exists an injection f from Q_{stable} to S_{stable} such that

- $f(q_0) = s_0$
- for any two distinct states $q_1, q_2 \in Q_{\text{stable}}$, states $f(q_1), f(q_2) \in Q_{\text{stable}}$ are Z-distinguishable
- for any $q \in Q_{\text{stable}}$ there exists a path $(s_0, a_1\beta_1, s_1)\ldots(s_{n-1}, a_n\beta_n, s_n)$ with the stable trace $a_1\beta_1\ldots a_n\beta_n$, such that $s_i = f(q_i)$, $q_i \in Q$, $1 \le i \le n$ and $s_n = f(q)$
- for any $q, p \in Q_{\text{stable}}$, $a \in I$, $\beta \in O^*$, there exists a simple path $(q, a\beta, p)$ iff there exists a simple path $(f(q), a\beta, s)$, such that s and $f(p)$ are Z-equivalent.

When the set of input sequences Z is clear from the context, we call an initial Z-quotient of FSM A just an *initial quotient* or simply a *quotient* of FSM A, and similarly for IOTS.

3.2. Inferring Z-Quotient of the SUT

We assume that a given SUT viewed as a black box can be modelled by a fully specified and deterministic FSM (or an IOTS) over the input I and output E sets. We also assume that a reliable reset can be performed on the SUT, so that several test sequences can be applied to the SUT from

its initial state. As usual, for testing we are restricted to controllable and observable actions, via interfaces or ports open for testing; namely let the set of controllable inputs be $I_{con} \subseteq I$ and the subset of observable outputs be $E_{obs} \subseteq E$. If some action of the given FSM is either non-controllable input or unobservable output then by testing we can infer a model which represents only a corresponding part of the given machine. This constraint motivates the following definition.

Definition 3. Given controllable inputs $I_{con} \subseteq I$ and observable outputs $E_{obs} \subseteq E$ for an FSM A, an FSM which is obtained from A by removing all transitions labelled with uncontrollable inputs and removing each non-observable output in the output sequences of the remaining transitions is called a *testable model of the FSM A*. Similarly, an IOTS obtained from L by removing all transitions with uncontrollable inputs and replacing all transitions with non-observable outputs by internal transitions is called a *testable model of the IOTS L*.

We further assume that we are given a set of input sequences Z needed to infer a Z-quotient of a testable model A by testing its implementation, the SUT. A basic idea of our inference method which directly follows from the clauses of the definition of Z-quotient is as follows:

- We start building an initial Z-quotient by including an initial state which could be injected to the initial state of A; the remaining states should not be Z-equivalent to those already in the quotient;
- We explore the states of A from the initial state step by step. For each visited state, we decide whether to include a corresponding state in the initial Z-quotient. If the current state is Z-equivalent to a state already visited then we do not explore states of A further from this state.
- For each state, the transitions are defined respecting the Z-equivalence.

This basic idea is described in more detail in the following algorithm. To represent the observed traces, we use a tree FSM.

Definition 4. Given a (prefix closed[2]) set U of observed traces of an FSM over input set I and output set O, the observation tree is FSM $(U, \varepsilon, I, O, E_U, h_U)$, where the state set is U, $E_U = \{\beta \in O^* \mid \exists u \in U, \exists a \in I\ (ua\beta \in U)\}$, and $h_U(u, a) = \{(ua\beta, \beta) \mid \exists \beta \in O^*\ (ua\beta \in U)\}$.

We use U to refer to both, a prefix-closed set of FSM traces and the corresponding observation tree FSM $(U, \varepsilon, I, O, E_U, h_U)$.

The quotient inference method includes two phases: first constructing an observation tree while identifying all the quotient's states and then determining transitions between the states. In the state identification phase, we apply inputs to an SUT, observe traces and add them to an observation tree FSM U. We perform Breadth First Search (BFS) on the tree and if the current state u is Z-distinguishable with each already traversed state in the observation tree FSM, we add the state u into the set of states of a quotient, which is initialized with $\{\varepsilon\}$. Otherwise, if there exists a traversed state w which is Z-equivalent to u, we label the state u with w, i.e., $label(u) = w$, u is not included into the states of a quotient, and the behaviour of the FSM A will no longer be explored from the state u. Once the tree stops growing, all the states of a quotient are identified. Transitions between the states of the quotient are determined from the transitions of the observation tree. Namely, a transition is considered if neither the source state nor any of its predecessors is labelled in the tree. If the target state is not labelled either, the same transition exists in the quotient. Otherwise the transition is redirected to the state which is used to label the target state.

The inference algorithm is formalized as follows.

Input:
The set of inputs I of an unknown testable FSM A;
the inference parameter Z.
Output:
An FSM $K = (Q, q_0, I, O, E, k)$.

1. $U := \{\varepsilon\}, q_0 := \varepsilon, Q := \{q_0\}$
2. **for** (each state u of U being traversed during Breadth First Search)
3. **begin**
4. **if** (u has no labelled predecessor)
5. **begin**
6. **while** (there exists $a_1a_2\ldots a_k \in (I \cup Z)$ such that $a_1a_2\ldots a_k \neq v_{\downarrow I}$, where $k \geq 1$, for any $v \in Tr(u)$) **do**
7. **begin**
8. Reset A to its initial state
9. Apply $u_{\downarrow I}$ to A

[2] Recall that a symbol of an FSM trace is a pair of an input from I and a sequence of outputs from O, so every prefix takes an FSM from its initial state into some state.

10. Apply inputs a_1, a_2, …, a_k to A, let the corresponding observed output sequences be $\beta_1, \beta_2, \ldots, \beta_k$
11. Add the trace $ua_1\beta_1a_2\beta_2\ldots a_k\beta_k$ and its prefixes $ua_1\beta_1$, $ua_1\beta_1a_2\beta_2$, …, $ua_1\beta_1a_2\beta_2\ldots a_{k-1}\beta_{k-1}$ to U
12. **end**
13. **if** (u is Z-equivalent to a traversed state w of U)
14. Label u with w, i.e., $label(u) = w$
15. **else**
16. Add u into Q
17. **end**
18. **end**
19. **for** (each transition $h_U(u, a) = (v, \beta)$, such that neither state u nor any of its predecessors is labelled)
20. **begin**
21. **if** (v is not labelled)
22. Add transition $k(u, a) = (v, \beta)$ to K
23. **else if** ($label(v) = w$)
24. Add transition $k(u, a) = (w, \beta)$ to K.
25. **end**
26. $O_K = O, E_K = E_U$.
27. Return the resulting FSM K as an initial Z-quotient of the FSM A.

We illustrate the procedure by inferring an $\{a\}$-quotient of FSM A in Figure 4.

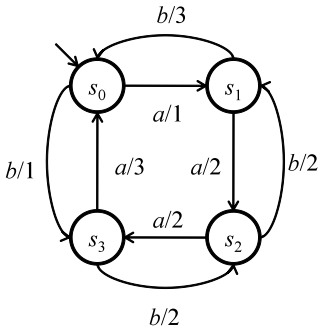


Figure 4: An FSM A

Initially, $U = Q = \{\varepsilon\}$. We have $(I \cup Z) = \{a, b\}$; we use the inputs a and b to explore the behaviour of FSM A from the state ε. The output 1 is observed in both cases and the tree becomes $U = \{\varepsilon, a1, b1\}$. We apply then a and b to state $a1$ and observe output 2 and 3, respectively. Then $U = \{\varepsilon, a1, b1, a1a2, a1b3\}$. We determine that state $a1$ is not $\{a\}$-equivalent to state ε, so state $a1$ is added into Q. Similarly, state $b1$ is added into Q and the tree becomes $U = \{\varepsilon, a1, b1, a1a2, a1b3, b1a3, b1b2\}$. When we apply a and b to state $a1a2$ and observe the output 2 in both cases, we determine that state $a1a2$ is $\{a\}$-equivalent to state $a1$. State $a1a2$ is labelled with $a1$ and we stop exploring further from this state. Similarly, we label $a1b3$, $b1a3$, $b1b2$ and we stop exploring from those states. The final observation tree U is depicted in Figure 5, and the set of states of a quotient is $Q = \{\varepsilon, a1, b1\}$.

Transitions between the initial state and states $a1$ and $b1$ are also transitions in the resulting quotient, while the target states of transitions from states $a1$, $b1$ are defined by the labels in the tree, e.g., the transition from $a1$ to $a1a2$ is redirected to state $a1$ in the quotient, shown in Figure 6.

The following theorem claims that the above method can be used to infer an initial Z-quotient of the FSM from which traces are collected during testing.

Theorem 3. If the inference method is applied to a deterministic FSM A and yields an FSM K, then FSM K is an initial Z-quotient of FSM A.

The proof is provided in Appendix B.

The above algorithm can also be used to infer an initial Z-quotient model of an IOTS; in the state identification phase, the identified states are stable states and the transitions between them represent simple paths.

3.3. Discussions

In this section, we discuss how the assumptions made about an SUT can be relaxed.

To infer an initial **Z**-quotient of a testable model for a given SUT our basic assumption is that the SUT can be modelled by an FSM. Since our FSM model may produce not only single outputs

in response to a single input, but multiple outputs in a row, by assuming a finite state model we, in fact, require that the SUT in hand never produce unbounded output sequences. If the SUT is treated as a black box, we need to check this property by testing. Thus, executing tests according to the proposed inference method against a real SUT, it is sufficient to set a bound on the allowed number of outputs produced by the SUT in response to a single test input. As soon as the length of any observed output sequence exceeds the given bound, the testing process may terminate. This equally applies to checking the absence of output livelock, one of basic assumptions used when an initial Z-quotient for a given SUT has to be inferred in the form of an IOTS.

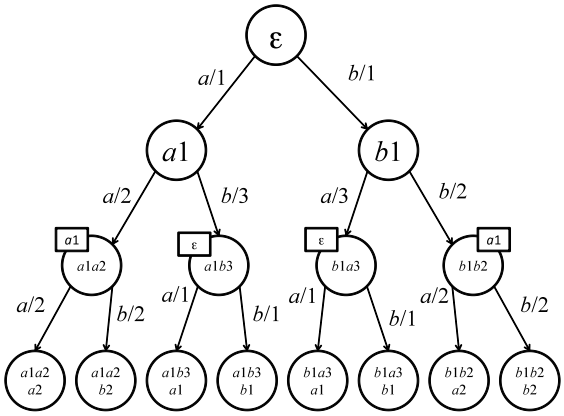


Figure 5: Observation Tree *U*

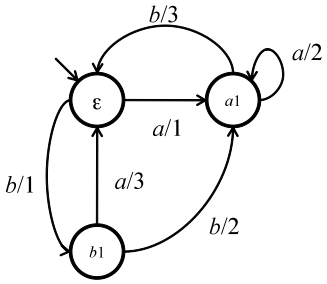


Figure 6: Initial {a}-Quotient *K* of FSM *A*

Another assumption about an IOTS model of the SUT is absence of internal livelock. When a test input is applied to a real SUT one has to use a timer to conclude that in response the SUT has not produced any observable output. However, it is difficult, if not impossible, to detect by testing that an IOTS modelling the SUT has an internal livelock. We may, however, conclude that the SUT had no internal livelock if after some test input it remains quiescent until a timeout expires and it can produce observable outputs in response to subsequent test inputs. This means that non-catastrophic livelock can still be present in an SUT and will not be represented in the inferred model, but catastrophic one which blocks the system terminates testing. The absence of observable outputs does not result in any output transition if the IOTS model is used and it yields the empty output sequence for a corresponding transition in the FSM model.

Our next assumption about the SUT is that if an FSM is its model, then it is a complete FSM; and if an IOTS is used then it is deadlock-free and fully specified. Indeed, the assumption that the SUT is ready to accept any input in any state is justified in many situations: FSM by definition does not "refuse" any input, and in IOTS a deadlock state cannot be observationally distinguished from a stable state where all inputs cause looping transitions. Nevertheless, it is possible to consider a partially specified FSM and IOTS models and adjust the results of previous sections to such models. The first adjustment we need is to assume that the set of defined input sequences of a testable model of the SUT is known. It should represent in fact the behaviour of the environment of the SUT and can be specified as an automaton. We also need to require that the set Z used in the Z-quotient contains only input sequences which are defined in every state of a testable model. We also notice that a model of a constrained environment helps reduce the number of input sequences used during the exploration phase. In fact, Steps 6-11 of the above procedure require application of all possible controlled inputs in each newly reached state, modelling the unconstrained environment. Since this may be sometimes costly, a judicious choice of controllable inputs may reduce the exploration, however, a more radical save can be achieved once a constrained environment for model inference is used.

Another assumption about models of a given SUT used in the inference method is determinism. In particular, we require that its testable FSM model be deterministic and its IOTS model be output-deterministic. The definition of initial **Z**-quotient is given for an observable (not necessarily deterministic) FSM, thus one can infer by testing Z-quotients which are nondeterministic FSMs or equivalently output-nondeterministic IOTSs. Theoretically, it is feasible, but only with the help of an additional assumption about the fairness of a nondeterministic system known as "all weather", aka "complete testing", assumption [LBP94][EGI10].

Finally, our approach for inferring a quotient assumes that the tester predefines a set of input sequences which allow one to distinguish distinct states of a testable model of the SUT. From the theoretical perspective, ideally this has to be its characterization set; if we wish to infer a precise model, which is the testable model itself. From a more practical viewpoint, it might be sufficient for this set to possess the discriminative power of identifying the most important states of the testable model, needed to represent a feature of the SUT targeted by the verification efforts. Z-equivalent states of the testable model do not change the behaviour of the SUT executing the chosen feature and thus are collapsed together in the quotient.

The approach to the model inference problem elaborated in this section can be used in an iterative way to increase the precision of the resulting model by augmenting the set of input sequences used to discriminate the states. In particular, when two states of the observation tree are Z-equivalent, but 1-distinguishable, an input which distinguishes them could be added to the set Z. On the other hand, using as an inference parameter a predefined discriminative input set instead of a predefined bound on the state number (as, e.g., in [PVY99]) offers new possibilities, such as inference of a quotient focusing on a part of the system of interest. State distinguishability achieved by the discriminative input set provides a useful characterization of states of the system which are reflected in the inferred model.

Concluding our discussion of the proposed approach for inferring a quotient of a testable model of a given system, we notice that the definition of a quotient as well as a procedure for its construction assume a universal environment, which can submit any controllable input to any system's state. At the same time, there might be applications where the environment is constrained and may not submit all the controllable inputs to every state. On the other hand, the tester may use such environment to specify its assumptions about transfer sequences which are expected to take the system into "most interesting" states which should be included into a quotient. The proposed approach can further be generalized by incorporating a model of a restricted environment, in the form of, e.g., an automaton whose language is the set of input sequences which could be submitted to a given system.

4. Modular System Verification by Inference, Testing, and Reachability Analysis

In this section, we first give some basic definitions related to modelling modular systems, where components use queues to communicate and then present an approach to verify a modular SUT with unknown components. The approach relies on the inference of a quotient of a system elaborated in the previous section; we illustrate it on an example and conclude by discussing how various simplifying assumptions about the SUT can be relaxed.

4.1. Basic Definitions

We consider a reactive modular system consisting of components communicating asynchronously, where each component reads inputs from its input queue and writes outputs to other components' input queues. Modelling queues, we distinguish the same action at the two ends of a queue by using the relabeling ' operator [HP09]. The operator is defined on input actions: for $a \in I$, $(a)' = a'$, and $(a')' = a$. It is lifted to the sets of input actions, traces, and IOTS: for an action set I, $I' = \{a' \mid a \in I\}$; for traces, ' is recursively defined as $\varepsilon' = \varepsilon$ and $(ua)' = u'a'$ for trace u if a is an input action, otherwise $(ua)' = u'a$; L' is obtained from L by relabeling each action $a \in I$ to a'. Then a (unbounded) *queue with input set I*, is an IOTS $\langle I^*, \varepsilon, I, I', \lambda_I \rangle$, denoted Q_I, where I^* is the state set and $\lambda_I = \{(u, a, ua) \mid u, ua \in I^*\} \cup \{(av, a', v) \mid av, v \in I^*\}$ is the transition relation. The only stable state of a queue is its initial state.

Let $C = \{C_1, \ldots, C_n\}$ be a system of communicating components, where $C_i = (S_i, s_{0i}, I_i, O_i, \lambda_i)$ is a finite IOTS with the stable initial state s_{0i}, such that $I_i \cap I_j = \varnothing$ and $O_i \cap O_j = \varnothing$ for $i \neq j$. For each component C_i with the input set I_i, there is an unbounded input queue $Q_{I_i} = (I_i^*, \varepsilon_i, I_i, I_i', \lambda_{I_i})$. Outputs of the queue are inputs of the component, we thus use $C_i' = (S_i, s_{0i}, I_i', O_i, \lambda_i)$ instead of C_i

= (S_i, s_{0i}, I_i, O_i, λ_i) when composing a component with its input queue. The *architecture* of the system is defined as a directed graph, $A = (C, D)$, where C is the set of nodes and D is the set of directed edges, called *connections*, such that $(C_i, C_j) \in D$ if $O_i \cap I_j \neq \varnothing$.

Let I and O denote the union of all input action sets and output action sets of the components and queues, respectively. The set $I_{ext} = I \setminus O$ contains *external inputs*; components constitute a *closed* system, if I_{ext} is empty, otherwise an *open* system. Let $O_{ext} = O \setminus I$ be the set of *external outputs* of the system, while $I \cap I_{ext}$ be that of *internal* actions. In this paper, we consider only an open system with at least one external output, such that for each component C_i it holds that if $a \in I_i$ then either $a \in I_{ext}$ or $a \in O_j$ and if $a \in O_i$ then $a \in O_{ext}$ or $a \in I_j$ for some C_j.

The behaviour of an open system usually depends on the speed of its environment. We distinguish two types of the environment, fast and slow. A *fast* environment can supply external inputs at any state of the system. A *slow* environment does so only when the system is in a stable global state.

The behaviour of the system operating in the fast environment is described by the IOTS $C_1' \parallel \ldots C_n' \parallel Q_{I_1} \parallel \ldots \parallel Q_{I_n}$, where $\parallel$ is the standard LTS parallel composition operator, $C_i' = (S_i, s_{0i}, I_i', O_i, \lambda_i)$, for $C_i = (S_i, s_{0i}, I_i, O_i, \lambda_i)$, and $Q_{I_i} = (I_i^*, \varepsilon_i, I_i, I_i', \lambda_{I_i})$. To describe the behaviour caused by the slow environment, we modify the composition to allow external inputs only in stable global states as follows.

Definition 5. Given a system of communicating components $C = \{C_1, \ldots, C_n\}$ and the set of queues over input alphabets of the components $Q = \{Q_{I_1}, \ldots, Q_{I_n}\}$, the *slow asynchronous product* (or simply product) of C, denoted Π, is the IOTS $(R, s_{01}...s_{0n}\varepsilon_1...\varepsilon_n, I_{ext}, O, \lambda)$, where $I_{ext} = I \setminus O$, $I = I_1 \cup \ldots \cup I_n \cup I_1' \cup \ldots \cup I_n'$ and $O = I_1' \cup \ldots \cup I_n' \cup O_1 \cup \ldots \cup O_n$; the set of states $R \subseteq S_1 \times ... \times S_n \times I_1^* \times ... \times I_n^*$ and the transition relation λ are the smallest sets obtained by applying the following inference rules:

- $s_{01}...s_{0n}\varepsilon_1...\varepsilon_n \in R$;
- if $a \in I_{ext} \cap I_i$, $(s_1...s_n\varepsilon_1...\varepsilon_n) \in R$ such that states s_1, ..., s_n are stable, then $(s_1...s_n\varepsilon_1\ldots\varepsilon_n, a, s_1...s_nb_1\ldots b_n) \in \lambda$ and $(s_1...s_nb_1\ldots b_n) \in R$ such that $b_i = a$, and $b_j = \varepsilon_j$ for $j \neq i$ (external input is buffered into the queue of the component C_i);
- if $a \in I_i$, $(s_1...s_nb_1\ldots b_n) \in R$ such that $a \in en(s_i)$, $b_i = av$, then $(s_1...s_nb_1\ldots b_n, a', s_1'...s_n'c_1\ldots c_n) \in \lambda$ and $(s_1'...s_n'c_1\ldots c_n) \in R$, such that $(s_i, a, s_i') \in \lambda_i$, and $c_i = v$; $s_j' = s_j$ and $c_j = b_j$ for $j \neq i$ (input is consumed from a queue, and input transition of C_i is executed);
- if $a \in O_i$, $(s_1...s_nb_1\ldots b_n) \in R$, such that $a \in en(s_i)$, then $(s_1...s_nb_1\ldots b_n, a, s_1'...s_n'c_1\ldots c_n) \in \lambda$ and $(s_1'...s_n'c_1\ldots c_n) \in R$, such that $(s_i, a, s_i') \in \lambda_i$, $s_j' = s_j$ for all $j \neq i$, and if $a \in I_j$ then $c_j = b_ja$, otherwise, $c_j = b_j$ (output transition of C_i is executed, and output is buffered into the queue of the component, for which it is an input).

Notice that all components' inputs, save external ones, become outputs of the composition along with external outputs, as is usually the case for input-output automata composition [LT89].

Definition 6. A system C has

- *unspecified reception*, if in the product Π, there exists a state $(s_1...s_nb_1\ldots b_n)$ such that for some component C_i, a is a prefix of b_i and $a \in I_i$, but $a \notin en(s_i)$;
- *compositional livelock*, if Π has output livelock;
- *divergence*, if Π is not a finite IOTS;
- *races*, if the component IOTSs are output-deterministic and there exist traces $\alpha, \beta \in Tr(\Pi)$ such that $\alpha_{\downarrow I_{ext}} = \beta_{\downarrow I_{ext}}$ and $\alpha_{\downarrow O_{ext}} \neq \beta_{\downarrow O_{ext}}$;
- *unused connection*, if there exists a connection $(C_i, C_j) \in D$ such that in the product Π, for each action $a \in O_i \cap I_j$, there is no transition labelled with a;
- *isolated component*, if there exists a component such that all its connections are unused.

The first four properties, namely, unspecified reception, compositional livelock, divergence, and races, indicate compositional problems in the system. Indeed, unspecified reception causes compositional deadlock, compositional livelock results in unbounded output sequences in the composition, divergence causes buffer overflow in a system with bounded queues, while race indicates that the system even if it is composed of output-deterministic components may exhibit a nondeterministic behaviour. The other two, the existence of unused connections and isolated components, point to redundancy in the system and do not affect the behaviour of the system, thus they are less critical. For example, isolated components which have thus only external inputs and outputs can be excluded from the given system and analysed independently. By this reason, we call a system *well-formed* if it has no unspecified reception, compositional livelock, divergence, and races. The number of unused connections may characterize the quality of the models used to represent an SUT, as we discuss later in Section 4.4.

Given a system of communicating components, the slow asynchronous product can be constructed (if it is finite, of course) and, thus, its well-formedness can be checked using a classical reachability analysis (RA) procedure which we will not discuss further in this paper. We simply assume that given a system $C = \{C_1, C_2, \ldots, C_n\}$ with bounded queues $Q = \{Q_{I_1}, \ldots, Q_{I_n}\}$ of known sizes, the procedure either confirms that the system is well-formed or outputs the following *witness* (diagnostic) traces for:

- unspecified reception of $a \in I_i$, a trace βa, such that β takes the product into a state, where the action a is not enabled in the corresponding state of some component C_i whose input queue contains just a;
- compositional livelock, a trace $\alpha\beta$, such that α takes the product into a state of a cycle labelled by the sequence β;
- divergence, a trace which causes violation of the size of a queue in the system[3].
- races, two traces with a common external input projection which leads to races[4].

A reachability analysis procedure can also detect any unused connections and isolated components in the given system.

4.2. Verification Problem and the Approach

Given a modular SUT consisting of components communicating asynchronously each of which can be modelled as a complete deterministic FSM, and thus as a corresponding IOTS, we want to verify the well-formedness of the system, assuming that precise models for some of them may be known. Clearly, if all the components' models are available then the problem can be solved using a RA procedure. Thus, at least one model is assumed to be missing.

We assume that the SUT architecture is known and has no unused connections; the system with isolated components can always be decomposed into connected subsystems. The knowledge of the SUT architecture means that in testing all internal output actions could be observable, e.g., by intercepting each message in transit, while only external input actions are controllable by the tester.

To infer missing models needed for checking the well-formedness of the SUT one could infer Z-quotients one by one by testing unknown components in isolation following the method of Section 3. This approach would require defining an appropriate value of the inference parameter Z for each component. The sets of input sequences needed for quotients inference would then differ for various components, and it is unclear how one could define them without taking into account the components' interactions. A trial-and-error approach can be followed, and even if it succeeds, it may result in "redundant" models of components which describe functionalities unused in a given system.

Another approach elaborated in this paper relies on the components' interactions and is summarized in Figure 7. The idea is to verify a modular SUT by first inferring a Z-quotient of a testable model of the slow asynchronous product and thus simultaneously inferring missing models of its components or at least their parts involved in the testable model. The obtained models are used to detect compositional problems by RA, which then can be either confirmed or refuted by testing components in isolation. In the latter case, the inferred models could be refined until the obtained models constitute a well-formed system or a compositional problem is confirmed in the SUT.

Note that if several actions can concurrently be executed in the SUT, the system could produce several output interleavings in response to a given external input sequence, and thus become output-nondeterministic. Our main goal is precisely to be able to detect such potential interleavings, even though when a system is tested in a practical configuration, only one interleaving can be observed. Therefore, we assume that during the inference the system behaves as an output-deterministic IOTS and use the inference method of Section 3. Actually, if it does not, this simply implies that a race could already be detected during the inference phase.

Among n components, $C_1, \ldots, C_n$, we assume that models for the first p components are unknown, while the remaining components are supplied with the precise, i.e., adequate, complete FSM/IOTS models $M_{p+1}, \ldots, M_n$.

An initial Z-quotient of the testable model of the SUT's slow asynchronous product can be inferred using the method in Section 3. The resulting Z-quotient M can be used to determine missing models of components by projecting the quotient onto the action sets of each component with a missing model. In particular, $M_i = (M_{\downarrow(I_i' \cup O_i)})'$, where $i = 1, \ldots, p$. Notice that if some

[3] This is, of course, not a sufficient, but necessary condition for divergence.

[4] One can use "a single message in transit" as a sufficient condition for their absence [PY98].

projection is a trivial model, while we know that the system has no unused connection (hence no unused components), then this indicates that the obtained Z-quotient misses interactions of that component and the set Z has to be reconsidered. The obtained models can be minimized to reduce the number of states and thus the complexity of the subsequent RA.

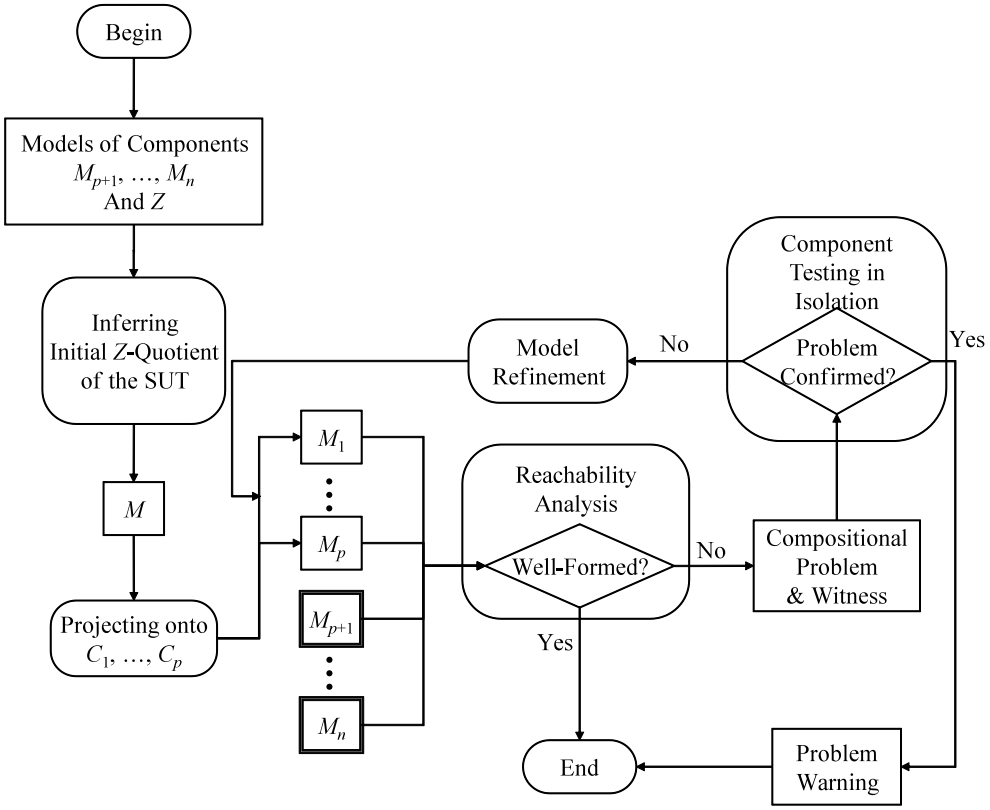


Figure 7: The Verification Approach

The resulting models together with the initially given models are an input to a RA procedure. The necessity of RA comes from the fact that even if the components themselves compose a well-formed system the inferred models do not necessarily do so. Each obtained model is only an approximation of the actual behaviour of a real component and may possess a behaviour absent in the component, so the inferred models may exhibit compositional problems. To detect divergence the RA procedure should also be given the maximal allowable size of each queue.

The models exhibit a compositional problem in two cases: either the problem exists in the SUT or the inferred models are not adequate models of the behaviour of the components exposed during the exploration step and need to be further refined. To confirm that a compositional problem detected in RA exists in the real system, one can check whether each projection of a witness trace, which was not observed in the exploration step, belongs to the set of traces of each involved component with an inferred model, by executing the projected trace against the components in isolation. Testing terminates as soon as the first component involved in the execution for which a discrepancy between the inferred model and actual behaviour exists is discovered. In this case, the compositional problem is refuted, and the newly obtained trace is used to refine the model of that component. RA is repeated again using the refined models. The iterative process terminates when either a compositional problem is confirmed to exist in the SUT or a well-formed system of models is obtained. Notice the SUT with compositional livelock should exhibit a cyclic behaviour in all the involved components when they are tested using the projected witness traces. Since the initially given models M_{p+1}, …, M_n are assumed to be precise, these components are omitted from testing.

While the main steps of this procedure are intuitive and clear from the previous discussions, the model refinement needs some explanation. Assume that for some witness trace, the component C_i in response to the input projection of the witness trace produces a trace α which is not in the inferred IOTS M_i. To refine the IOTS M_i one needs to replace outputs produced by M_i in response to the input projection by newly observed ones. To achieve this, we use the fact that any incorrect output in the model M_i can only be a result of state merging in the Z-quotient IOTS M which is obtained from the global observation tree U or in its projection to the action sets of C_i. Indeed, the global observation tree contains only traces actually observed from the SUT and

additional traces occur when states are merged. To obtain a refined model for the component C_i, it is thus sufficient to determine the local observation tree U_i by projecting the global observation tree U, add the newly observed trace to it, $U_i := U_i \cup \{\alpha\}$, and minimize the states of the obtained tree FSM (using the trace inclusion relation). The problem of constructing a minimal machine consistent with a given finite regular language is well studied from different perspectives, such as automata identification [BF72], FSM minimization [KVB97] and test coverage analysis [PBY96]. Its complexity is known [KVB97]; notice, however, that in our case, there is no need to aim at constructing a machine with the minimal number of states; in fact, any machine without deadlock states could be used for RA. Removing deadlock states reduces the number of iterations involving detecting unspecified receptions (usually caused by such states), testing in isolation, and model refinement.

Dealing with unspecified receptions, the witness trace βa could be extended to end in a stable state. The projection of such a trace $\beta a \gamma$ could provide more inputs after a to be tested on the component C_i thus reducing the number of steps as will be shown on the example.

Dealing with livelock, we need to test whether a projected cycle corresponds to a real cycle in each involved component. To that end, we set a repetition bound r to generate tests corresponding to the projection of $\alpha\beta^r$. If testing provides a trace that does not cycle r times for one of the components, then the model of that component is updated with the new trace. Otherwise, we assume that the livelock exists in the SUT. A similar argument is used for divergence since we only get a necessary condition for divergence from the witness.

4.3. Example

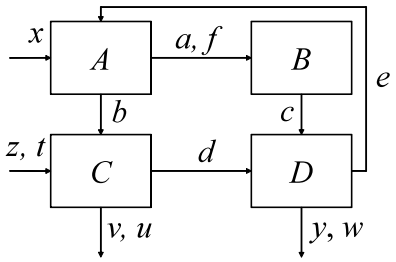


Figure 8: A Modular System

We illustrate the proposed approach using the example shown in Figure 8. It consists of four components. External inputs and outputs are labelled with letters from *t* to *z*, while internal ones range from *a* to *f*. Among the inputs, *t* is not controllable, while among the outputs, *u* is not observable. The models of components *A*, *B*, *C*, and *D* are depicted in Figure 9, Figure 10, Figure 11, and Figure 12, respectively. In the figures depicting IOTS in this section, double circles denote stable states.

In this example, we suppose that the models of the components *A* and *B* are known, while the components *C* and *D* are black boxes.

In the example, we take Z as {*xx*}. Note that *x* and *z* are controllable inputs, and from the knowledge of *A*, it makes sense to consider that sending two consecutive inputs *x* will trigger sufficiently diverse behaviours in the rest of the system (after the first *x*, *A* cycles over *xx*).

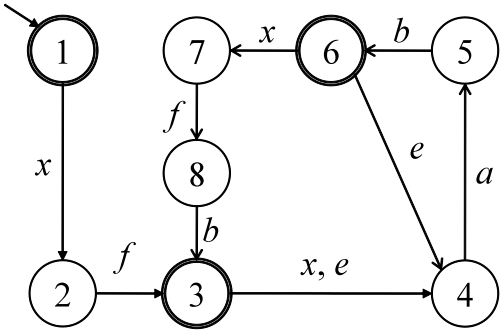


Figure 9: Component *A*

Initial Z-Quotient Inference

We use the inference algorithm in Section 3.2. Starting from the initial state ε, the global observation tree U obtained after applying the external input strings in the set $I \cup Z = \{z, x, xx\}$ is depicted in Figure 13. The set of states Q of the initial Z-quotient is $\{\varepsilon\}$.

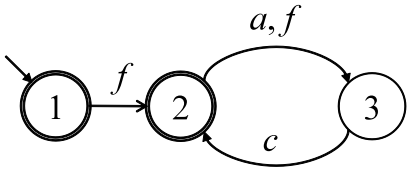


Figure 10: Component *B*

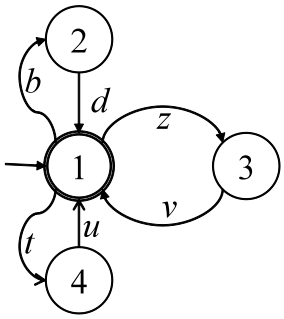


Figure 11: Component *C*

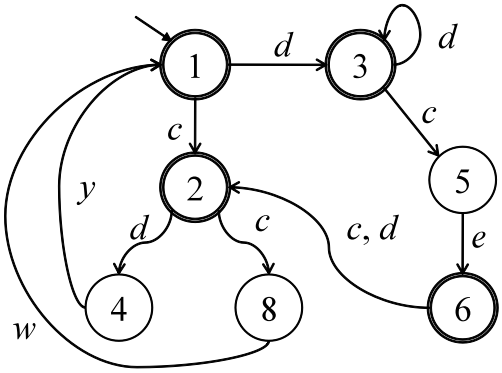


Figure 12: Component *D*

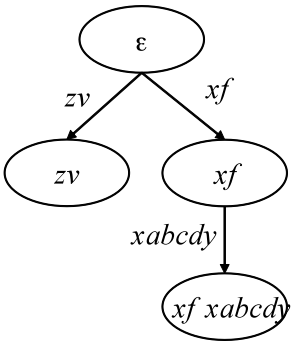


Figure 13: Behaviour Exploration from ε

We assume here that the concurrent events a and b result in the interleaving cd, which the system under test always produces in our testing configuration.

After that, we explore the behaviour of the system starting from state zv. The obtained U is depicted in Figure 14. The state zv is Z-equivalent to state ε. Thus, we label state zv with ε, and Q is still $\{\varepsilon\}$.

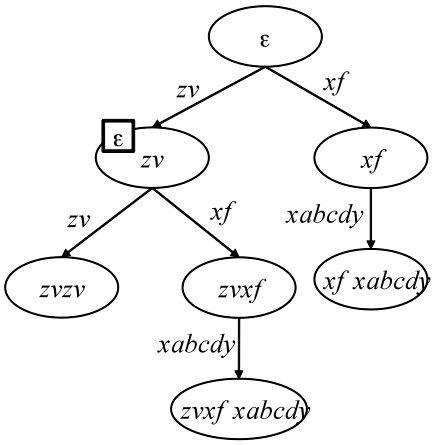


Figure 14: Behaviour Exploration from *zv*

Then, we explore the behaviour of the system starting from state *xf*. The obtained *U* is depicted in Figure 15. The state *xf* is not Z-equivalent to any visited stable state. Thus, $Q = \{\varepsilon, xf\}$.

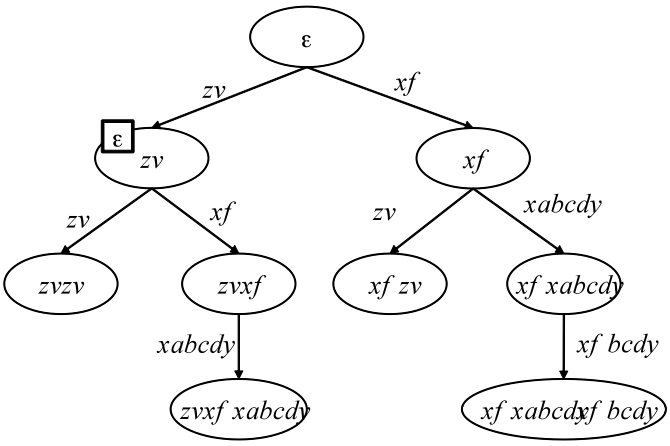


Figure 15: Behaviour Exploration from *xf*

Next, we explore the behaviour of the system starting from state *xfzv*. The state *xfzv* is Z-equivalent to state *xf*. Thus, we label state *xfzv* with *xf*, and *Q* is still $\{\varepsilon, xf\}$.

After, we explore the behaviour of the system starting from state *xfxabcdy*. The obtained *U* is depicted in Figure 16. The state *xfxabcdy* is not Z-equivalent to any visited stable state. Thus, $Q = \{\varepsilon, xf, xfxabcdy\}$.

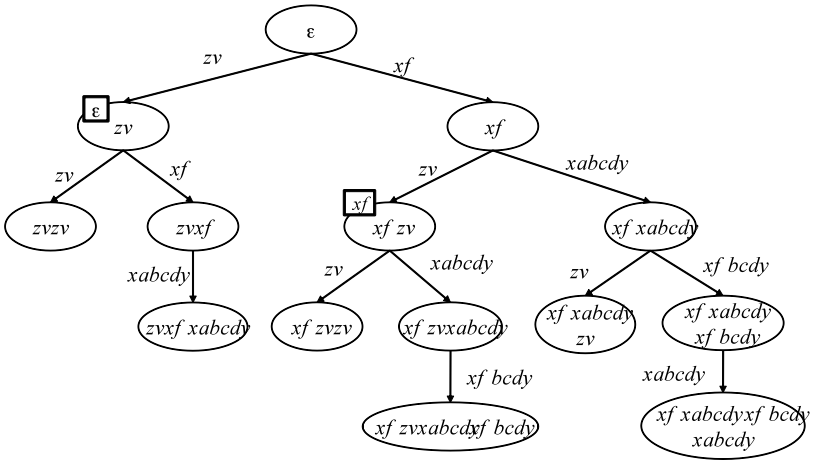


Figure 16: Behaviour Exploration from *xfxabcdy*

As the next step, we explore the behaviour of the system starting from state *xfxabcdyzv*. The state *xfxabcdyzv* is Z-equivalent to state *xfxabcdy*. Thus, we label state *xfxabcdyzv* with *xfxabcdy*, and *Q* is still $\{\varepsilon, xf, xfxabcdy\}$.

Then, we explore the behaviour of the system starting from state *xfxabcdyxfbcdy*. The obtained *U* is depicted in Figure 17. At this point, we identify that the state *xfxabcdyxfbcdy* is Z-equivalent to state *xf*. We label the state with *xf*. The observation tree construction terminates.

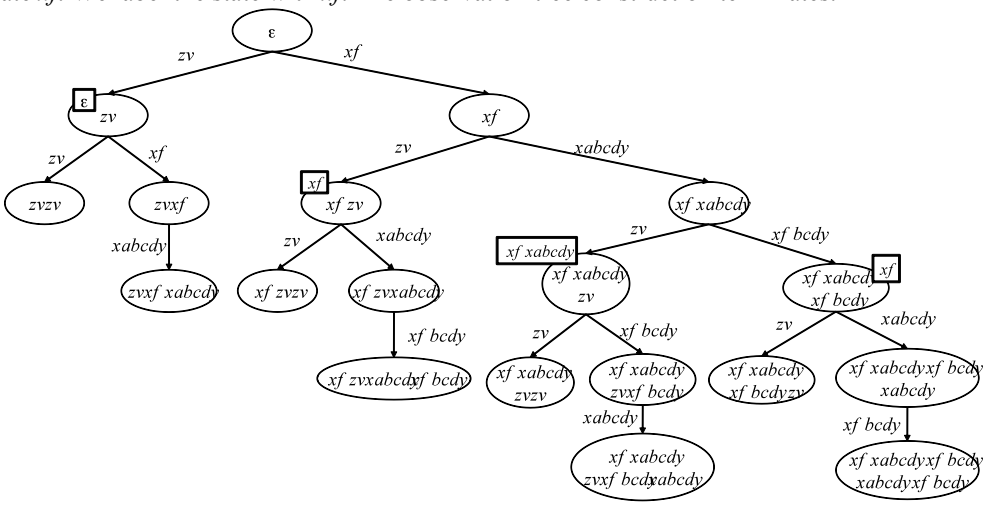


Figure 17: Behaviour Exploration from *xfxabcdyxfbcdy*

In the transition determination phase, we obtain the FSM *M*, which is an initial Z-quotient of the testable model of the given system depicted in Figure 18.

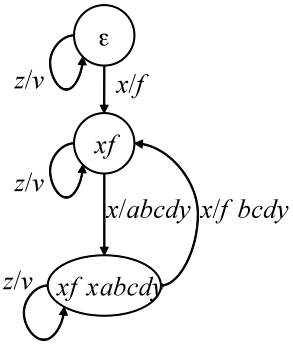


Figure 18: The Initial Z-Quotient *M* as FSM

Next, we determine models of components *C* and *D* by projection. M_3 and M_4 are shown in Figure 19, and Figure 20, respectively.

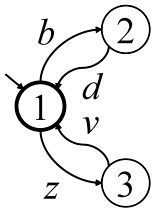


Figure 19: M_3

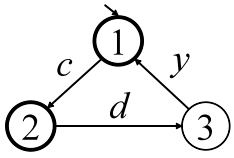


Figure 20: M_4

At this point, we may note that M_3 is a testable model of component *C* and the model M_4 of component *D* is only a part of the actual model of that component. In particular, the absence of the interleaving *dc* made it impossible to observe the *e* event and the corresponding branch in component *A*. RA will spot this missing interleaving.

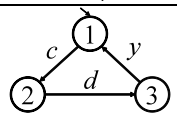
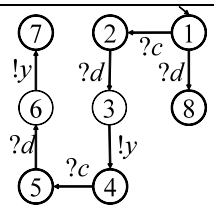
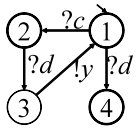
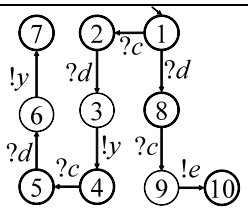
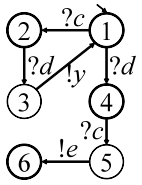
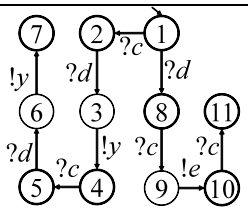
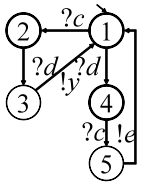
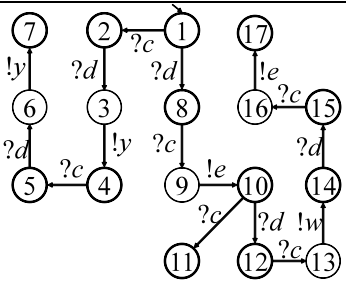
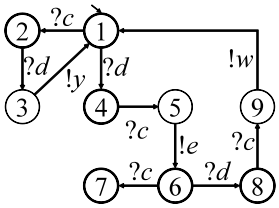
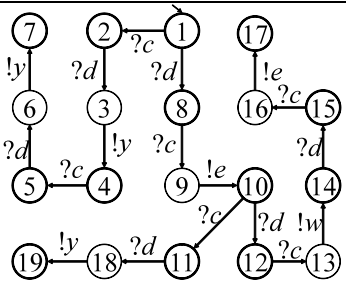
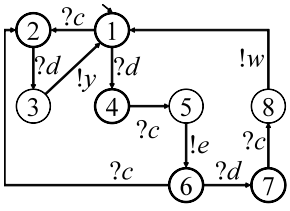
Iterative Refinement

Now we perform reachability analysis of the system composed of *A*, *B*, M_3 and M_4 to check whether it is well-formed. This starts the iterative process of reachability analysis, testing in isolation, and component model refinement as described in Section 4.2. In these iterations, only M_4 is updated. The iterations are shown in Table 1. The table contains local observation trees and models of the components *D* as well as a witness trace for each version of the model. Note that the reachability analysis is performed using the model checker SPIN. The sequence chart generated by SPIN for the first step is shown in Figure 21, while detection of livelock is illustrated in Figure 22.

The RA of the obtained models results now in the witness traces for races: *xx'ff'xx'aa'cc'bb'dd'y* and *xx'ff'xx'aa'bb'dd'cc'ee'aa'bb'dd'cc'w*. In fact, the external input sequence *xx* yields several traces in the product which differ in their output projections, *y* and *w*. We project the two traces to

each component, and find that all the obtained traces are already in the corresponding local observation trees. The compositional problem is confirmed. With this report the procedure terminates. To further illustrate the proposed approach, we next assume that the detected compositional problem is resolved by the system designer and our procedure is used to check the updated design.

Table 1: Iterative Model Refinement

	U_4	M_4	Witness traces
			*xx'ff'xx'aa'bb'd* (unspecified reception)
1			*xx'ff'xx'aa'bb'dd'c* (unspecified reception)
2			*xx'ff'xx'aa'bb'dd'cc'ee'aa'c* (unspecified reception)
3			Prefix: *xx'ff'xx'* Cycle: *aa'bb'dd'cc'ee'* (livelock)
4			*xx'ff'xx'aa'bb'dd'cc'ee'aa'cc 'bb'd* (unspecified reception)
5			*xx'ff'xx'aa'cc'bb'dd'y* and *xx'ff'xx'aa'bb'dd'cc'ee'aa'bb 'dd'cc'w* (race)

Model Update

When the compositional problem is reported, the system designer re-checks the design, and confirms that the race exists in the real system. Then the system designer changes the design and correspondingly the implementation of component *D* as depicted in Figure 23.

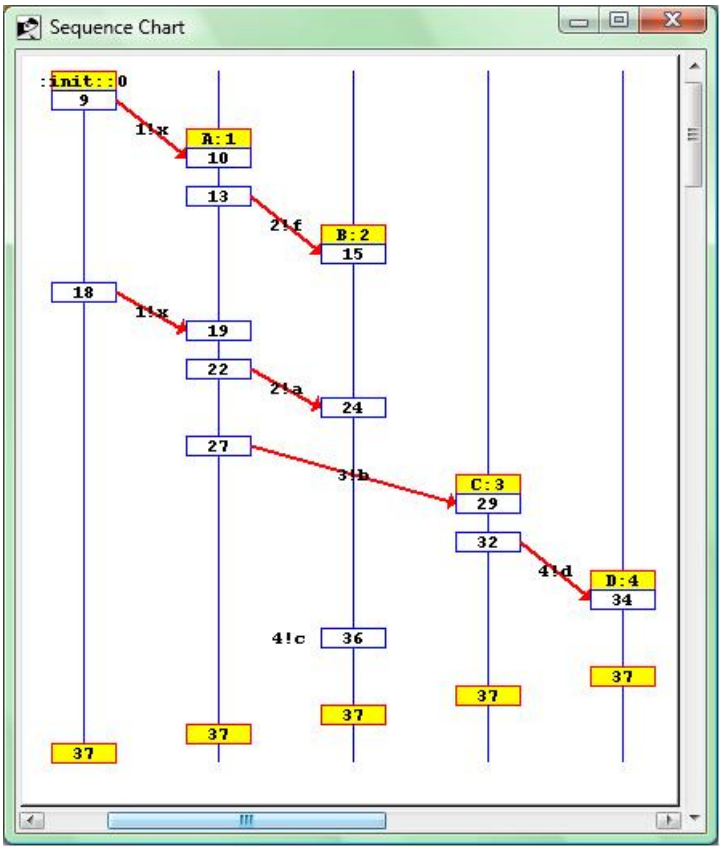


Figure 21: Unspecified Reception

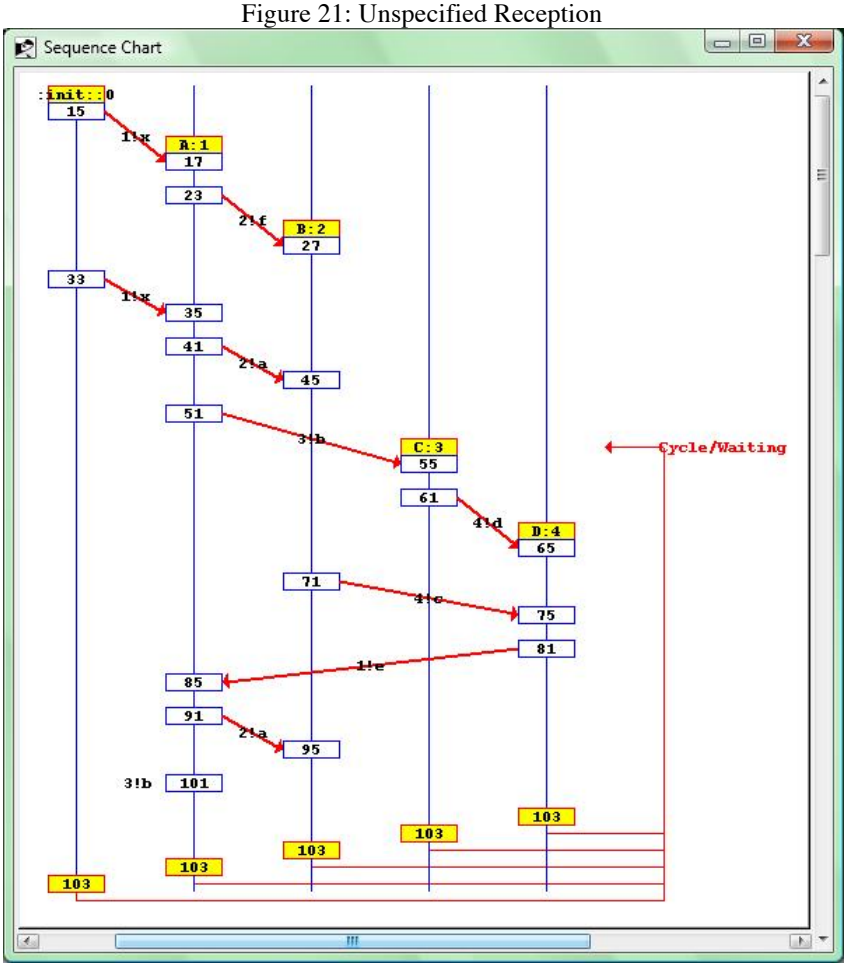


Figure 22: Livelock

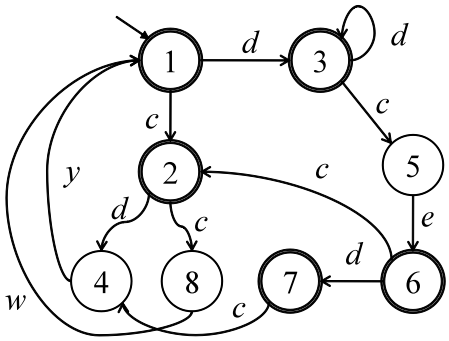
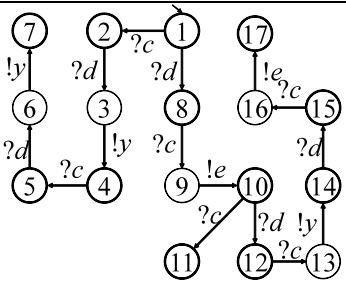
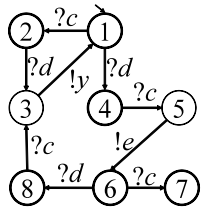
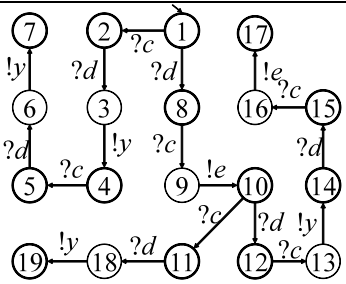
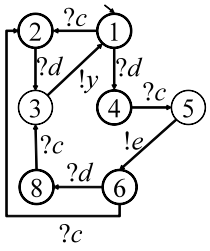


Figure 23: Updated model of component *D*

Now, we redo the verification procedure. Until iteration 3 in **Table 1**, the observations and the models obtained are the same as before. The subsequent iterations are depicted in Table 2.

Table 2: Iterations after model update

	U_4	M_4	Witness traces
4			*xx'ff'xx'aa'bb'dd'cc'ee'aa'cc'bb'd* (unspecified reception)
5			no compositional problem is detected

In iteration 5, no compositional problem is identified, and thus the procedure terminates.

4.4. Relaxing the Assumptions about the Modular SUT

Presenting the approach to verification of a modular system with unknown components, we made a number of simplifying assumptions about the SUT in hand; some of them are not essential and can be relaxed, along with those listed in Section 3.3.

The verification approach relies on the accuracy of the initially supplied models and avoids determining their local observation trees from the global observation tree and inferring their models. If, however, the accuracy of supplied models cannot be taken for granted then the observed traces of a concerned component can be used to check whether they actually belong to the component's model. If it is not the case then either the verification process terminates or continues provided that the model is refined or completely discarded (the component needs then complete model inference). The initially supplied models are assumed to be fully specified, defining the behaviour of the components for each input action. Nevertheless, it is possible to allow these models to be partial; in this case, RA may discover an unspecified reception for a component with a partial model. The model could then be completed, using observations obtained by testing the component in isolation, should the verification continue after discovering incompleteness of the model.

The assumptions used by the approach to infer an initial **Z**-quotient of the system in a slow environment are that the SUT architecture has no unused connection and the inference parameter is set such that each connection is at least once used when the **Z**-quotient is inferred. However, the absence of unused connections may not always be taken for granted; a system in hand may have "dead" connections and even components. To discover them by testing requires repeatedly

extending the inference parameter; though, there seems to be no general termination rule. Conversely, without some knowledge about the application domain and specifics of a given SUT, it is difficult to choose the right inference parameter even if there is no dead connection. In the situations, when the value of this parameter cannot guarantee the use of each and every connection, the verification procedure may still continue with a given inference parameter even if no activity is observed on some connections. The ratio of the number of connections used in the product to their total number in the given architecture may be used to characterize the degree to which the SUT architecture is discovered in testing and thus the quality of the inferred models in a well-formed system. In addition, if the global observation tree contains no action executed by some component, with or without the initial model, then the component is declared isolated in the given architecture for a fixed inference parameter.

Finally, the approach assumes that the SUT can be disassembled to perform components testing in isolation needed to confirm or refute a compositional problem found in the inferred models. One can argue that it may not always be possible and the SUT should only be tested as an integrated system. In this scenario, one possibility is to check whether the detected problem persists when the inference parameter is extended and more precise models are inferred. Additional knowledge about the SUT or heuristics would then be needed to terminate the verification process.

5. Related Work

State model learning is widely addressed, specifically in the grammatical inference works, e.g., [BDG97], [KV94] [Hig10]. Angluin presents a seminal algorithm [Ang87] to infer regular language as deterministic finite automaton, based on the Minimally Adequate Teacher (MAT) paradigm. This paradigm uses an equivalence oracle which decides on the equivalence between the inferred machine and the ideal minimal recognizer. In our work, the state Z-equivalence relation can be viewed as an approximated equivalence oracle: checking Z-equivalence of all states (including the initial one) is a weakened relation from checking full equivalence. A MAT provides a counterexample in case of non-equivalence, which in our case corresponds to test sequences in u.Z for non pruned traces u.

Learning and testing through model checking approach is used to infer a grey box system [EGP06]. However, the upper bound on the number of states in the system is required to test conformance of the conjectured model to the actual system. On the other hand, the notion of Z-quotient provides a means for inferring a variable size approximation without upper bound on the number of states. Additionally, whereas the length of conformance testing sequences is exponential in the upper bound, Z-equivalence provides a flexible way to avoid exponential blow-up.

The observation tree FSM used in this paper is, in fact, an input-output version of a prefix tree machine [CW98] used in various techniques for learning an automaton from positive examples. However, we do not require samples of the behaviour given for inference, the samples (traces) are obtained by testing whose termination is determined based on a predefined inference parameter, a set of input sequences Z. At the same time, as in a common paradigm in learning from positive examples, the local observation tree FSM is also iteratively minimized. The rule used for machine minimization in the proposed approach is language containment (input-output trace inclusion). Similar to the parameter k used for k-tail in [BF72], the parameter Z allows one to control the precision and complexity of the resulting machine. However, differently from that work, the latter is a deterministic FSM.

There is much less work published on the direct inference of modular systems, although [EGP06] and [PGB08] for instance consider inference of components in modular systems. The work [HBP06] relies on observed traces to construct automata models of communicating components which are then model-checked using user defined properties. An object flattening technique [MP05] is used to collect system behaviour and then invariants are calculated on the behaviours to check against the new version of the system. This work is more related to regression testing. Moreover, the system behaviour is observed while the system is running as in [HBP06]. In this paper, we rely on testing a modular system by stimulating it through external inputs and then use the observations to obtain tuneable approximated models. The verification of a modular system treated as a black box on the architectural level is also addressed in [BMP06]. Similar to the previous approaches, the system is monitored at runtime by instrumenting the middleware, no testing strategy is used. On the contrary, we infer models by testing and use them to check the system for compositional problems.

Our past work [LGS06] and [SLG07] in this domain also concerned the inference of components as finite state machines through testing. In fact, given an input alphabet I of a

component, we were implicitly inferring an I-quotient of each component in isolation without defining the Z-quotient. Moreover, the previous approaches focused on learning components separately, one component at a time; whereas in this paper, we proposed to test the integrated system avoiding thus unnecessary testing efforts to learn models which may not be related to the composition. An earlier version of this paper appeared in [GLP08], where we elaborated several initial ideas of the proposed modular system verification approach. That work introduced the notion of a k-quotient, which is a special case of the proposed Z-quotient, namely, I^k-quotient, and the inference algorithm of [GLP08] is now further refined along with underlying assumptions about a given system and information available for inference.

6. Conclusion

In this paper, we offered a solution to the problem of modular system verification by blending together techniques of inference, testing, and reachability analysis.

We first defined a controlled approximation of finite state systems with the notion of a Z-quotient. More precisely, we defined the notion of initial Z-quotient, which is well-suited to testing, as it covers parts of the system that can be reached by testing while still being based on a Z-equivalence relation, and associated Z-distinguishability w.r.t. a set of input sequences Z. The precision of this approximation can be varied with the parameter Z. We then proposed an approach to infer models of a modular system. We first infer an initial Z-quotient approximation of (a testable model of) the modular system, from which we get initial models of components by projecting the quotient. Then we iterate between reachability analysis (RA) of intermediate models and pinpointed testing of components in isolation to check potential problems and refine models. RA of the models is used to identify composition problems, such as unspecified receptions, livelocks, divergences, and races. A witness (diagnostic) trace is then used to test concerned components in isolation either to confirm that a problem exists in the real system or to obtain new observations. The models are then refined using new observations until the obtained models are well-formed.

The approach assumes that the modular system can be modelled as a deterministic finite state system. Actually, since it is assumed to be built from deterministic components, non-deterministic behaviours would be the result of communication races which should be avoided and would be detected either in testing or in reachability analysis. A nice point of the approach is that it can reveal non-determinism that would not be detected by testing in a fixed environment. One key advantage of the approach is that it adapts to the verification goal and efforts, with the help of the inference parameter Z, which allows one to find a compromise between complexity of testing of the integrated system and precision of the resulting models. Moreover, the use of large sets of input sequences is completely avoided in testing a component in isolation, since only single diagnostic test is executed in each iteration. Another advantage of the approach is that inferred models capture the functionalities of components used in the given system; unused behaviours of components are not modelled.

As a future work, it would be interesting to investigate whether instead of testing a number of components in isolation based on a witness trace one would test just a subsystem consisting of these components to reduce the number of iterations needed to infer well-formed models. There are also a number of options for treating witness traces to update the global observation tree once the individual models are refined. This may help converge faster and shorten the RA process. It is known that reachability analysis can provide more than one witness traces, evidencing multiple problems in one step. The treatment of multiple traces at a time could be another improvement in the approach. Finally, the inference method is based on a universal environment which can provide all controllable inputs to a system in any stable state. Just as we use an inference parameter with a restricted set of input sequences for state distinguishability, it might be worth considering constrained environments that could provide only specific transfer sequences in the exploration step.

Acknowledgement

This work was partially supported by the EU FP7 Project no. 257876, “SPaCIoS: Secure Provision and Consumption in the Internet of Services” (www.spacios.eu).

References

[Ang87] D. Angluin, Learning Regular Sets from Queries and Counterexamples, Information and Computation, 2:87–106, 1987.

[BDG97] J. L. Balcazar, J. Diaz, and R. Gavalda, Algorithms for Learning Finite Automata from Queries: A Unified View. In Advances in Algorithms, Languages, and Complexity, Kluwer, pages 53–72, 1997.

[BF72] A. Biermann and J. Feldman, On the Synthesis of Finite State Machines from Samples of their Behavior. IEEE Transactions on Computers, 21(6):592–597, 1972.

[BMP06] A. Bertolino, H. Muccini, and A. Polini, Architectural Verification of Black-box Component-based Systems. In Proceedings of the 3rd International Workshop on Rapid Integration of Software Engineering Techniques (RISE), LNCS 4401, pages 98–113, 2006.

[CGP99] E.M. Clarke, O. Grumberg and D. Peled, Model Checking. Cambridge, Mass. MIT Press, 1999.

[CW98] J. E. Cook and A. L. Wolf, Discovering Models of Software Processes from Event-Based Data. ACM Transactions on Software Engineering and Methodology, 7(3):215–249, 1998.

[EGI10] K. El Fakih, R. Groz, M.N Irfan, M. Shahbaz, Learning Finite State Models of Observable Nondeterministic Systems in a Testing Context. In 22th IFIP International Conference on Testing Software and Systems (ICTSS), short papers, pages 97-102, 2010.

[EGP06] E. Elkind, B. Genest, D. Peled, and H. Qu, Grey-box Checking. In Proceedings of 26th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), pages 420–435, 2006.

[GLP08] R. Groz, K. Li, A. Petrenko, M. Shahbaz, Modular System Verification by Inference, Testing and Reachability Analysis, In Proceedings of 20th IFIP International Conference on Testing of Software and Communication Systems (TestCom), pp. 216-233, 2008.

[God97] P. Godefroid, Model Checking for Programming Languages Using VeriSoft, In Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), ACM Press New York, pages 174–186, 1997.

[HBP06] H. H. Hallal, S. Boroday, A. Petrenko, and A. Ulrich, A Formal Approach to Testing Properties in Causally Consistent Distributed Traces. Formal Aspects of Computing, 18 (1): 63-83, 2006.

[Hig10] C. de la Higuera, Grammatical Inference: Learning Automata and Grammars, Cambridge University Press, 2010.

[HP09] J. Huo, A. Petrenko, Transition Covering Tests for Systems with Queues, Software Testing, Verification and Reliability, 19 (1): 55-83, 2009.

[KV94] M. J. Kearns and U. V. Vazirani, An Introduction to Computational Learning Theory. MIT Press, Cambridge, MA, USA, 1994.

[KVB97] T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli, Synthesis of FSMs: Functional Optimization, Boston, USA, Kluwer Academic Publishers, 1997.

[LBP94] G. Luo, G. v. Bochmann, A. Petrenko, Test Selection Based on Communicating Nondeterministic Finite State Machines Using a Generalized Wp-Method, IEEE Transactions on Software Engineering, 20(2): 149-162, 1994.

[LGS06] K. Li, R. Groz, and M. Shahbaz, Integration Testing of Components Guided by Incremental State Machine Learning. In Proceedings of Testing: Academia and Industry Conference - Practice And Research Techniques (TAIC PART), pages 231–247, 2006.

[LT89] N. Lynch, M. Tuttle, An Introduction to Input/output Automata, CWI-Quarterly, 2(3):219–246, 1989.

[MP05] L. Mariani and M. Pezzè, Behavior Capture and Test: Automated Analysis of Component Integration. In Proceedings of 10th International Conference on Engineering of Complex Computer Systems (ICECCS), pages 292–301, 2005.

[Ner58] A. Nerode, Linear Automaton Transformations. In Proceedings of the American Mathematical Society, 9(4):541-544, 1958.

[PBY96] A. Petrenko, G. v. Bochmann, and M. Yao, On Fault Coverage of Tests for Finite State Specifications, Computer Networks and ISDN Systems, 29(1): 81-106, 1996.

[PGB08] C. S. Pasareanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, H. Barringer, Learning to Divide and Conquer: Applying the L* Algorithm to Automate Assume-Guarantee Reasoning, Formal Methods in System Design 32(3): 175-205 (2008).

[PVY99] D. Peled, M. Y. Vardi, M. Yannakakis, Black Box Checking, In Proceedings of 12th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), pages 225-240, 1999.

[PY98] A. Petrenko and N. Yevtushenko, Solving Asynchronous Equations. In Proceedings of 11th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), pages 231–247, 1998.

[SLG07] M. Shahbaz, K. Li, and R. Groz, Learning and Integration of Parameterized Components Through Testing. In Proceedings of 19th IFIP International Conference on Testing of Software and Communication Systems (TestCom), pages 319–334, 2007.

[TB73] B. A. Trakhtenbrot, and Y. M. Barzdin, Finite Automata, Behaviour and Synthesis. North-Holland, 1973.

[Tre96] J. Tretmans, Test Generation with Inputs, Outputs and Repetitive Quiescence. Software - Concepts and Tools 17(3): 103-120, 1996.

[Vas73] M. P. Vasilevskii. Failure Diagnosis of Automata. Cybernetics and Systems Analysis, 9:653-665, 1973.

Appendix A: Proof of Theorem 2

The first part of Theorem 2 is "if Z is a characterization set of A, then FSMs A and K are equivalent."

The FSM K is a Z-quotient of A, there exists an injection f from Q to S. To demonstrate that K is equivalent to A, we first prove that f is a bijection, i.e., $f(Q) = S$.

Assume that $f(Q) \neq S$. $f(Q) \neq \emptyset$, as $f(q_0) = s_0$. Then since A is initially connected, there exists a transition from $s \in f(Q)$ to $t \notin f(Q)$, i.e., $(t, \beta) \in h(s, a)$ for some input a and output sequence β. Since K is Z-quotient and it is a complete FSM, there should be a transition such that $(p, \beta) \in k(f^{-1}(s), a)$. Moreover, t and $f(p)$ are Z-equivalent. Since Z is a characterization set of the minimal FSM A, $t = f(p)$, i.e., $t \in f(Q)$. A contradiction. Thus, $f(Q) = S$.

According to the last condition of the definition of a quotient, for any $p, q \in Q$, $a \in I$, and $\beta \in O^*$, $(p, \beta) \in k(q, a)$ iff there exists $s \in S$, such that $(s, \beta) \in h(f(q), a)$ and s and $f(p)$ are Z-equivalent. Since Z is a characterization set of A, $s = f(p)$. Thus, we have $(f(p), \beta) \in h(f(q), a)$. In other words, the transitions in K are also preserved by the bijection f. Thus, f is an isomorphism and the FSMs K and A are equivalent.

The second part of Theorem 2 is "if A has distinguishable but Z-equivalent states, then A and K are distinguishable."

Assume that the two minimal FSMs A and K are equivalent. Thus we have $|Q| = |S|$. Thus, f is an isomorphism. Let the two distinguishable but Z-equivalent states be s and t, then there exist $p, q \in Q$, $p \neq q$, such that $f(p) = s$, and $f(q) = t$. By the definition of quotient, $f(p)$ and $f(q)$ are Z-distinguishable, this implies that states s and t have to be Z-distinguishable. A contradiction.

This concludes the proof of Theorem 2.

Appendix B: Proof of Theorem 3

We introduce an additional notation to simplify the presentation. Given a deterministic FSM $A = (S, s_0, I, O, E, h)$, state s and trace $u \in Tr_A(s)$, we let s-after-u to denote the state that is reached by A executing trace u from the state s.

Let the unknown deterministic FSM be $A = (S, s_0, I, O, E, h)$, and an FSM derived by the algorithm is $K = (Q, q_0, I, O, E, k)$. Without loss of generality, we assume that the states of K are in fact traces constituting states in the observation tree $U = (U, \varepsilon, I, O, E, h_U)$ constructed by the algorithm.

We need to prove that K is an initial Z-quotient of A, i.e., we need to prove that there exists an injection f from Q to S satisfying all the conditions in the definition.

Consider a function f from Q to S such that for $q \in Q$, $f(q) = s_0$-after-q.

(1) According to Step 1 of the algorithm, $f(q_0) = s_0$-after-$\varepsilon = s_0$.

(2) Growing an observation tree from an arbitrary state u, we explore the behaviour of the FSM A by applying input sequences from the set Z and observing the set of traces $\{v \in Tr_A(s_0\text{-after-}u) \mid v_{\downarrow I} \in Z\}$; it is equal to $\{v \in Tr_U(u) \mid v_{\downarrow I} \in Z\}$ in U. For two distinct states $q_1, q_2 \in Q$, we know that q_1 and q_2 are Z-distinguishable in U. In other words, $\{v \in Tr_U(q_1) \mid v_{\downarrow I} \in Z\} \neq \{v \in Tr_U(q_2) \mid v_{\downarrow I} \in Z\}$. Thus, $\{v \in Tr_A(s_0\text{-after-}q_1) \mid v_{\downarrow I} \in Z\} \neq \{v \in Tr_A(s_0\text{-after-}q_2) \mid v_{\downarrow I} \in Z\}$. This means s_0-after-q_1 and s_0-after-q_2 are Z-distinguishable, i.e., $f(q_1)$ and $f(q_2)$ are Z-distinguishable. But then $f(q_1) \neq f(q_2)$, thus, f is an injection.

(3) For $q = a_1\beta_1 a_2\beta_2 \ldots a_n\beta_n \in Q$, this is a state of a quotient, hence according to the algorithm, in U there exist $q_1 = a_1\beta_1$, $q_2 = a_1\beta_1 a_2\beta_2$, …, $q_{n-1} = a_1\beta_1 a_2\beta_2 \ldots a_{n-1}\beta_{n-1}$ and a path $(\varepsilon, a_1\beta_1, q_1)$ $(q_1, a_2\beta_2, q_2) \ldots (q_{n-1}, a_n\beta_n, q_n)$ from ε to $q_n = q$, and $q_i \in Q$ $(1 \leq i \leq n)$. Thus, in A there exists a path $(s_0,$

$a_1\beta_1$, s_0-after-q_1) (s_0-after-q_1, $a_2\beta_2$, s_0-after-q_2) … (s_0-after-q_{n-1}, $a_n\beta_n$, s_0-after-q), moreover, s_0-after-$q_i = f(q_i)$, for all $1 \le i \le n$, and s_0-after-$q = f(q)$.

(4) for any $q \in Q$ and $a \in I$, $\beta \in O^*$, $(p, \beta) \in k(q, a)$ iff there exists $s \in S$, such that $(s, \beta) \in h(f(q), a)$ and s and $f(p)$ are Z-equivalent. There are two directions to be considered. In one direction, for $q \in Q$, $a \in I$, suppose that there exist a transition $(f(q), a\beta, s)$ in A and a corresponding transition $(q, a\beta, qa\beta)$ in U. Executing the algorithm, the state $qa\beta$ is either labelled or not. If $qa\beta$ is not labelled, then $(q, a\beta, qa\beta)$ is also a transition in K, and $f(qa\beta) = s_0$-after-$qa\beta = f(q)$-after-$a\beta = s$. If the state $qa\beta$ is labelled, then let $label(qa\beta) = p$, this means that in U $qa\beta$ is Z-equivalent to p, while in A s_0-after-$qa\beta$ is Z-equivalent to s_0-after-p, and in K we have $(p, \beta) \in k(q, a)$. Then s is Z-equivalent to $f(p)$. In the other direction, for $q \in Q$, $a \in I$, suppose there exist a transition $(q, a\beta, p)$ in K, and a corresponding transition $(q, a\beta, qa\beta)$ in U. If $qa\beta = p$, there exists a transition (s_0-after-q, $a\beta$, s_0-after-p) = ($f(q)$, $a\beta$, $f(p)$) in A. Otherwise, in U $label(qa\beta) = p$, which means that in U $qa\beta$ is Z-equivalent to p, and thus in A, s_0-after-$qa\beta$ is Z-equivalent to s_0-after-p. Thus, in A there exists a transition (s_0-after-q, $a\beta$, s) = ($f(q)$, $a\beta$, s) and s is Z-equivalent to s_0-after-$p = f(p)$.

The injection f possesses all the conditions of the definition of quotient; thus, K is an initial Z-quotient of A.


L I G