Contributions to model based test generation and monitoring strategies and their networking applications

Natalia Kushik

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAXXXX

Thèse de Habilitation à Diriger des Recherches

Contributions to model based test generation and monitoring strategies and their networking applications

Habilitation à Diriger des Recherches de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Informatique, données, intelligence artificielle

Thèse présentée et soutenue à Palaiseau, le 15/09/2022, par

NATALIA KUSHIK

Composition du Jury :

Rob Hierons Professor, University of Sheffield	Rapporteur
Pascale Le Gall Professeur, CentraleSupélec	Rapporteur
Franz Wotawa Professor, Graz University of Technology	Rapporteur
Burkhart Wolff Professeur, Université Paris-Saclay	Président
Roland Groz Professeur, Grenoble INP UGA	Examinateur
Tiziano Villa Professor, University of Verona	Examinateur
Djamal Zeghlache Professeur, Télécom SudParis	Examinateur

Abstract

The work presents a number of contributions in the area of model based testing (MBT), on the one hand, and some MBT applications, on the other hand. MBT generally relies on a formal specification of the system under test that later on allows to assure that its implementation conforms to the specification (or not). In our case, we mostly focus on the analysis of reactive systems, i.e., the systems working in request-response mode and moreover the behavior of these systems is sequential. Therefore, we mostly study transition or state models that change their states when an input is applied and/or an output is produced. Application areas of this work mostly cover networks and in particular dynamic and programmable networks whose components need to be thoroughly tested and verified. At the same time, as the behavior of reactive systems is not only sequential but also (highly) nondeterministic, we pay additional attention to the methods and techniques developed for testing (with the guaranteed fault coverage) and monitoring against nondeterministic and probably non-observable specifications. We discuss the problems that appear in the latter case and draw a particular attention to the state identification issues in nondeterministic specifications. We present the current state of the art in the area of state identification and propose original solutions for some Finite State Machine (FSM)/Automata classes with the complexity estimation. Note that some particular network components can also have rather a combinational behavior, such as for example, forwarding devices, and thus we also draw our attention to the logic circuit based testing and related fault models. At the same time, sequential circuits can be considered as scalable representations for FSMs and thus, we also consider testing against sequential circuits. Note that some of the contributions of this work remain purely fundamental and we still do not have interesting case studies for the specifications in question, however, for some others we discuss their applications in the area of network management. Moreover, for particular “network-driven” test purposes, we discuss other – sometimes simpler – testing and verification possibilities that allow under certain assumptions, assure the correct functioning of network components.

Acknowledgements

First of all, I would like to thank the jury members who helped evaluating this work:
Rob Hierons, Pascale Le Gall, Franz Wotawa, Roland Groz, Burkhart Wolff, Tiziano Villa, and Djamal Zeghlache, many thanks for accepting to attend this habilitation defense and giving all your valuable feedback.

Particular acknowledgements go of course to the Reviewers, Rob, Pascale, and Franz, and their comments and suggestions on the manuscript, as well.

Since the very first steps, my research activities were guided by my PhD advisor, Nina Yevtushenko. Thank you Nina, for all your efforts, your enthusiasm and opportunities you have opened. Thanks a lot to the professors and colleagues from my Alma Mater Tomsk State University.

Since my arrival to Télécom SudParis, I was greatly accompanied by Ana Cavalli, Stephane Maag, and Djamal Zeghlache, among other colleagues, of course. Thank you very much for your trust and your kind support.

I would like to thank Benjamin Doerr who helped me all the way through the habilitation process at IP Paris.

All my colleagues and co-authors whom I tried to mention in the manuscript, thank you very much for the hard work. My previous and current students, thank you for the constant motivation.

My dearest family members and my friends, your support is appreciated more than anything, thank you very much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation, context and research challenges

As information and software technologies develop rapidly, more and more attention is paid to the correct functioning of the related software and hardware components. Indeed, for example, in critical systems guaranteeing the correct behavior of software/hardware components and their compositions is crucial. At the same time, the complexity of discrete event systems and in particular, communicating systems that are mainly considered in this thesis, increases as well. The behavior of communicating systems such as communication protocols, services, etc. is often nondeterministic (and sometimes non-observable) and thus, their testing and verification become more challenging.

Following the general tendency in testing such systems, we consider state models as relative specifications [vBP94, DEM+10, LSKP96, Kön12]. We assume that a System under Test (SUT) can accept the inputs and produce the outputs, moving from one state to another. The internal system state is assumed to be unknown (Black box), and testing is performed via the application of input sequences and the observation of the output reactions, with further conclusions about the correctness of the SUT. Non-intrusive monitoring strategies, on the contrary, aim at omitting the first step, i.e., no inputs are applied, the system behavior is only being observed and in this case, the community sometimes refers to it as *passive testing* (see, for example [LNS+97, CMDO09, CRM12, ACC+04, CGP03, MHN18]).

If the SUT can be adequately modeled by a Finite State Machine or an Automaton, then the general testing steps should, on the one hand, check the output on each transition, and, on the other hand, verify that the implementation reached the state that was

expected. In fact, *classical* FSM-based testing techniques are known to start with the checking sequence derivation [Hen64]. One can note three main steps that test generation strategies usually rely on: 1) to reach a certain designated state (from the initial one), 2) at a given state, to traverse (to cover) the outgoing transitions under each input, and 3) to distinguish the state that was reached from all others. To execute these steps, state identification sequences for the specification machine are usually applied [LY94, LY96]. Homing and synchronizing sequences are used to identify the final or current state of the machine, while distinguishing sequences are applied to uniquely conclude about its initial state. Such sequences are well studied for complete deterministic FSMs and Automata, where the first work of Moore is related to the 50-s of the last century [Moo56]. At the same time, for nondeterministic and moreover non-observable and/or partial specifications, there are some gaps in the literature for the state identification problems, that this work aims to cover. In particular, this comes to the first three **Research Challenges** (partially) addressed in the manuscript (and related publications):
RC1: Methods and techniques for deriving homing, distinguishing and synchronizing sequences for nondeterministic and non-observable specifications should be provided;
RC2: The length of the state identification sequences should be assessed, including the reachability of the upper bounds;
RC3: The complexity of the existence check and derivation of homing, distinguishing and synchronizing sequences for nondeterministic specifications should be evaluated.

Note also that from the practical point of view, not always the upper bounds established for **RC2** are reachable; same for the complexity classes and related completeness/hardness of the problems in **RC3**. Therefore, another branch of research directions of the author (with the colleagues) is also devoted to studying specific FSM classes where the worst complexity upper bounds are not reachable.

Obviously, state identification problems are essential for *stateful* systems, however, in reality some communication components, such as for example, forwarding devices, can be still adequately modeled with a single state. In this case, the output which is produced at a given time instance only depends on the current input and does not depend on the state, i.e., the behavior is not sequential but rather combinational. Generally, such systems can be modeled as combinational logic circuits and their testing relies on covering the paths in the specification circuit which go from primary inputs to primary outputs and consists of traversing related logic gates. Mutation testing can be effectively applied in this case, when certain faults are inserted directly in the combinational logic and an input pattern that detects this fault is derived. One of the well known faults in this case are *Single Stuck-at* faults when a given gate output is assumed to be stuck at Constant 1 or Constant 0 (see,

for example [KPKR95, MMS15, Pat05]). Related fault models have shown their effectiveness in hardware testing, however their applicability to testing communication components still remains challenging. At the same time, logic circuits can serve as a scalable representation for the FSMs and can be considered at specifications at a lower abstraction level. However, in this case, sequential circuits are considered where the configurations of the latches correspond to FSM states.

Therefore, novel (networking related) fault models and their expressiveness w.r.t. the well studied single stuck-at faults is also interesting to investigate. This work covers the aforementioned issues through addressing the following challenges:
RC4: The effectiveness of the well studied fault models for logic circuits should be evaluated for the up to date network components;
RC5: Novel fault models for logic circuit testing should be proposed; their effectiveness should be evaluated w.r.t. the single stuck-at faults.

As mentioned above, the main application area of the results is the area of networks and their analysis. Moreover, *classical* or traditional networks and their modeling with state or transition systems have been largely studied (for example, the work [vBP94] summarizes well the related achievements in 90-s of the previous century). We thus make an attempt, through the collaborations with networking research groups, mainly at Télécom SudParis and Airbus Defence and Space, to provide novel model based test generation strategies for dynamic and programmable networks. In this case, the general underlying topology is given beforehand, and this topology mostly describes the physical layer or the fixed non-changeable resources. The virtual links can appear and disappear w.r.t. these physical/resource possibilities. Otherwise, a link can be always kept but its parameters (weights) can dynamically change their values. For example, a bandwidth that is assigned to a given link can be changed dynamically[1]. These networks and their related testing and verification techniques are currently developing and in this work, we aim at addressing the following related challenges:
RC6: Novel verification strategies should be proposed for programmable and dynamic networks;
RC7: Once verified the dynamic/programmable network description, one should assure the implementation indeed conforms to the user request, i.e., novel MBT techniques should be proposed for that matter.

[1] In this case, we assume the potential changes in time, when the parameter values belong to a certain (finite) set.

This work (partially) addresses the challenges mentioned above and presents the related contributions that were obtained by the author in collaboration with the researchers from Télécom SudParis, the Ivannikov Institute for System Programming of the Russian Academy of Sciences, Tomsk State University, Sabanci University, American University of Sharjah, National Taiwan University, and Airbus Defence and Space. In Conclusions, the author (critically) evaluates these contributions and discusses the future research challenges.

1.2 Major contributions

The main contributions of this work can be informally divided into two parts: i) fundamental studies devoted to test generation and monitoring strategies for reactive systems, that can be described as automata or FSMs (with the guaranteed fault coverage, whenever possible); ii) MBT and monitoring strategies for programmable and dynamic networks, i.e., the application of (some fundamental) results. Note that in many cases, contributions from the first part stay rather theoretical as when testing network components in the second part, the specifications quite often get to be much simpler, or on the contrary, much more complex. We nonetheless, continue working in both directions, on the one hand, studying various specifications for possibly nondeterministic and partial reactive systems and solving MBT related problems, and on the other, we continue deriving and applying formal models and methods for up to date network components, using existing test generation strategies and developing novel approaches.

The main contributions of this work can be therefore summarized as follows:

- We provided novel techniques for deriving state identification sequences for nondeterministic, possibly non-observable FSMs; for some classes of input/output automata we also provided the methods for deriving homing and synchronizing sequences.

- We filled in some gaps in the estimation of the complexity of the existence check and derivation of state identification sequences for nondeterministic FSMs; note that in a general case, the existence check problem remains PSPACE-complete for preset sequences.

- We estimated the length of the state identification sequences and showed that differ-

ently from deterministic FSMs, moving from preset to adaptive experiment strategy[2] would not necessarily help to get a polynomial length of the sequence of interest.

- Given the *unpromising* complexity results, we studied and identified specific classes of nondeterministic machines where the worst complexity upper bounds are not reachable.

- At a lower abstraction level, we proposed novel fault models based on logic circuits and investigated their effectiveness and their correlation with the known single stuck-at faults.

- We investigated the applicability of the logic circuit based fault models for testing programmable network components, in particular, for testing SDN enabled switches.

- We proposed novel fault models, in fact that omit the specification at all, when testing SDN frameworks; we also adjusted model checking based testing approaches for testing SDN frameworks where the test purposes are related to potential races.

- We proposed verification techniques for dynamic and programmable network topologies, as well as their monitoring strategies using model checking approaches.

1.3 Structure of the manuscript

The manuscript consists of 6 chapters. Chapter 1 contains the Introduction which the Reader currently follows.

Chapter 2 briefly presents the necessary background. We start through introducing model based testing in general. As one of the advantages of the related techniques is their exhaustiveness, we discuss the fault coverage, and to be able to guarantee the latter (in some cases) we come to the fault models. Finally, we introduce the main state based models that are used in this work; those are mainly finite state machines, that being at current state, accept an input, produce an output, moving to the next state. Note that this model requests that each input is followed by an output which is a very strict constraint that however allows providing various testing techniques including efficient state identification. Nevertheless, we also investigated other models, such as input/output automata where the constraint mentioned above is not necessarily present.

[2] In an adaptive strategy, the next input to be applied is chosen based on the output reaction to the previously applied inputs.

Chapter 3 contains some selected original results in the area of state identification for nondeterministic FSMs. We focus on the preset and adaptive strategies for homing, synchronizing and distinguishing sequences. We analyze the complexity of the corresponding existence check and derivation problems and discuss possible reductions for this complexity. In particular, we identify some specific FSM classes where the worst complexity upper bounds are not reachable.

Chapter 4 continues the contributions to the model based testing strategies, however in this case we use a lower abstraction level and come to logic circuits as related specifications. For the systems that have a single internal state, i.e., rather 'stateless' systems, the input uniquely determines the output to be produced, differently from the sequential behavior (as in FSMs for example), where the current state matters. Stateless systems can be modeled by combinational circuits and in fact, there can be found some network components for which the corresponding model is quite adequate. An example is a switch or a forwarding device which given the values of the network packet parameters, forwards it accordingly (the output port is identified). We therefore present some logic circuit based fault models and the original results on the correlation of these models. Both, combinational and sequential circuits are considered, as the latter can also be considered as a scalable representation for an FSM.

Chapter 5 focuses on the networking applications of the model based testing strategies. We study dynamic and reconfigurable networks and discuss how such *dynamicity* can simplify or complicate the testing and verification steps. Some stateful and stateless specifications have been derived for the network components of interest and we discuss the related test objectives. Note that these objectives reflect not only functional aspects, i.e., conformance testing, but also specific non-functional issues that can occur in distributed systems, such as for example, races in channels.

Chapter 6 summarizes the results presented in the manuscript. We also discuss a number of perspectives concerning the current and future research directions in the area of model based testing, its long standing problems such as, for example state identification, and its current applications in distributed systems, for example in (future) networks.

Chapter 2

Preliminaries

This chapter aims at (briefly) presenting the necessary notions and introducing the notations that are utilized in the manuscript.

2.1 MBT and monitoring; fault models, and guaranteed fault coverage

When testing any software or hardware systems, we are usually interested in providing certain guarantees. Ideally, we would like to conclude that a given system *has no bugs at all, no security issues, etc.* However, this task generally is not solvable. At the same time, introducing certain assumptions about the system under test or under verification, can allow drawing some conclusions about the absence of the bugs of certain types, or it can let us assure that a given system possesses certain properties. The more bugs (or the bugs of more types) are guaranteed to be not present is the system, the higher is the fault coverage of the related testing technique. We however cannot provide any guarantees unless we provide a formal specification of the system under test/verification. Therefore, when it comes to MBT, we usually have the System (or Implementation) under Test or SUT/IUT and its formal description - *specification*.

The system specification is used to derive the *test sequences* or *test cases*. A finite set of test sequences or a *test suite*, is applied to an IUT and the output reactions are observed. If the output reactions *observed* are as *expected*[1] by the specification, then the

[1] They can be exactly the same, or for example, included in the set of allowed possible reactions, etc.

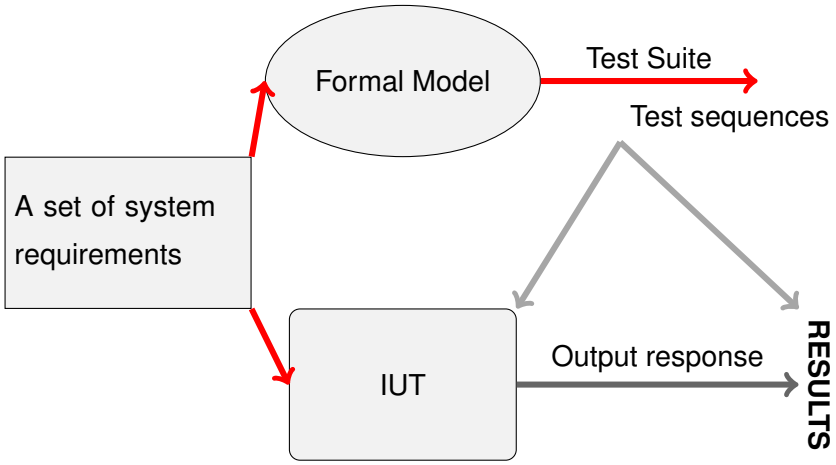

Figure 2.1: MBT general schema

test suite *passed*, otherwise the test suite *failed*. A general schema of the MBT is shown in Fig. 2.1.

Once a system specification is derived, we assume that the implementation is modelled using the same formalism. For example, if a system behavior is described as a finite state machine then an implementation can only have two types of faults, namely transition and output faults; for combinational and sequential circuits as system specifications *classical* faults are, for example, single stuck-at faults. We aim as deriving test suites with the *guaranteed fault coverage*, i.e., we would like to assure that each output and/or transition fault is detected by the test suite TS. However, when it comes to the observation of the output reactions and their comparison with the expected ones, as mentioned above, the conformance relation is what matters. Simply, we can assume the equivalence or equality in this case, but not necessarily - another interesting conformance relation is inclusion or reduction, when an implementation does only what is allowed by the specification and no more than that. Therefore, in order to provide the guarantees when deriving test suites, a fault model is necessary.

A *fault model* [PYvB96] usually is a triple $< \mathcal{S}, @, FD >$ where $\mathcal{S}$ is the specification or the formal description of the system behavior, $@$ represents the conformance relation between an implementation I under test and the specification $\mathcal{S}$, while FD is a fault domain which limits the possible implementations, i.e., $I \in FD$. We are interested in an exhaustive test suite that detects each implementation $I \in FD$ that is not conforming to $\mathcal{S}$, i.e., $I \not@ \mathcal{S}$. Depending on how the implementations from FD are given, one can consider black, white and grey box testing methodologies. Generally, in the case of black box, we know nothing but the maximum number of states of each machine $I \in FD$

which is defined by the corresponding conformance relation.

[Vas73, Cho78, SD88, FvBK$^+$91]; in white box testing scenario, we assume that all the potential faulty implementations are explicitly enumerated [PM64], [KYC14a] and thus, the fault *diagnosis*[2] can be performed at once with testing [GvBD93, EPYvB03]; for the grey box, a mutation machine can be derived [EDYvB12, KPY99, PNR16], such that sub-machines of this machine represent potential faulty implementations. In this manuscript, we are mostly interested in black and white box testing which will be further applied to testing network components.

Note that the testing strategy described above is often referred to as *active* testing, which is, in fact, used to emphasize the possibility of controlling the IUT. Indeed, note that the input sequences are being applied to the IUT in Fig.2.1 and only after the testing conclusions are drawn. This is however, not always possible, in particular when it comes to certain crucial network components that cannot be tested in a stand alone mode. In this case, only observations are available and these observations are made where observation points are available. This is where *monitoring* or *passive* testing becomes of a big help (see for example, [KLCY16], [CRM12, ACC$^+$04, LNS$^+$97, LMM16, Mil98, MA01]).

In the general case, we can assume that input sequences coming to the IUT as well as its output reactions can be observed by the tester or the monitor. For each input/output sequence α/β the tester usually checks that this input/output pair does not violate the specification (this can be done for example, via the simulation of the behavior of $\mathcal{S}$ on the input sequence α). If on the contrary, a violation is observed for a given input/output pair, then the tester returns the verdict '*fail*'. A corresponding input sequence α can be then alerted together with the verdict.

For the testing and monitoring strategies discussed in the work, we mostly consider $\mathcal{S}$ to be described as a corresponding finite state machine or an automaton. The former can be represented by a sequential circuit in a scalable way. For a system that does not have a sequential behavior, the specification $\mathcal{S}$ can be provided as a corresponding combinational circuit. These models are formally introduced further in the manuscript.

[2]Note again, that we do not speak here about the software debugging [WGL$^+$16] directly, however, the latter is possible if there is an established correspondence between (some) faults in the specification $\mathcal{S}$ and those in the source code.

2.2 Finite State Machines and Automata

A *finite state machine* (FSM) [Gil62], or simply a machine (sometimes referred to as an *automaton with output* [TB73]), is a 5-tuple $\mathbf{S} =< S, I, O, h_S, S_{in} >$ where S is a finite nonempty set of states with the set $S_{in} \subseteq S$ of initial states, I and O are finite input and output alphabets, and $h_S \subseteq S \times I \times O \times S$ is a *transition relation*. FSM $\mathbf{S}$ is *non-initialized* if $S_{in} = S$ and in this case, we omit the set S_{in} of initial states and a non-initialized FSM is denoted as a 4-tuple $< S, I, O, h_S >$. FSM $\mathbf{S}$ is an *initialized* FSM if $|S_{in}| = 1$ and FSM $\mathbf{S}$ with the initial state s_j is denoted $\mathbf{S}/s_j$. Otherwise, we write $\mathbf{S} =< S, I, O, h_S, s_0 >$ to denote the fact that FSM $\mathbf{S}$ is initialized with the initial state s_0. If $1 < |S_{in}| < |S|$ then FSM $\mathbf{S}$ often is called *weakly initialized*.

FSM $\mathbf{S}$ is *nondeterministic* if for some pair $(s,i) \in S \times I$, there exist several pairs $(o,s') \in O \times S$ such that $(s,i,o,s') \in h_S$; otherwise, the FSM is *deterministic*. FSM $\mathbf{S}$ is *observable* if for every two transitions (s,i,o,s_1), $(s,i,o,s_2) \in h_S$ it holds that $s_1 = s_2$. FSM $\mathbf{S}$ is *complete* if for every pair $(s,i) \in S \times I$, there exists a transition $(s,i,o,s') \in h_S$; otherwise, the FSM is *partial*. An example of a complete observable nondeterministic FSM (taken from [YKK19]) is shown in Figure 2.2. We will not discuss hereafter what are the reasons for the specifications of interest to be nondeterministic, as in fact this question is rather *philosophical*. We note that in reactive communicating systems, such as telecommunication protocols, services, etc. nondeterministic behavior can be stated directly in the specification, due to possible output responses to the same inputs at the same states (optionality), or it can also come from partial observability and controllability of an SUT [EGGC09]. It is also possible that for the *compactness* of the formal specification of an SUT, its nondeterministic specification is provided, instead of an equivalent deterministic one.

In FSMs, the behavior relation h_S is extended to input and output sequences in usual way and given an input sequence $i_1 \dots i_l$, we say that an output sequence $o_1 \dots o_l \in out(s, i_1 \dots i_l)$ if and only if there exists a state s' such that $(s, i_1 \dots i_l, o_1 \dots o_l, s') \in h_S$. Given an input/output pair io and a state s of a complete FSM $\mathbf{S}$, a subset of states that contains each state s' of FSM $\mathbf{S}$ such that $(s,i,o,s') \in h_S$ is the *io-successor* of state s. The io-successor of state s can be empty and then sometimes we say that the io-successor of state s *does not exist*. A *trace* tr of FSM $\mathbf{S}$ at state s is a sequence of input/output pairs which label consecutive transitions starting from state s, $tr = i_1 o_1 \dots i_l o_l$ (or $i_1/o_1 \dots i_l/o_l$). A sequence $i_1 \dots i_l$ is an *input* sequence of the trace while $o_1 \dots o_l$ is an *output* sequence. If FSM $\mathbf{S}$ is complete and observable, given state s and a trace γ of the FSM at state s, the γ-successor of state s is state s' (in fact, a singleton $\{s'\}$) which is reached from s

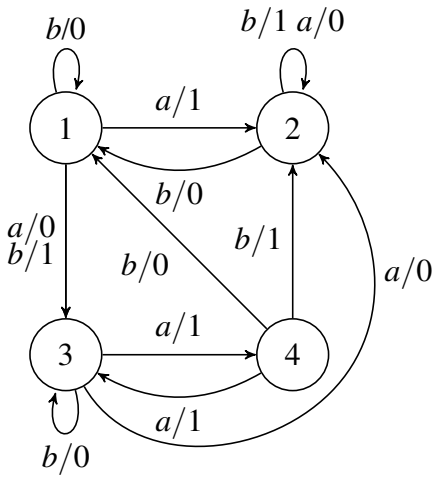

Figure 2.2: Complete observable nondeterministic FSM $\mathbf{S}$

via the trace γ. If FSM $\mathbf{S}$ is non-observable then the γ-successor of state s is the set of all states which can be reached from s via the trace γ. If γ is not a trace at state s then the γ-successor of state s is empty. Given a non-empty subset S' of states and a trace γ of the FSM, the γ-successor of S' is the union of γ-successors over all states of the set S'. For example, for the FSM in Figure 2.2 an $aba/001$-successor of the subset of states $S' = \{1,2\}$ is the set $\{2,4\}$ while an $aba/111$-successor of the set $S' = \{1,2\}$ in empty (or does not exist).

The notion of an FSM is very close to the *Automaton* model that does not support output responses, i.e., automaton transitions are labeled by actions that are not divided into inputs and outputs and usually these actions are called inputs or, simply, $letters$. One may eliminate outputs at each transition of a given FSM in order to get the underlying automaton that can be nondeterministic for a nondeterministic FSM. Note however, that there can be more complex definitions for automata and some of them we utilize in this manuscript. For example, the actions can be split into inputs and outputs, however, differently from an FSM in the automaton behavior not each input is necessarily followed by an output.

A (non-initialized) Input/Output Automaton is a 4-tuple $\boldsymbol{S} =< S, I, O, T_S >$ where S is a finite set of states; I and O are finite non-empty disjoint sets of inputs and outputs, respectively; $T_S \subseteq S \times I \times S \cup S \times O \times S$ is a transition relation where 3-tuples $(s,i,s') \in T_S$ and $(s,o,s') \in T_S$ are transitions. The machine can contain a special output $\delta \in O$ that represents the *quiescence* [Tre96] at the states where only the transitions under inputs are defined. In some sense, it means that no output can be produced after the transition is executed, and this fact is defined by this special output δ. An example of such an automaton is given in Figure 2.3. We note that this automaton is even more specific,

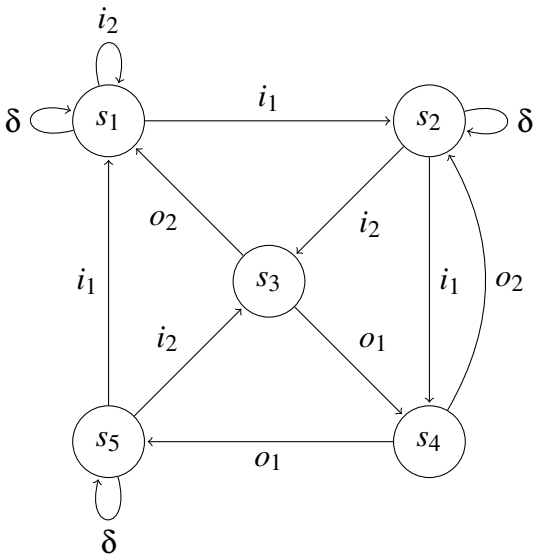

Figure 2.3: Input/Output Automaton ***S***

it contains for example a property that not only inputs and outputs are split into disjoint sets but also when at a state where an input is applied, no output can be produced and vice-versa. We study this special class of automata further in the manuscript.

The FSM and automata models briefly introduced above (and their modifications) will further serve as specifications for testing purposes. We will present some of the analysis problems for the related models and our related contributions in the area, which help improving testing or monitoring strategies for communicating systems.

Chapter 3

State identification in MBT, formal verification and monitoring

The chapter is devoted to the presentation of some existing and original results in the area of state identification of finite state systems. Most of the results that contain the author's contributions, are obtained for (nondeterministic) FSM state identification. Note that the problem of state identification is one of the major problems in FSM based testing and we first present the relevant motivation, referring the Reader to the so-called W-method (and its modifications) [Vas73, Cho78].

3.1 Reachability and distinguishability in FSM based testing and monitoring

Currently, there exist a number of FSM based test generation strategies, and related methods keep improving. For a survey of such methods, the Reader can for example, refer to [DEM$^+$10], where a short description of them is presented, as well as the relevant experimental results (on the test suite length, in particular). All these methods such as HSI, DS, etc. somehow represent an improvement of the so called W-method that was originally proposed by Vasilevskii [Vas73], and later adjusted by Chow [Cho78]. Below, we briefly introduce the W-method and some related constraints for the exhaustiveness of the returned test suite.

The method is used for deriving test suites with the guaranteed fault coverage against

complete deterministic machines. Note that the specification machine should also be reduced (*minimal*), which means that its states are pair-wise distinguishable. The notion of distinguishability will be introduced a bit later as it refers to the initial state identification. In fact, a distinguishing sequence for a pair or a bigger subset of states allows to uniquely determine the initial state of this subset after the application of the sequence (and the observation of the corresponding output response). When the machine is deterministic it means that the output reactions on this sequence are pair-wise different.

Generally, in W-method, given the initialized specification FSM $\mathcal{S}$, one should first reach each FSM state, then traverse each transition at the reached state and in the end, append a so called characterization or W set. The latter consists of the sequences distinguishing each state pair. The sequences that allow reaching each FSM state from the initial one, form the reachability or state cover set, correspondingly. In other words, we first reach a state, assure that the output at the next transition is correct, and then assure that the reached final state in the implementation is exactly the same as it was expected in $\mathcal{S}$[1].

Note that this *original* version of the method has been modified in various ways, for example, in order to reduce the height and/or the width of the resulting tree which is obtained via the concatenation of the related reachability set, the input alphabet and the W set. A general idea behind such optimization is to substitute the W set with a smaller one. If there exists a distinguishing sequence for the specification machine, this sequence can be thus appended instead of the W set (DS method). Otherwise, only *targeted* sets can be appended as well which distinguish the reached state from any others (HSI method, for example). As mentioned above, the Reader may check a survey [DEM+10] that presents various W modifications such as DS, HSI, H, etc., and moreover supports the results with the experimental evaluation. We will skip this part as we mostly contribute to the derivation of the related sets, but we note that the obtained results can be and are utilized in the modified W versions. In the current chapter, we will instead focus on the presentation of the (original) results for the reachability and distinguishability for (nondeterministic) specifications.

When it comes to system verification [Hol03] which generally consists of checking certain properties over it, or otherwise, when performing a non-intrusive monitoring of an SUT, reachability analysis can be also of help. In fact, knowing a current state of the system can in some cases reduce the verification/monitoring efforts. Assume that for a given SUT a set $P = \{p_1, p_2, \ldots, p_k\}$ of properties should be checked. Assume also that the specifi-

[1] This is done under the assumption that the number of implementation states is no bigger than that of the specification.

cation $\mathcal{S}$ contains n states $s_1, s_2, \ldots, s_n$, respectively. Not all properties can be relevant for checking at state s_j, and a subset of properties $P_j \subseteq P$ can be defined (of those that indeed should be checked). In this case, when verifying the system behavior we could first bring the system to a known state s_j and after that check the subset $P_j \subseteq P$. If the system is tested passively, and no input stimuli are allowed, then the state s_j can sometimes be deduced from the SUT input/output traces being observed. And later on again only relevant properties would be checked.

As the last point forms one of contributions of the author, in particular, on the use of the final state identification for improving monitoring strategies [KLCY16], we will give more details here and illustrate the idea on an example. In fact, let us consider a *Simple Connection Protocol (SCP)* for that matter. The SCP is a protocol designed to 'connect' two entities, negotiating the quality of service at the connection establishment [ACC$^+$04]. The SCP allows connecting an entity called the *upper layer* to an entity called the *lower layer*. The upper layer dialogues with SCP to fix the quality of service (QoS) desirable for the future connection. Once this negotiation succeeds, the upper layer comes to the lower layer requesting the establishment of a connection satisfying the quality of service previously agreed on. The lower layer accepts or refuses this connection request. If the lower layer accepts the request, then it informs the upper layer that the connection was established and the upper layer can start transmitting data. Once the transmission of data is finished, the upper layer sends a message to close the connection. If the lower layer refuses the connection, the system allows the upper layer to make three requests before informing the upper layer that all the connection attempts failed. If the upper layer would like to be connected to the lower layer, it is necessary to restart the QoS negotiation from the beginning. After the connection gets established (accepted), the upper layer can send data to the lower layer with a guaranteed QoS. Each time the upper layer sends any data, the lower layer acknowledges the total amount of received data. The FSM describing the behavior of this protocol is shown in Figure 3.1. Note that this FSM is more complex than that presented in Section 2, as it contains context variables, predicates and input parameters. It is an *Extended* machine whose formal definition we omit due to the room constraints; we invite the Reader to see such a definition, for example in [PBG04]. These machines sometimes allow to represent the SUT behavior in a more compact way, *partially hiding* the states, for example, in the vector of context variable values. We deal with a similar model in Chapter 5, namely we construct extended automata when analyzing the possibilities of races in distributed architectures.

For the SCP, we use an Extended FSM (or EFSM for short) that has the following inputs and outputs [ACC$^+$04]. The upper layer can request the desired QoS level with the mes-

sage $req(QoS)$ with QoS in the range $[0,3]$. The lower layer replies whether it can support the desired QoS level or not with the messages $nosupport(QoS)$ or $support(QoS)$. The upper layer then issues the message $conn$ (an output) trying to establish the connection. The replies can be:

- $accept(QoS)$ if the connection guarantees QoS,
- $refuse$ if the lower layer is busy,
- $abort$ if more than two refused attempts have occurred.

The upper layer then can issue the $data(size, value)$ message to transmit the data. Each data message is acknowledged with the message $ack(DataCountOut)$. At any point, if the upper layer decides to end the connection, the message $reset$ can be sent. The $reset$ message should be replied with an $abort$ message by the lower layer. Finally, any input at a wrong state should be replied with an error message (err).

Assume that for the SCP protocol we need to check the following properties.

- *Receipt confirmation*: When some data are sent to the lower layer, an acknowledgement (ack) should be sent to the upper layer, i.e., the following input/output sequence should be observed: $data/ack$.
- *Connection attempt management*: At least two refused attempts should be allowed before definitely rejecting the upper layer connection, which corresponds to the $conn/refuse.conn/refuse.connect/abort$ input/output sequence.
- *Client communication termination*: Any successful connection must be properly terminated; the upper layer sends a $reset$ after the connection was established, and before requesting a new QoS or connection, the $abort$ message has to be produced. Thus, the following input/output sequence should be observed: $conn/accept(QoSOut).-$ $.reset/abort$.

As mentioned above, not all the properties are pertinent to each EFSM state. Therefore, when monitoring the SUT, after determining its current state only properties that are related to that state can be checked. For example, if the IUT is at state s_3 the first property should be checked. However, if the SUT is at state s_2, there is no need to check this property. At the same time, there exist many non-functional requirements that can be checked

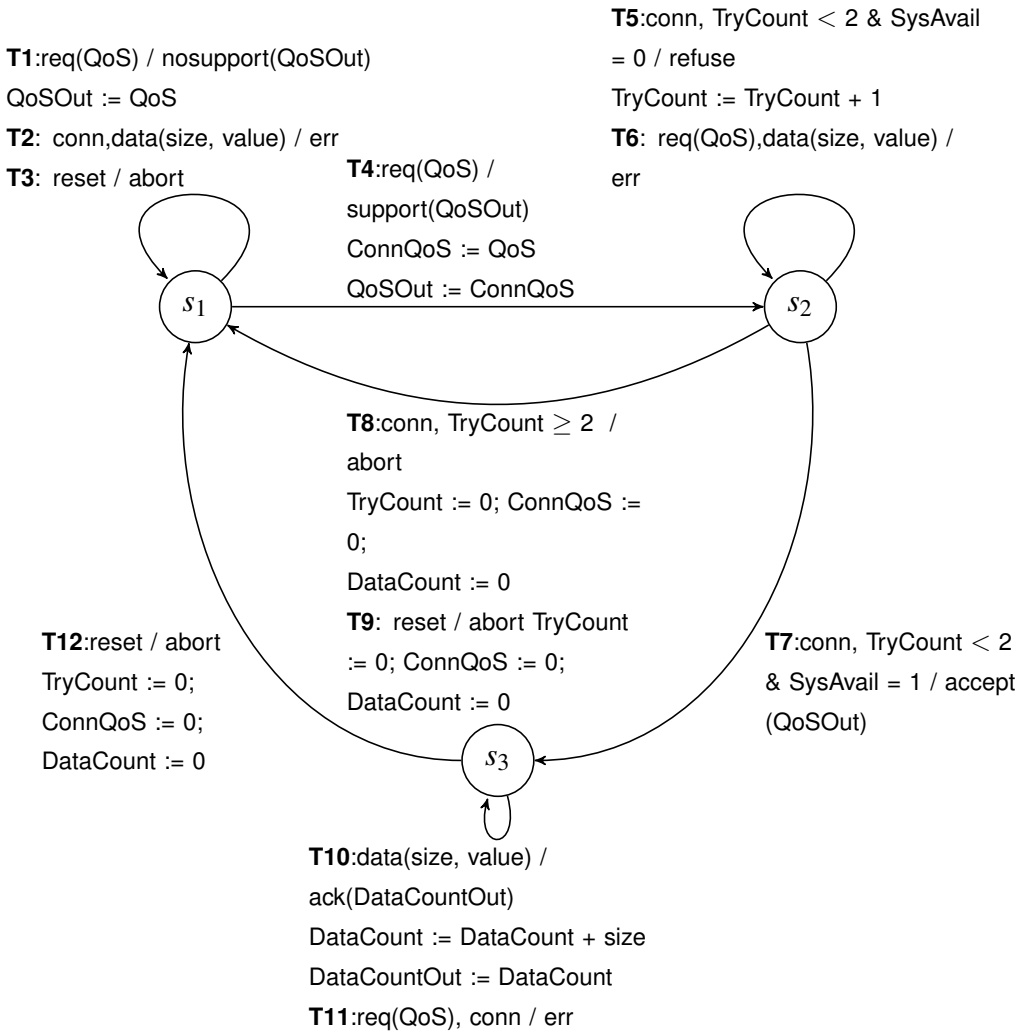

Figure 3.1: Extended FSM for the SCP

at state s_3 - one can verify, for example, that the length of transferred data does not exceed the available disk space, or that the QoS provided is indeed the one guaranteed at the QoS negotiation phase. Note that checking such non-functional properties while the SUT is not at state s_3 puts an unnecessary load on the monitoring system. Determining the current state of the SUT can be done through construction of possible homing and/or synchronizing sequences, for example, up to a given length. Further, during the system monitoring the observed sequence is compared to the pre-built ones, and if two coincide, then the current state is uniquely determined and only relevant state properties are thus checked.

Note that in some cases, even the reducing of uncertainty of the current system state, can already minimize the monitoring or verification efforts, and for that matter so called S'-synchronizing sequences can be used [San05] (the final state is not necessarily unique

but definitely belongs to the set S'). We later on present some contributions that were obtained in the area of state identification for nondeterministic specifications, that can thus help, optimizing testing and/or verification efforts.

3.2 'Gedanken' experiments: problem statement and related work

Distinguishing, homing and synchronizing experiments for FSMs form a set of so called 'gedanken' experiments [Moo56] that have been introduced by Moore, in the middle of the previous century. An experiment in this case is the process of applying an input sequence (or a set of those), observing an output reaction of the machine under experiment (if needed) and drawing a conclusion about the final or the initial state of this machine. As mentioned above, homing and synchronizing sequences (HS and SS, respectively) are used to identify the final or the current state of the machine, while distinguishing sequences (DS) are built for the initial state identification. The way the experiment is carried out defines the type of the experiment, which can be either *preset* or *adaptive*; the experiment is adaptive if the next input to be applied is chosen based on the previously observed outputs, and the experiment is preset if the outputs need to be observed only at the end of the experiment, or need not to be observed at all. A preset experiment is thus represented by just an input sequence, whereas an adaptive experiment is represented by a tree-based structure which can be referred to as an adaptive sequence or a test case [LY94, PY05]. Therefore, when it comes to state identification for FSMs and automata, usually we are interested in the existence check of the corresponding preset or adaptive experiment; further, a derivation strategy for such experiment should be proposed. Alongside, the complexity of both problems is another challenging issue, as well as the length of the corresponding sequences or the size of the relevant tree-based structures. Note that studying these tasks for nondeterministic FSMs/automata and some their classes is one of the main areas of interest of the author. This chapter therefore briefly presents the related contributions, and to better highlight those, we first start with a short presentation of the related work.

The problems of checking the existence and derivation of state identification sequences have been widely investigated in the past seventy years. Major results obtained in this area mainly concern the deterministic FSM case: for non-initialized complete deterministic minimal machines the existence decision and derivation of an appropriate sequence for

the final state identification (HS and SS) can be performed in polynomial time [San05]. Moreover, in some cases, such sequences always exist, such as for example, an HS of polynomial length (w.r.t. the number of FSM states) for a complete deterministic connected reduced FSM.

However, even for the 'good' cases of HS and SS for deterministic FSMs the problem becomes much harder when it comes to constructing a shortest HS or SS. Indeed, the problem of deriving a shortest HS/SS is NP-hard even for complete non-initialized deterministic minimal FSMs [San05].

DSs make the above problems much harder; indeed, even for complete deterministic FSMs, the existence check of a DS is PSPACE-complete [LY94]. Note that in some cases, the DS existence check complexity can be reduced via an adaptive strategy. A remarkable example of such complexity reduction for complete deterministic FSMs has been proposed in [LY94] where the existence check of an adaptive distinguishing sequence has been proven to be solved in polynomial time with respect to the number of FSM states.

For nondeterministic FSMs, the paper [ACY95] can serve as another example - the existence check of an adaptive distinguishing sequence for a pair of states of an observable machine can also be solved in polynomial time. Since then, adaptive strategies gained a lot of attention, as the complexity in some cases can be indeed reduced dramatically and some of the contributions presented below, cover adaptive strategies for the state identification, with the polynomial complexity.

For nondeterministic machines, again, the problems listed above become harder. Indeed, as shortest synchronizing sequences can have exponential length in this case [IS04], their existence check and derivation cannot be performed in polynomial time. At the same time, distinguishing a state pair in an observable nondeterministic FSM can also require a sequence of an exponential length [SEY07] (it is linear for complete deterministic FSM).

When it comes to the derivation techniques for preset and adaptive state identification experiments, existing strategies, in our opinion, can be classified into three large groups. Methods of the first group follow a classical successor tree based approach. Techniques from the second group rely on an iterative derivation of the experiment of interest; induction base in this case is usually a set of sequences merging or distinguishing (separating or splitting) state pairs. The third group of techniques is based on the use of various solvers for the final state identification[2]. Successor tree based approaches have been

[2]The author partially contributed to all these groups for nondeterministic machines and their 'gedanken' experiments.

largely investigated for deterministic complete automata and FSMs; the interested Reader can for example refer to [San05] where such methods are well presented for homing and synchronizing sequences. Tree based approaches can be utilized when constructing distinguishing sequences as well, moreover, not only a successor tree but also a splitting tree can be exploited [LY96]. The difference between these two is that in a splitting tree, nodes can be labelled by input sequences or by related subset of states; successor tree *unrolls* the behavior of the machine and usually its nodes are labelled by state subsets. Note that splitting tree approaches are utilized for deterministic specifications to make sure that such state subset partitioning is always possible.

This group of techniques, in particular, was studied by the author during her PhD [Kus13][3] at Tomsk State University, Russia, under the supervision of Prof. N. Yevtushenko. In fact, methods for deriving homing and distinguishing experiments were proposed (also published in [KEY11]) and it has been shown that for nondeterministic but observable FSMs, the truncated successor tree can be constructed in the following way. The root of the tree for a complete FSM $\mathbf{S} =< S, I, O, h_S >$ is labeled with the set of the pairs $\overline{s_p, s_q}$, where $s_p, s_q \in S$, $s_p \neq s_q$; the nodes of the tree are labeled by sets of pairs of the set S. Edges of the tree are labeled by inputs and there exists an edge labeled by i from a node P of level j, $j \geq 0$, to a node Q such that a pair $\overline{s_p, s_q} \in Q$ if this pair is an io-successor of some pair from P. The set Q contains a singleton if io-successors of some pair of P coincide for some $o \in O$. If the input i distinguishes each pair of states of P, then the set Q is empty. Given a node P at the level k, $k > 0$, the node is *terminal* if one of the following conditions (truncating rules) holds.

Rule-1: P is the empty set.

Rule-2: P contains a set R without singletons that labels a node at a level j, $j < k$.

Rule-3: P has only singletons.

These rules allow establishing the conditions for existence check and derivation of an HS for a nondeterministic FSM. Given a path of the truncated successor tree from the root to a node labeled with the set of singletons or with the empty set, the input sequence α that labels this path is an HS for the FSM $\mathbf{S}$. At the same time, if the successor tree has no nodes labeled with a set of singletons or with the empty set, i.e., is not truncated using Rules 1 or 3, then FSM $\mathbf{S}$ is not homing.

The same technique can be used for checking the existence and derivation of an SS for the FSM $\mathbf{S}$, the last rule just requires that all the singletons are the same. Similarly, when it is used for deriving a DS, Rule 3 checks for containing a singleton, and the corresponding

[3] This part therefore appears in the related work and not in the contributions of the current thesis.

branch thus becomes unpromising. If the successor tree has no nodes labeled with the empty set, i.e., is not truncated using Rule 1 then there is no distinguishing sequence for $\mathbf{S}$.

In iterative derivation approaches (second group), state identification sequences are usually pre-calculated for state pairs; in fact, for deterministic automata and their SSs it also serves as a criterion for the existence check [Nat86, Epp90]. Once state identification sequences are derived for each state pair, one can start with the first state pair and apply the corresponding sequence, then another (third) state is added to the set of states. The behavior of the machine is simulated on the first sequence and the related state successors are computed. Whenever these successors are not singletons, the corresponding state identification sequence for the resulting pair is appended to the initial sequence. Another state is then added to the set of states and the process is repeated iteratively. Such approach allows building HSs and SSs, accordingly, moreover, for SSs it automatically proves the cubic upper bound on their length [San05, Epp90].

Currently, the application of solvers for the existence check and/or derivation of state identification sequences becomes very popular. For an SS derivation of potentially partial nondeterministic automata, the interested Reader can refer to [SV19, SV18], where the problem is reduced to SAT solving via a corresponding encoding of inputs and transitions of the machine. SAT and constraint solving has been also applied for the checking sequence generation [NPR18, PAGO19]. In this case, a checking sequence is utilized to distinguish a potential mutant from the specification and to some extent can be treated as a distinguishing sequence for a direct sum of the specification and implementation FSMs, i.e., a sequence identifying the initial state of the machine with two initial states. Note that in the cited works even non-classical FSMs are considered, namely the authors take into account symbolic inputs and outputs.

Below, we present some of the original results, in the area of the existence check and derivation of preset and adaptive homing, distinguishing and synchronizing sequences for nondeterministic FSMs and automata.

3.3 Novel results in the area of (adaptive) experiments for initial and final state identification

3.3.1 Deriving preset and adaptive homing, distinguishing, synchronizing experiments for nondeterministic FSMs/Automata

Deriving preset HS, DS and SS for nondeterministic FSMs - successor tree based approach

Similar to the existence check and derivation approaches for the state identification, we start with the successor tree. As mentioned above, first results were obtained during the PhD of the author and thus, were listed as a related work. Hereafter, we note that these results are very well adjustable for the case of non-observable machines.

In particular, in [KLCY16] we were interested in deriving a set of non-redundant homing sequences of length l or less, for a subset $S' \subseteq S$, $|S'| > 1$, of a given FSM $\mathbf{S}$ that can be non-observable. For that matter, one can build a similar truncated successor tree. However, this time the root of the tree is labeled with the set S', while the nodes of the tree are labeled by state subsets such that none of them is a proper subset of another. Edges of the tree are labeled by inputs and there exists an edge labeled by an input i from a node labeled by P at level j, $j \geq 0$, to a node labeled by Q if Q is the set of i-successors of all subsets of P. The set Q contains a singleton if non-empty io-successors of some subset of P coincide for some $o \in O$ or if corresponding non-empty io-successors of some subset of P do not intersect. Sets which are proper subsets of other items are deleted from the set Q.

Given a node labeled with the set P at the level k, $k > 0$, the node is terminal if one of the following conditions holds.
Rule-1: P contains only singletons.
Rule-2: The depth of the node P is greater than l.

If the successor tree has no nodes labeled with the singletons only, then there is no homing sequence of length l or less for the subset S'. Otherwise, each path to a node labeled with the set of singletons is added to the set of non-redundant homing sequences.

An example of such successor tree and related HSs of length up to two is shown in Figure 3.3. This tree is produced for the *FSM slice* of the Extended FSM in Figure 3.1.

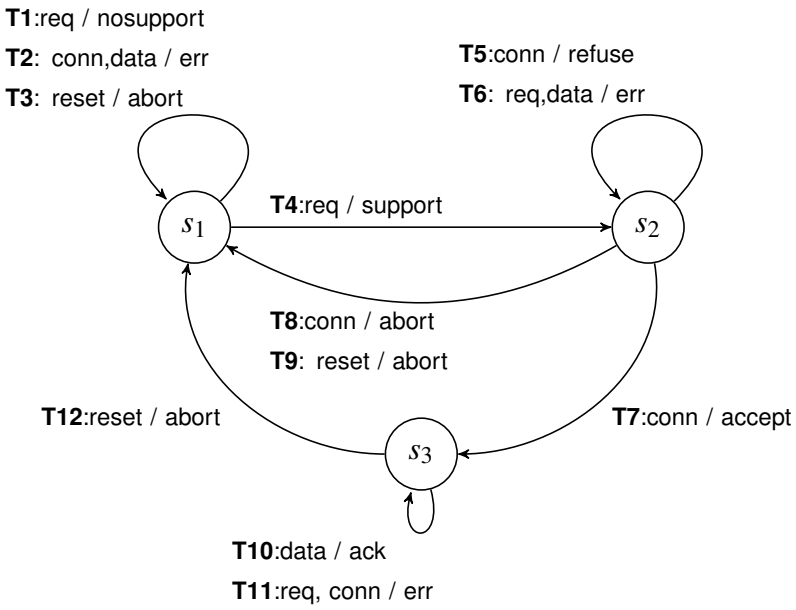

Figure 3.2: FSM slice for the EFSM in Figure 3.1

The latter is obtained after deleting from a given EFSM all the predicates, parameters and update functions (the slice itself is shown in Figure 3.2).

According to the successor tree in Figure 3.3, the resulting set of (non-redundant) homing sequences for the FSM in Figure 3.2 is the following:
$\{req.reset, req.conn, req.data, reset, data.rec, data.conn, data.reset, conn.req, conn.data, conn.reset\}$.
This set can be used for reducing the monitoring efforts as discussed in Section 3.1.

An interesting remark that we would like to draw the Reader's attention to, is that once obtaining the set of all singletons for the observable machine, we can prolong the corresponding input sequence if we need to. However, for the non-observable machines this is not necessarily the case, indeed, not each prolongation of an HS remains an HS - this is due to the fact that a singleton can produce a state pair, triple, etc. after an application of a given input[4].

Note that this approach can be applied for generating synchronizing sequences, all the singletons obtained due to Rule-1, should be the same; only in this case the node is declared terminal.

For generating distinguishing sequences for non-observable FSMs, one can build a similar successor tree and make sure that for each node there is no merging at the same

[4]One can however put any input as a prefix to any HS for a non-initialized non-observable machine, without loosing the necessary property.

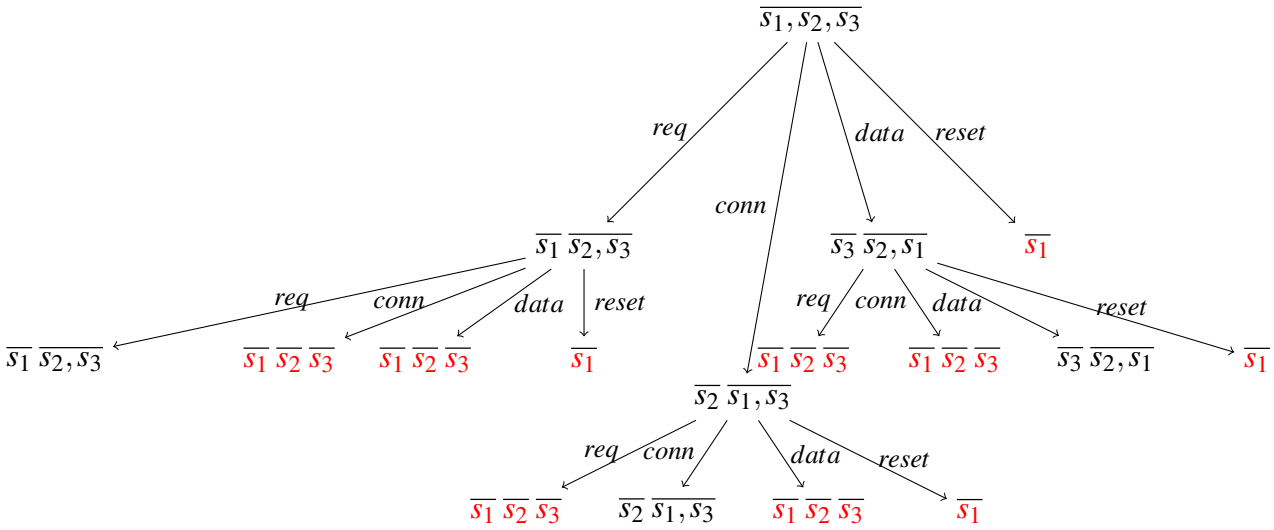

Figure 3.3: Truncated successor tree for the FSM in Figure 3.2

state. Otherwise, such a node should be declared terminal as well. Another option is to proceed, as discussed above, considering the subsets of state pairs, putting in the root all the pairs of set S' and terminate when a node is labelled by the empty set. The sequence that labels a path to such a node is a DS or a separating sequence for the given nondeterministic, possibly, non-observable machine.

Alternative strategies for preset state identification sequences - reducing the problem to SS in automata

The successor tree built with subsets of state pairs can be naturally considered for deriving a transition diagram of a special automaton whose states are pairs of states of the FSM $\mathbf{S}$ together with the designated state $sink$. Note that there are no outputs in this automaton, nor any quiescence, i.e., this is a *classical* automaton considered in the works related to SS derivation [Vol08, IS04, SV18, Mar14]; actions (letters) of the automaton in question correspond to the FSM inputs.

The transitions of the automaton are defined according to the tree branches where the state $sink$ is reached under input i when a corresponding pair is distinguished by the input i or for some $o \in O$, the io-successor is a singleton. An HS exists for the FSM $\mathbf{S}$ if and only if the obtained automaton has a synchronizing sequence to the $sink$ state. The latter thus provides an alternative way to derive (all) homing sequences for a given FSM. Note that a similar automaton can be derived for the set of all distinguishing sequences,

the automaton can become partial when, given a pair $\overline{s_p, s_q}$, for some input i there exists o such that the io-successor of $\overline{s_p, s_q}$ is a singleton. We further present the strategies for deriving an automaton, that contains either all HSs or all DSs for a given complete nondeterministic observable FSM.

Consider a nondeterministic FSM $\mathbf{S} =< S, I, O, h_S >$, $S = \{s_1, \ldots, s_n\}$, in [KY15c], we propose to derive an automaton ${S^2}_{home}$ such that the set of all synchronizing sequences of this automaton coincides with the set of all homing sequences of FSM $\mathbf{S}$, i.e., $L_{home}(S) = L_{synch}({S^2}_{home})$.

Algorithm 1: Deriving the automaton ${S^2}_{home}$

Input : A complete observable nondeterministic FSM $\mathbf{S} =< S, I, O, h_S >$

Output: Automaton ${S^2}_{home}$

States of ${S^2}_{home}$ are pairs $\overline{s_j, s_k}$, $j < k$, and the designated state $sink$ while actions are inputs of the FSM $\mathbf{S}$ (letters);

for *each input* $i \in I$ **do**

  for *each state* $\overline{s_j, s_k}$ *of the automaton* ${S^2}_{home}$ **do**

    if $\{s_p, s_t\}$ *is the* io*-successor of the set* $\{s_j, s_k\}$ *for some output* $o \in O$, $p < t$ *and* $j < k$ **then**

      Add to the automaton ${S^2}_{home}$ the transition ($\overline{s_j, s_k}$, i, $\overline{s_p, s_t}$)

    if *for each output* $o \in O$ *the* io*-successor of the pair* $\overline{s_j, s_k}$ *is a singleton or states* s_j *and* s_k *are distinguished by the input* i **then**

      Add to the automaton ${S^2}_{home}$ the transition ($\overline{s_j, s_k}$, i, $sink$)

By construction, the automaton ${S^2}_{home}$ has the following features. First of all, ${S^2}_{home}$ can be nondeterministic. Moreover, for a sequence $\alpha \in I^\star$, a pair $\overline{s_p, s_q}$, $s_p, s_q \in S$, $s_p \neq s_q$, α takes the automaton ${S^2}_{home}$ from the pair $\overline{s_p, s_q}$ to the $sink$ state if and only for each trace $\gamma \in (IO)^\star$ with the input projection α, the γ-successor of $\overline{s_p, s_q}$ is the empty set or a singleton. Therefore, the following statement holds.

Proposition 1 *An input sequence* α *is a homing sequence for the FSM* ***S*** *if and only if* α *is a synchronizing sequence for* ${S^2}_{home}$.

The set of all homing sequences of the FSM $\mathbf{S}$ is thus equal to the set of all synchronizing sequences of the automaton ${S^2}_{home}$, i.e., $L_{home}(\mathbf{S}) = L_{synch}({S^2}_{home})$. As an example, consider an FSM with a flow table in Table 3.1.

Table 3.1: Example FSM $\mathbf{S}$

Input/State	0	1	2	3
i_0	$3/(0,3)$	$1/(0,3)$	$2/(0,3)$	$3/(0,3)$
i_1	$1/(0,2)$ $1/(0,3)$ $1/(2,3)$	$0/(0,2)$ $0/(0,3)$	$2/(0,2)$ $2/(1,2)$ $2/(2,3)$	$3/(0,3)$ $3/(1,3)$ $3/(2,3)$
i_2	$2/(0,1)$ $2/(0,3)$ $2/(1,3)$	$2/(0,1)$ $2/(0,3)$ $2/(1,3)$	$0/(0,1)$ $0/(0,3)$ $1/(1,3)$	$3/(0,3)$ $3/(1,3)$ $3/(2,3)$ $1/(0,1)$

Note this example is interesting as it shows an exponential upper bound on the length of a homing sequence[5]. Table 3.1 contains transitions of the corresponding FSM $\mathbf{S}$ with four states $0,1,2,3$, the set $I=\{i_0,i_1,i_2\}$ of inputs, and the set $O=\{(j,k),(j<k)\&(j,k\in\{0,1,2,3\})\}$ of outputs. The sequence $\alpha=i_0i_1i_0i_2i_0i_1i_0$ is an HS for the FSM $\mathbf{S}$, and this HS is the shortest in this case.

The flow table of the automaton $S^2{}_{home}$ returned by Algorithm 1 is shown in Table 3.2.

Table 3.2: $S^2{}_{home}$ automaton for the FSM $\mathbf{S}$

Input/State	$\overline{0,1}$	$\overline{0,2}$	$\overline{0,3}$	$\overline{1,2}$	$\overline{1,3}$	$\overline{2,3}$	*sink*
i_0	$\overline{1,3}$	$\overline{2,3}$	*sink*	$\overline{1,2}$	$\overline{1,3}$	$\overline{2,3}$	*sink*
i_1	$\overline{0,1}$	$\overline{1,2}$	$\overline{1,3}$	$\overline{0,2}$	$\overline{0,3}$	$\overline{2,3}$	*sink*
i_2	*sink*	$\overline{0,2}$ $\overline{1,2}$	$\overline{2,3}$ $\overline{1,2}$	$\overline{0,2}$ $\overline{1,2}$	$\overline{1,2}$ $\overline{2,3}$	$\overline{0,1}$ $\overline{0,3}$ $\overline{1,3}$	*sink*

By direct inspection, one can assure that for the automaton $S^2{}_{home}$, a shortest synchronizing sequence is also the sequence $i_0i_1i_0i_2i_0i_1i_0$. Any prolongation of this sequence is a synchronizing sequence for the automaton $S^2{}_{home}$ and a homing sequence for the FSM $\mathbf{S}$.

In order to describe the set of all distinguishing sequences one can rely on the same technique with a very slight difference: when for some $o\in O$, the io-successors of states of some pair of P coincide there is no edge from the node labeled by i to the next tree

[5]The class of the machines with the reachable exponential upper bound on the length of an HS was proposed and studied in [KY13, Kus13].

level in the corresponding successor tree. Indeed, when building the automaton $S^2{}_{dist}$, for each input $i \in I$ and each state $\overline{s_j, s_k}$ of the automaton $S^2{}_{dist}$ we check if states s_j and s_k are distinguished (separated) by this input; and if so, we add the transition ($\overline{s_j, s_k}$, i, $sink$). If for each $o \in O$, the io-successors of states s_j and s_k do not coincide and $\{s_p, s_t\}$ is the io'-successor of the set $\{s_j, s_k\}$ for some $o' \in O$, $p < t$ and $j < k$, we add to $S^2{}_{dist}$ the transition ($\overline{s_j, s_k}$, i, $\overline{s_p, s_t}$), respectively. The automaton can be partial, since for some pair $\overline{s_j, s_k}$, $j < k$, of states of FSM $\mathbf{S}$, there can be no transition under input i if states s_j and s_k have the same nonempty io-successor for some output o. The Reader can refer to [KY15c] for more details, examples, and related algorithms and propositions.

Deriving Homing and Synchronizing sequences for Input/Output Automata

The automata that were considered above represent a very particular case, as the actions are not divided into inputs and outputs, a potential quiescence at a state is not considered, etc. In fact, synchronizing experiments are mostly studied for this exact class of automata (see for example, [IS04, San05, SV19, Vol08, ÇKY+18] and among the long standing problems for this class there is conjecture that was recently proven. Namely, this is the Cerny conjecture which concerns complete deterministic automata (without outputs) and in particular, the length of a shortest SS. The conjecture claims it is quadratic w.r.t. the number of states of the automaton while it is proven to be no more than cubic [Vol08]. Trahtman recently uploaded his paper to archive where he presented a proof of this conjecture (currently the archive contains the 4-th version of the paper) [Tra20]. Note again that in the aforementioned works, there is a very restricted class of automata, however in testing applications it is more convenient to deal with a reactive system when inputs and outputs are separated. Therefore, in one of our works (together with the researchers from ISP RAS, Russia) we considered a little more generic case of automata which we describe below[6].

We consider Input/Output Automata briefly introduced in Chapter 2 (see an example automaton in Figure 2.3). Moreover we restrict the automata class under consideration, namely, we are interested only in those input/output automata for which the following holds:

1. At each state only inputs or only outputs are allowed, i.e., $S = S_1 \cup S_2$, $S_1 \cap S_2 = \emptyset$ and $T_S \subseteq S_1 \times I \times S \cup S_2 \times O \times S$;

2. The transition diagram does not contain cycles/loops labeled with outputs, i.e., the

[6]The results are published in [KYBK18].

language of the machine does not contain traces with infinite postfix of outputs;

3. The machine has a special output $\delta \notin O$ that represents the quiescence [Tre96] at the states where the transitions under inputs are defined; at each state $s \in S_1$, there is a loop under δ, namely $(s, \delta, s) \in T_S$.

We return again to the example input/output automaton, consider a machine in Figure 2.3. The automaton ***S*** has five states, namely $S = \{s_1, \ldots, s_5\}$, where $S_1 = \{s_1, s_2, s_5\}$ and $S_2 = \{s_3, s_4\}$. At each state from the set S_1 the automaton accepts inputs i_1 and i_2. However, when the machine is at state s_3 or s_4 no inputs can be accepted and only outputs o_1 or o_2 can be produced. Note that this automaton precisely belongs to the considered class.

In order to define HS and SS for such a class of input/output automata, we introduce the following *experiment hypothesis*:
We assume that before applying any input, a tester (or any experimenting entity) waits for a given maximal output timeout t.

The experiment is performed as follows: the tester expects an output in t time units; if the machine produces one, then the timer is reset and the tester waits for another t time units. If no output is produced by the system in t time units then the tester applies the next input (if any) and resets the timer. In fact, this hypothesis is introduced to somehow avoid an infinite (indeed, "very-very long") waiting time before applying the next input (in FSMs and experiments with them, this never happens as each input we apply requires an immediate output reaction to be produced).

The latter explains the necessity of introducing the specific output $\delta \notin O$, namely whenever the output is not observed we assume that the system/machine produced the output δ. Such extension of the output alphabet allows to define the corresponding HSs and SSs for an Input/Output automaton. A sequence $\alpha = i_1 i_2 \ldots i_k$ is *synchronizing* for the automaton ***S*** if there exists a state $s \in S$ such that for each trace $\beta_1 i_1 \beta_2 i_2 \ldots \beta_k i_k \beta_{k+1}$ where p is the length of a longest sequence of consecutive outputs and $\beta_j \in (O \cup \{\delta\})^p$, $j = 1, \ldots, k+1$, it holds that the $\beta_1 i_1 \beta_2 i_2 \ldots \beta_k i_k \beta_{k+1}$-successor of the set S is either empty or equals $\{s\}$. A sequence $\alpha = i_1 i_2 \ldots i_k$ is *homing* for the automaton ***S*** if for each trace $\beta_1 i_1 \beta_2 i_2 \ldots \beta_k i_k \beta_{k+1}$, $\beta_j \in (O \cup \{\delta\})^p$, $j = 1, \ldots, k+1$, it holds that the $\beta_1 i_1 \beta_2 i_2 \ldots \beta_k i_k \beta_{k+1}$-successor of the set S is either empty or is a singleton. For example, for the automaton ***S*** in Figure 2.3, $\alpha = i_1 i_1$ is a homing sequence.

We propose to derive an SS for an automaton ***S*** where actions are divided into inputs

and outputs via an iterative elimination of the transitions labeled by outputs. Such transitions can always be omitted as for the automata class considered here, there does not exist a state where transitions under inputs and outputs are defined at the same time. In other words, we propose to derive an automaton where only the transitions under inputs are left and then use the SS derivation strategies for the automata without outputs. Therefore, we build an automaton $\boldsymbol{A} =< S_1, I, T_A >$ which at the beginning has an empty set of transitions, $T_A = \emptyset$. For each transition $(s, i, s') \in T_S$, where $s, s' \in S_1$, we iteratively add to T_A this transition (s, i, s'). At the same time, for each transition (s, i, s''), where $s \in S_1$ and $s'' \in S_2$, we add to T_A the transition (s, i, s''') where state $s''' \in S_1$ and s''' is in a β-successor of s'' in the automaton $\boldsymbol{S}$, $\beta \in O^*$. As a result, each synchronizing sequence α for $\boldsymbol{A}$ is a synchronizing sequence for $\boldsymbol{S}$.

The automaton $\boldsymbol{A}$ for the example input/output automaton $\boldsymbol{S}$ is shown in Figure 3.4.

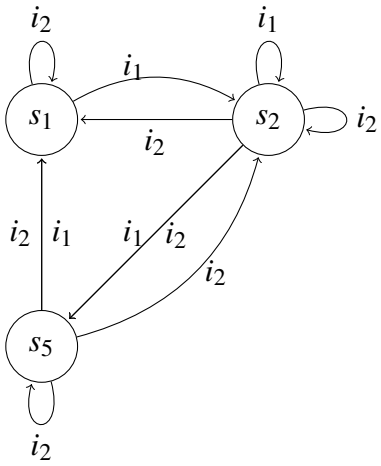

Figure 3.4: Automaton $\boldsymbol{A}$ for SS derivation for the input/output automaton $\boldsymbol{S}$

By direct inspection, one can check that the automaton $\boldsymbol{A}$ is not synchronizing and thus, there is no SS for $\boldsymbol{S}$.

In order to derive an HS for an input/output automaton $\boldsymbol{S}$, one can also reduce the problem to the HS derivation for well studied models, in particular for the HS derivation of a corresponding FSM. This FSM can be obtained via simulation and concatenation of potential output reactions at the relevant states. In fact, we propose to derive an FSM $\mathbf{M} =< S_1, I, O \cup O^2 \cup \cdots \cup O^p \cup \{\delta\}, T_M >$ which at the beginning has an empty set of transitions, i.e., $T_M = \emptyset$, where p is the length of a longest output trace of the automaton $\boldsymbol{S}$. Later, for each state $s \in S_1$, such that $(s, i, s') \in T_S$, $s' \in S_1$, we add to the T_M the transition (s, i, δ, s'). At the same time, for each state $s \in S_1$, such that $(s, i, s') \in T_S$, $s' \in S_2$, we add to the T_M the transition $(s, i, o_1 o_2 \ldots o_k, s'')$, $k \leq p$, where $s'' \in S_1$ is the $o_1 o_2 \ldots o_k$-successor

of state s'. Note, that similar to the SS existence check and derivation, a sequence α is homing for the automaton ***S*** if and only if α is a homing sequence for the FSM $\mathbf{M}$.

For the example input/output automaton ***S***, the FSM $\mathbf{M}$ is shown in Figure 3.5.

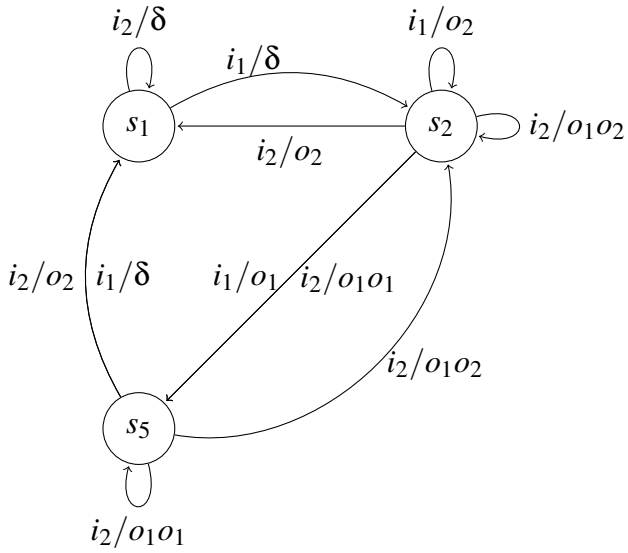

Figure 3.5: FSM $\mathbf{M}$ for the input/output automaton ***S***

One can check that the sequence $\alpha = i_1 i_1$ is homing for the FSM $\mathbf{M}$, and it is thus an HS for the automaton ***S*** in Figure 2.3 as well.

Therefore, the results obtained for classical FSMs and automata can be used for more generic models, especially, when the corresponding problem reduction, for instance, as shown above, can be found.

Using QBF solvers for HS existence check and derivation

As mentioned above, the satisfiability of Boolean formulas have been used, for example, for the existence check of an SS of a given automaton. In our case, together with the partners from the Russian Academy of Sciences and National Taiwan University, we proposed to apply Quantified Boolean formulas for the existence check and derivation of preset and adaptive HSs for nondeterministic FSMs. The results are presented in [WTJK17, TWJ+21], and in this manuscript we mostly show the preset strategy. More details, in particular, about the adaptive HS (for FSM state pairs) can be found in [TWJ+21].

We work with a quantified Boolean formula (QBF) Φ over the set of variables $X = X_1 \cup \cdots \cup X_k$, with $X_i \neq \emptyset$ and $X_i \cap X_j = \emptyset$ for $i \neq j$, expressed in its prenex form as

$Q_1X_1,\ldots,Q_kX_k.\phi$, where $Q_i \in \{\exists,\forall\}$ with $Q_i \neq Q_{i+1}$, and ϕ is a quantifier-free Boolean formula over variables X. Moreover, the matrix ϕ is in the conjunctive normal form. Given a non-initialized complete NFSM $\mathbf{S} =< S,I,O,h_S >$, we aim at finding a shortest preset homing sequence. We search from length 1 to the theoretical upper bound $2^{\binom{|S|}{2}}-1$ of a shortest homing sequence, for observable FSMs [KY13], and $2^{2^{|S|}}$ for non-observable FSMs. If a sequence of interest is not found, then the given FSM is not homing[7]. The QBF formulation for the bounded preset HS checking is as follows.

As the sets S, I, O are finite, we perform Boolean encoding on the states, input symbols, and output symbols with current-state variables $\vec{s}$, next-state variables $\vec{s'}$, input variables $\vec{x}$, and output variables $\vec{y}$. The transition relation h_S of the machine can be thus represented by the characteristic function $T(\vec{s},\vec{x},\vec{y},\vec{s'})$ in terms of the Boolean variables. Note that in the QBF formulation, we rely on time-frame expansion and denote the variables at the t^{th} time-frame with a superscript index t.

Then the QBF corresponding to the existence of a preset homing sequence of length n can be expressed as

$$\exists\vec{X},\forall\vec{Y},\forall\vec{S},\forall\vec{S^*}.[\Delta^{(n)}(\vec{X},\vec{Y},\vec{S}) \wedge \Delta^{(n)}(\vec{X},\vec{Y},\vec{S^*}) \rightarrow (\vec{s}^n = \vec{s^*}^n)],$$

where variables $\vec{X} = (\vec{x}^1,\ldots,\vec{x}^n)$, $\vec{Y} = (\vec{y}^1,\ldots,\vec{y}^n)$, $\vec{S} = (\vec{s}^0,\ldots,\vec{s}^n)$, $\vec{S^*} = (\vec{s^*}^0,\ldots,\vec{s^*}^n)$, and $\Delta^{(n)}$ is the conjunction of the transition relation of n time-frames, i.e., $\Delta^{(n)}(\vec{X},\vec{Y},\vec{S}) = \bigwedge_{k=1}^{n} T(\vec{s}^{k-1},\vec{x}^k,\vec{y}^k,\vec{s}^k)$ and $\Delta^{(n)}(\vec{X},\vec{Y},\vec{S^*}) = \bigwedge_{k=1}^{n} T(\vec{s^*}^{k-1},\vec{x}^k,\vec{y}^k,\vec{s^*}^k)$, where variables $\vec{s^*}$ are fresh copies of the variables $\vec{s}$.

By construction, this formula is true if and only if the underlying complete NFSM has a preset homing sequence of length n.

Similarly, the existence check of an adaptive homing strategy of height n where all the paths have the same length can be expressed as

$$\exists\vec{x}^1,\forall\vec{y}^1,\exists\vec{x}^2,\forall\vec{y}^2,\ldots,\exists\vec{x}^n,\forall\vec{y}^n,\forall\vec{S},\forall\vec{S^*}.((\Delta^{(n)}(\vec{X},\vec{Y},\vec{S}) \wedge \Delta^{(n)}(\vec{X},\vec{Y},\vec{S^*})) \rightarrow (\vec{s}^n = \vec{s^*}^n)),$$

where $\vec{X} = (\vec{x}^1,\ldots,\vec{x}^n)$, $\vec{Y} = (\vec{y}^1,\ldots,\vec{y}^n)$, $\vec{S} = (\vec{s}^0,\ldots,\vec{s}^n)$, $\vec{S^*} = (\vec{s^*}^0,\ldots,\vec{s^*}^n)$ are vectors of input variables, output variables, state variables, and fresh copies of state variables, respectively, and $\Delta^{(n)}$ is the conjunction of the transition relation of n time-frames, i.e., $\Delta^{(n)}(\vec{X},\vec{Y},\vec{S}) = \bigwedge_{k=1}^{n} T(\vec{s}^{k-1},\vec{x}^k,\vec{y}^k,\vec{s}^k)$ and $\Delta^{(n)}(\vec{X},\vec{Y},\vec{S^*}) = \bigwedge_{k=1}^{n} T(\vec{s^*}^{k-1},\vec{x}^k,\vec{y}^k,\vec{s^*}^k)$.

Note that the condition of the same length of the paths in this case is important, as

[7] Note however, that the non-existence of a preset HS does not imply the same result for the adaptive HS.

there exist non-observable machines where it cannot be held. We adjusted the formulas accordingly for a more general case and invite the Reader to refer to [TWJ+21], respectively. As in some cases, state pair homability is necessary and sufficient and in all cases, it is a necessary condition for the FSM to be (adaptively) homing, we also studied such particular pair-wise cases, and adjusted the QBF formulation accordingly. At the same time, pair-wise homability itself is discussed in more details ahead in the current chapter.

We note that the advantage of using solvers for the state identification or even other tasks, is in fact the existence of various solvers and their availability. In other words, once a necessary encoding is performed, as for example above, the (QBF) formula can be efficiently checked and the HS existence check can be verified. We should however note, that such utility not always leads to the most efficient solutions. In fact, in the mentioned works the interested Reader can find detailed experiments on the comparison of various QBF solvers, such as DepQBF [LB10], CAQE[RT15], among others. An interesting tendency was observed when comparing to a successor tree based approach implemented in FSMHSGen [Lop19]. The successor tree strategy is more efficient in terms of performance (time-wise) than the QBF solvers. However, for nondeterministic machines FSMHSGen runs out of memory faster than the QBF solvers, and therefore for some cases for example, where the HS does not exist, QBF solvers can go deeper in their search (up to a bigger length). Therefore, an interesting perspective that we plan, in fact, to study in the future, is a *hybrid* possibility - up to some level we descend in the successor tree and then run a solver for a weakly initialized corresponding FSM. This is not done yet and is left as further challenges.

Iterative approaches for adaptive final state identification sequences

In this section, we first discuss how the adaptive HS (or an AHS for short) existence for a complete observable nondeterministic FSM can be decided in polynomial time. The corresponding results were presented in [KY15b]; we omit the proofs as usual inviting the interested Reader to check the relevant publication.

We are interested in checking if a given FSM $\mathbf{S} =< S, I, O, h_S >$ is *adaptively homing*, i.e., if it possesses an AHS. An important property for almost all iterative approaches approaches in general is a possibility to reduce the AHS existence check task to the same one, but over a smaller state subset. The latter is possible, indeed, due to the following statement.

Proposition 2 *A complete observable non-initialized FSM* $\mathbf{S}$ *is adaptively homing if and only if each pair of two different states is adaptively homing.*

At the same time, for a given state pair, one can efficiently decide if it is homing or not, using the corresponding intersection of the related languages at these states. Let us consider two different states s_1 and s_2 of $\mathbf{S}$, without loss of generality. We derive the intersection $\mathbf{S}/s_1 \cap \mathbf{S}/s_2 = (Q,(s_1,s_2),I,O,h_{\mathbf{S}/s_1 \cap \mathbf{S}/s_2})$ in a usual way. States of $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$ are pairs (s_j,s_k), $j<k$, $j,k=1,\ldots,n$, and there is a transition $((s_j,s_k),i,o,(s'_j,s'_k)) \in h_{\mathbf{S}/s_1 \cap \mathbf{S}/s_2}$ if and only if $s'_j \neq s'_k$ and s'_j, s'_k are io-successors of states s_j and s_k. If for all o there are no such io-successors then a transition at the state (s_j,s_k) under input i is not defined.

Proposition 3 *Given two states* s_1 *and* s_2 *of a complete observable FSM* $\mathbf{S} =< S,I,O,h_S >$ *and the intersection* $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$*, states* s_1 *and* s_2 *are not adaptively homing if and only if the intersection* $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$ *has a complete submachine.*

Therefore, states s_1 and s_2 are adaptively homing if and only if the intersection $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$ has no complete submachine, i.e., each submachine has an input undefined in some state.

The existence of a complete submachine can be checked by iterative removal from the intersection $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$ each state that has an undefined input along with its incoming transitions. If at the end, the initial state is also removed then the two given states are adaptively homing, otherwise they are not adaptively homing. The procedure is polynomial with respect to the number of states [VYB+12] when the number of inputs and outputs are polynomial with respect to the number of states, and thus, the complexity of checking the existence of a complete submachine of $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$ is polynomial, since this machine has at most $n(n-1)/2$ states. The latter allows to conclude that the problem of checking the existence of an adaptive homing sequence for a complete observable FSM $\mathbf{S}$ is in P.

Note however, that we will further discuss that for a preset HS the problem is more complex. In fact, it can also be seen from the results related to the solvers' usability for the HS existence check. We will now discuss how an AHS can be constructed from those AHSs built for state pairs.

Indeed, state pairs $\{s_1,s_2\}$ and their related AHS can be iteratively utilized for the construction of the AHS for the machine $\mathbf{S}$. In this case, we propose to represent the AHS by a corresponding *test case* [PY05]. Given an input alphabet I and an output alphabet

O, a *test case* $\mathbf{TC}(I,O)$ is an initialized initially connected observable single-input output-complete FSM over input alphabet I and output alphabet O that has an acyclic transition graph. In other words, at each state either only one input with all possible outputs is defined or there are no outgoing transitions, and in the latter case, the state is a *deadlock* state. A test case is a partial FSM once $|I| > 1$. By definition, a test case $\mathbf{TC}(I,O)$ represents an adaptive experiment with a complete FSM $\mathbf{S}$ over alphabets I and O in the following way. If input i_1 is a defined input at the initial state t_0 of $\mathbf{TC}(I,O)$ then first, the input i_1 is applied to the FSM $\mathbf{S}$ under investigation and $\mathbf{TC}(I,O)$ moves to the i_1o-successor t_1 of state t_0 if $\mathbf{S}$ produces the output o as the response to the input i_1. The next input to apply is the input defined at state t_1, etc. The procedure terminates when a deadlock state is reached. The *height* of the test case $\mathbf{TC}(I,O)$ is the length of a longest trace from the initial state to a deadlock state of $\mathbf{TC}(I,O)$ and it specifies the length of a longest input sequence defined in the test case that can be applied to the FSM $\mathbf{S}$ during the adaptive experiment.

Given FSM $\mathbf{S} =< S,I,O,h_S >$, a test case $\mathbf{TC}(I,O)$ is a *homing* test case (HTC) for $\mathbf{S}$ if for every trace γ from the initial state to a deadlock state, the γ-successor of the set S in $\mathbf{S}$ is a singleton or the empty set. For the example FSM $\mathbf{S}$ in Figure 2.2, a HTC is shown in Figure 3.6.

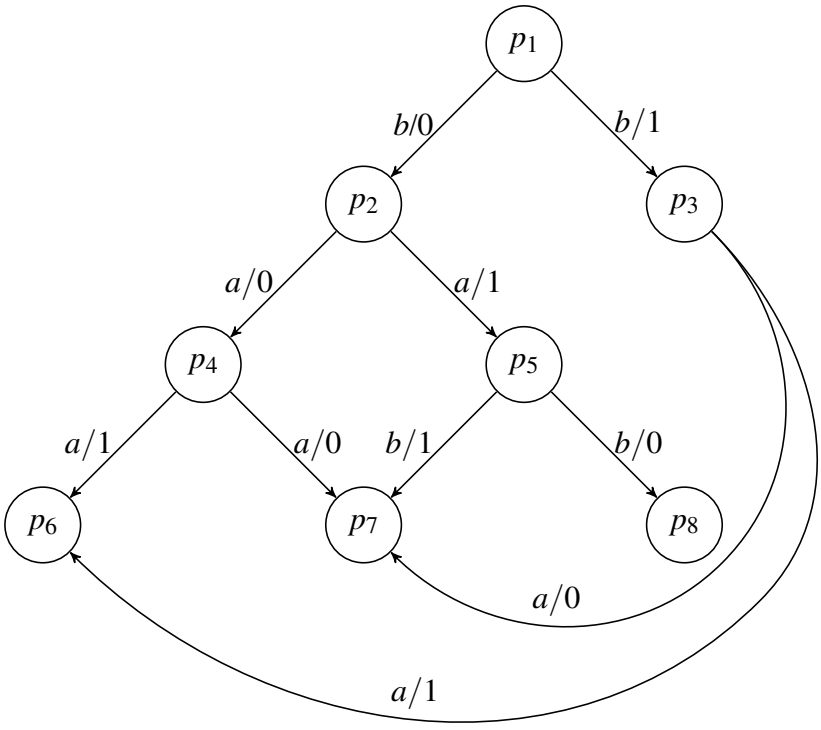

Figure 3.6: **HTC** for FSM $\mathbf{S}$ in Figure 2.2

The iterative approach for building an HTC starts with a state pair, consider a homing test case $\mathbf{HTC}_{\{s_1,s_2\}} = \mathbf{P}_{1,2}$ for the set $\{s_1, s_2\}$. We derive a test case $\mathbf{P}_{1,2,3}$ by adding the state s_3 into the set labeling the initial state of $\mathbf{P}_{1,2}$ and obtain $\mathbf{P}_{1,2,3}$ that includes all the transitions of $\mathbf{P}_{1,2}$. Subsets of states that label deadlock states of the intersection $\mathbf{S} \cap \mathbf{P}_{1,2,3}$ are not necessary singletons but they contain at most two states, since the specification FSM is observable. Each pair of different states of $\mathbf{S}$ is homing, thus, for each deadlock state $(\{s_i, s_j\}, p)$ of $\mathbf{S} \cap \mathbf{P}_{1,2,3}$ the initial test case $\mathbf{P}_{1,2,3}$ is concatenated with the corresponding test case $\mathbf{P}_{i,j}$. Proceeding in the same way the test case $\mathbf{P}_{1,2,\ldots,n}$ is derived. By construction, the test case $\mathbf{P}_{1,2,\ldots,n}$ is a homing test case for the FSM $\mathbf{S}$. A schematic representation of this strategy is given in Figure 3.7.

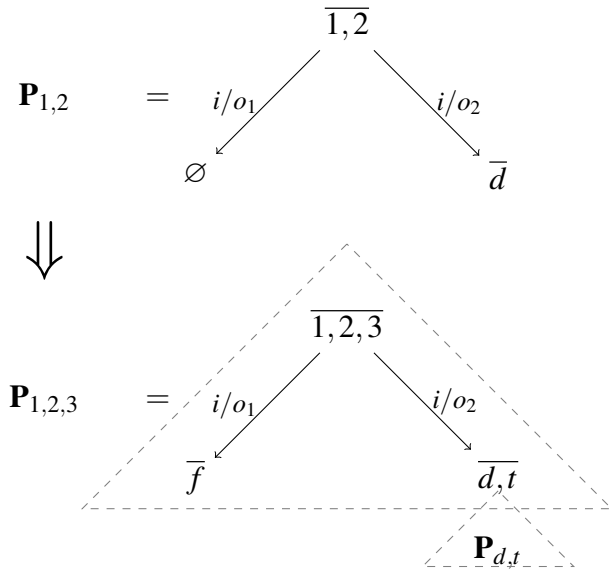

Figure 3.7: $\mathbf{P}_{1,2,3}$ derivation scheme

Note that this result was first discussed in [KEYC16a]. Later, with the colleagues from ISP RAS, we strengthened the aforementioned result. In particular, we estimated the number of states in the corresponding test case [YKK19]. In fact, we have shown that for the FSM $\mathbf{S}$ with n states and a homing pair $\{s_1, s_2\}$ of $\mathbf{S}$, there exists a homing test case $\mathbf{HTC}_{\{s_1,s_2\}}$ such that the set of states of $\mathbf{HTC}_{\{s_1,s_2\}}$ is the union of the set of all pairs of different states of FSM $\mathbf{S}$, the set of singletons of S and the deadlock state D. Moreover, given a j-homing state[8] q_1 and an m-homing state q_2, if $j \leq m$, then state q_2 is unreachable from state q_1 in $\mathbf{HTC}_{\{s_1,s_2\}}$. The number of transitions of the test case $\mathbf{HTC}_{\{s_1,s_2\}}$ therefore does not exceed $|O|n(n-1)/2$. We also proposed an algorithm that is based on an iterative enumeration of state pairs, for deriving a homing test case $\mathbf{HTC}$

[8]The subset is a j-homing set if it is a $(j-1)$-homing, or there exists an input $i \in I$, such that for each $o \in O$, its io-successor is either empty or is a $(j-1)$-homing set.

for the FSM $\mathbf{S}$ with at most $|O|(n-1)^2n/2$ transitions. For that matter we make use of two arrays: $IOsuc$ and $Input$. The first is a two dimensional array; columns of $IOsuc$ correspond to states of the given FSM while the rows correspond to possible io pairs, i.e., given a state and an io pair, the related cell has either the corresponding io-successor or it is empty (the transition is not defined). In fact, this is a representation of the specification machine itself. While the array $Input$, at the same time, stores for each j-homing $\{s_a, s_b\}$ pair the corresponding input $i_{\{s_a,s_b\}}$ that brings $\{s_a, s_b\}$ to a $(j-1)$-homing set. Such additional arrays allow the derivation of the homing test case $\mathbf{HTC}$ in polynomial time. In fact, the following result has been established.

Theorem 1 *Given a homing FSM* $\mathbf{S} =< S, I, O, h_S >$, $|S| = n$, *and the maximum integer* k *such that FSM* $\mathbf{S}$ *has a pair of different states that is* k*-homing, there exists a homing test case of the height at most* $(n-1)k$ *with at most* $(n-1)^2n/2+n+1$ *states, at most* $|O|(n-1)^2n/2$ *transitions; the (time) complexity of deriving this test case is* $\mathrm{O}(|O|n^5)$ *when* $\mathbf{S}$ *is represented by the array* $IOsuc$ *and an array* $Input$ *is already derived.*

The interested Reader may find the details, including the proofs in [YKK21].

Note that adaptive homing sequences, derived iteratively (or otherwise), can be effectively prolonged to adaptive synchronizing sequences, that we similarly represent via synchronizing test cases. As in the case of preset final state identification sequences, a synchronizing test case is a homing one with additional constraints on the singletons. In particular, a homing test case $\mathbf{TC}(I,O)$ is a *synchronizing* test case ($\mathbf{STC}$) for the FSM $\mathbf{S}$, if there exists a state s such that for every trace γ of $\mathbf{TC}(I,O)$ from the initial to a deadlock state, γ-successor of S is either $\{s\}$ or the empty set.

The existence check of synchronizing test cases is reduced to that one of homing, with additional conditions of so called *definitely-reachable* states (or simply *d-reachable*). State $s' \in S$ is *d-reachable* from state $s \in S$ if there exists a test case $\mathbf{P}(s,s')$ over alphabets I and O such that for every trace γ of $\mathbf{P}(s,s')$ from the initial state to a deadlock state, the γ-successor of state s in FSM $\mathbf{S}$ is either the empty set or is the set $\{s'\}$. We refer to such a test case as a *d-transfer* test case. In [PY11], necessary and sufficient conditions are established that allow to check if a state $s \in S$ is definitely reachable from the initial state s_0 of the initialized FSM $\mathbf{S}$. In [KYY16], together with the colleagues from Tomsk State University and Sabanci University, we adjusted this procedure for arbitrary states s and s'. In fact, the main result in [KYY16] is probably the following.

Proposition 4 *There exists a synchronizing test case for a complete observable FSM*

$\mathbf{S} =< S, I, O, h_S >$, *if and only if FSM* $\mathbf{S}$ *is homing and there exists a state* $s' \in S$ *such that for each state* $s \in S$ *state* s' *is definitely reachable from* s.

This statement gives a constructive approach for deriving an $\mathbf{STC}$. Consider a trace γ that takes $\mathbf{HTC}$ from the initial state to a deadlock state. The $\gamma - successor$ of each state of the set S either does not exist or contains a unique state s. Since for any state $s \in S$, there exists a d-transfer test case $\mathbf{P}(s, s')$, then each trace δ that takes $\mathbf{P}(s, s')$ from the initial state to a deadlock state takes the FSM $\mathbf{S}$ from state s to state s', i.e., the $\gamma\delta$-successor of each state of the set S either does not exist or contains a unique state s'. In other words, $\mathbf{HTC}$ is thus prolonged up to $\mathbf{STC}$ via appending the relevant d-transfer test cases.

As an example, consider an FSM $\mathbf{S}$ in Fig 3.8. Note that state 3 is d-reachable state for this FSM (from state 1 via input b and from state 2 via input a).

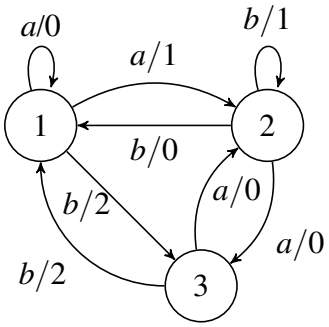

Figure 3.8: A complete observable nondeterministic FSM $\mathbf{S}$ where state 3 is d-reachable from states 1 and 2

The synchronizing test case for $\mathbf{S}$ in Figure 3.8 is shown in Figure 3.9. Note again, that differently from preset sequences, the adaptive synchronizing experiment can be longer or shorter depending on the output reactions of the machine under experiment. For example, for the FSM in Figure 3.8, if the machine replies 0 to the input b, the experiment can be over after the next b (same if it replies 1 in fact at the first step). However, if the machine replies 2 as the output reaction to the first input, then the synchronizing experiment can only be finished in 3 steps.

Iterative approach for adaptive distinguishing sequences for merging-free FSMs

The results presented above can be extended for distinguishing sequences, however only for a very specific class of FSMs. Similar to AHS, adaptive distinguishing sequences can be also represented by a test case and an iterative approach for building such a test case

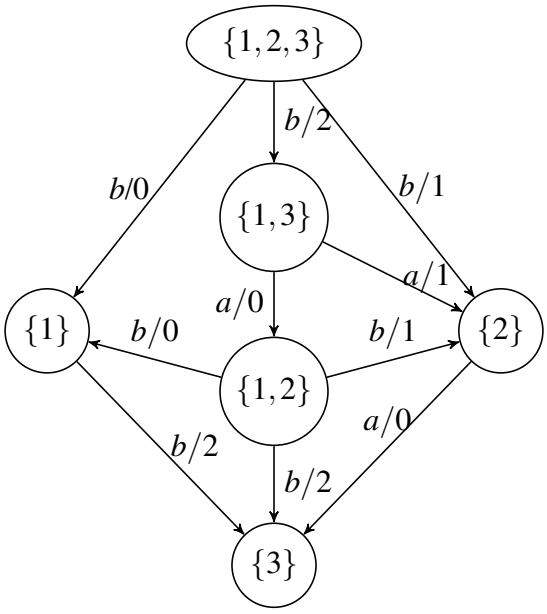

Figure 3.9: $\mathbf{STC}$ for FSM $\mathbf{S}$ in Figure 3.8

can be applied. A test case $\mathbf{TC}(I,O)$ is a *distinguishing* test case (DTC) for an FSM $\mathbf{S} =< S,I,O,h_S >$, if every trace γ from the initial state to a deadlock state can be a trace at most at a single state of the set S.

Generally, the generation of DTCs can be performed by an iterative construction of all 1-distinguishing, 2-distinguishing, ..., k-distinguishing[9] subsets of states; until the set S is obtained. However, this strategy can return a test case of an exponential height (see the next section for the estimation of the DTC length/height).

As mentioned above, the iterative approach based on the test cases for state pairs cannot be directly applied to an arbitrary FSM. Nevertheless there exists a special class of nondeterministic FSMs, for which the problem of checking whether an FSM is adaptively distinguishing (ADS exists) can be performed in a polynomial time, and the length of a shortest distinguishing case (if it exists) is polynomial with respect to the number of FSM states. Such a class of nondeterministic FSMs was presented in [YK15].

Given a complete observable FSM $\mathbf{S}$, we assume that for each input i and each two different states of the FSM the non-empty io-successors of these states do not coincide for any output o. We further refer to such FSM as a *merging-free* FSM. By definition, for a complete observable merging-free FSM $\mathbf{S}$, for any sequence $\gamma \in (IO)^*$, the non-empty

[9] A subset of states is k-distinguishing if it is $(k-1)$-distinguishing, or there exists an input $i \in I$, such that for each $o \in O$, its io-successor is either empty or is a $(k-1)$-distinguishing set, and in addition, the io-successors of two different states of it do not coincide.

γ-successors of two different states of S do not coincide. Such merging-free machines are of interest in our case, as almost all the results presented for AHSs of complete observable FSMs are valid for such a class.

Proposition 5 *Given two states s_1 and s_2 of a complete observable merging-free FSM $\mathbf{S} =< S, I, O, h_S >$ and the intersection $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$, states s_1 and s_2 are adaptively distinguishing if and only if the intersection $\mathbf{S}/s_1 \cap \mathbf{S}/s_2$ has no complete submachine, i.e., each submachine has an input undefined in some state.*

Again, similar to the AHS, a complete observable merging-free FSM $\mathbf{S}$ is adaptively distinguishing if and only if each pair of its states is adaptively distinguishing. The procedure for building the $\mathbf{DTC}_{\{s_1,s_2,\ldots,s_n\}}$ repeats the one for the AHS in this case. We start with the $\mathbf{DTC}_{\{s_1,s_2\}}$ and then add state s_3 to the initial state and therefore construct $\mathbf{DTC}_{\{s_1,s_2,s_3\}}$ appending the necessary test cases in the relevant deadlock states of $\mathbf{DTC}_{\{s_1,s_2\}}$. The procedure repeats iteratively until the last state s_n is added and the $\mathbf{DTC}$ for $\mathbf{S}$ is derived.

Therefore, if a complete observable merging-free FSM $\mathbf{S} =< S, I, O, h_S >$, $|S| = n$, is adaptively distinguishing then the length of a shortest adaptive distinguishing sequence is of the order n^3. Note that in the manuscript, we omit examples of such merging-free FSMs and the corresponding DTCs, however the interested Reader can for example check the publication [YK15], where more details are provided.

Due to a room of the manuscript, some techniques for deriving (adaptive) state identification sequences for nondeterminstic FSMs have been omitted. For example, we briefly mentioned above k-homing and k-distinguishing state subsets and the related iterative approaches for building HTCs and DTCs. We proposed such approaches in collaboration with the researchers from the Russian Academy of Sciences / Tomsk State University, American University of Sharjah, and Télécom SudParis. The algorithms and proofs can be found in [KEYC16b]. We also note that recently we extended the results over S'-homing and S'-synchronizing test cases, when for example, the set of related singletons to finish the experiment, is limited by a subset S' of FSM states. The latter was done in collaboration with researchers from the Russian Academy of Sciences and the related results are published in [YKK21].

3.3.2 Evaluating the length of state identification sequences and the complexity of related problems

This section is devoted to a brief summary of the original results obtained in the area of the complexity estimation for the state identification of nondeterministic FSMs. In particular, we are interested in the reachability of the worst cases when it comes to the length of the related sequences, and for some preset and adaptive state identification sequences we managed to prove such reachability for specific classes of nondeterministic FSMs.

Preset homing sequences and relevant complexity

We start with the final state identification and in particular, with the HSs for that matter. First of all, we note that differently from deterministic FSMs, not every complete nondeterministic observable reduced FSM has a homing sequence. Moreover, as shown in the PhD thesis of the author, according to **Rules 1-3** for the truncating of the successor tree, the length of a shortest homing sequence for a complete observable FSM $\mathbf{S} =< S, I, O, h_S, S_{in} >$, $|S| = n$, $|S'| = m, m \leq n$, is limited by the value $2^{\binom{n}{2}} - 2^{\binom{n}{2}-\binom{m}{2}}$. Note that this upper bound concerns a potentially weakly initialized FSM. If the machine $\mathbf{S}$ is non-initialized, then it gives $2^{\binom{n}{2}} - 1$, accordingly. Note also that in [SEY07], the reachability of the upper bound $2^{n^2/4}$ of the length of a distinguishing (in fact, separating, for nondeterministic FSMs[10]) sequence has been proven. For an observable FSM, each separating sequence is also a homing sequence, but the question of reachability for the HS is still interesting. This question was in fact studied in the PhD of the author and the results were published [KY13]. We thus do not include these results in the current manuscript, but we will only mention that there exists a class of complete observable nondeteministic machines with $n > 3$ states and $(n-1)$ inputs, such that a shortest HS for each FSM of this class has length of $2^{n-1} - 1$. Therefore, the exponential upper bound for the HS length is reachable, and we are currently working on reducing this gap between $2^{n-1} - 1$ and $2^{\binom{n}{2}} - 1$, to be able to say more about the reachability.

The exponential upper bound on the length of an HS for nondeterministic machines automatically implies that the corresponding decision problem of the HS existence check cannot be in the class NP. More precisely, the corresponding complexity was estimated in [KKE14]. In order to prove the PSPACE-completeness of the related decision problem, one can return to the truncated successor tree and formulate first a simpler problem (can

[10] The output reactions on this sequence do not intersect.

be formulated for non-initialized or even weakly initialized machines).

α-HOMING
Input: a complete observable nondeterministic FSM $\mathbf{S} =< S, I, O, h_S, S' >$, $|S| = n$, $|S'| = m, m \geq 2$, an input sequence α.
Problem: Is α a homing sequence for the FSM $\mathbf{S}$?

In fact, α is a homing sequence for $\mathbf{S}$ if and only if the terminal node in the α-branch is labeled with the empty set or with the set P that has only singletons (see the corresponding successor tree). The latter automatically proves that α-HOMING $\in$ PSPACE, as when deriving a branch of the tree labeled α, we only store the set P of state pairs reached after j steps. Given an input i_{j+1}, we update P by the set of all $i_{j+1}o$-successors of pairs of P for all outputs $o \in O$. Correspondingly, for an observable FSM the cardinality of the updated set is at most $n(n-1)/2+n$. Therefore, at each step the internal memory of polynomial size (with respect to the number of states of FSM $\mathbf{S}$) is utilized.

HOMING problem (existence check of an HS for a nondeterministic FSM) complexity is estimated in the same way with just the only difference that the next input to test is not the next input of a sequence α but an input which is nondeterministically chosen. If after at most $2^{\binom{n}{2}} - 2^{\binom{n}{2}-\binom{m}{2}}$ steps a terminal node is not reached using **Rules 1** or **3** then there is no HS for a given FSM. Otherwise, a nondeterministically generated sequence that leads to a terminal node due to **Rules 1** or **3**, is the HS for FSM $\mathbf{S}$. Therefore, HOMING $\in$ NPSPACE and due to Savitch's theorem [Sav70] it is in PSPACE. Note that even for deterministic machines that are not reduced, this HOMING problem is PSPACE-complete [San05], therefore it remains the same for the case of nondeterministic FSMs. The same results hold for preset and adaptive homing sequences for partial deterministic weakly initialized and non-initialized FSMs; these classes of FSMs we studied (with the colleagues from Sabanci University and Tomsk State University) in [YYK17], accordingly.

We again draw the Reader's attention to the fact that the exponential upper bounds on the length of SS and DS for nondeterministic automata and FSMs have been previously established [IS04, SEY07]. We omit any details concerning these results as they do not take part in the author's contributions. Instead, we move to the adaptive distinguishing sequences, where some original results (with the colleagues from American University of Sharjah and Tomsk State University) concerning the height of the corresponding experiment have been obtained.

Evaluating the height (length) of a DTC

We first of all remind that for deterministic machines, adaptive DSs help a lot in terms of complexity reduction. The height of the corresponding experiment becomes quadratic w.r.t. the number of FSM states [LY94]. Note also that if we used an incremental approach for building a DTC for a nondeterministic FSM, starting from 1-distinguishing, 2-distinguishing, etc. subsets of states, until reaching the set S which is k-distinguishing for a certain k, the complexity in terms of the number of these iterations is already exponential. Indeed, what if we have to enumerate all the state subsets before coming the set S? The latter gives the upper bound on the height of the experiment as $\sum_{i=2}^{m} \binom{n}{i}$, where m is the number of initial states for a weakly initialized machine. In [EYK18], we prove the reachability of this upper bound for a non-initialized machine. Indeed, we present a class of nondeterministic FSMs with n states such that the set S is $k = (2^n - n - 1)$-distinguishing and not $(2^n - n - 2)$-distinguishing, i.e., a shortest DTC has precisely the height of $(2^n - n - 1)$. As usual, we omit most of the details and related proofs, presenting mostly the intuition about how such a machine is constructed.

Similar to [KY13],[HYC12], we introduce a special linear order over the set of all non-empty subsets of the state set S that are not singletons. In fact, we construct an appropriate sequence (chain) of such subsets. The number of all such subsets is $(2^n - n - 1)$. Given such a sequence of subsets of states, we show that the tail subset with two states of the sequence is the only subset that is 1-distinguishing. Moreover, for each other subset g, it holds that if the following subset in the sequence is t-distinguishing, $t > 0$, then the set g is the only $(t+1)$-distinguishing subset of the given FSM.

Given the set $S = \{1, \ldots, n\}$ (without loss of generality) and $0 < k \leq n$, let $C^{\vdash}(n,k)$ denote the sequence of all subsets of the set S of the cardinality k headed by the subset $\{n-k+1, \ldots, n\}$ and tailed by the subset $\{1, \ldots, k\}$, and $C^{\dashv}(n,k)$ denote the sequence of subsets $\{1, \ldots, k\}$, $\ldots$, $\{n-k+1, \ldots, n\}$ where the subsets of $C^{\vdash}(n,k)$ are written in the reverse order. Thus, $C^{\vdash}(n,1)$ is a sequence of singletons $\{n\}, \ldots, \{2\}, \{1\}$ while $C^{\dashv}(n,1) = \{1\}, \{2\}, \ldots, \{n\}$. $C^{\vdash}(n,n) = C^{\dashv}(n,n) = \{1, \ldots, n\}$. We assume that the sequence $C_x^{\vdash}(m,r)$ (or $C_x^{\dashv}(m,r)$) is obtained by adding the item x to each subset of the sequence $C^{\vdash}(m,r)$ (or $C^{\dashv}(m,r)$). As usual, we use $C.C'$ to denote the concatenation of the two sequences C and C'.

Given integers n and k, $n > 1$, $1 \leq k \leq n$, and sequences $C^{\vdash}(m,r)$ for all $k \leq m < n$ and $0 < r < k$, we use the following formulas to derive $C^{\vdash}(n,k)$. $C^{\vdash}(n,k)$ = $C_n^{\vdash}(n-1, k-1).C_{n-1}^{\dashv}(n-2, k-1) \ldots C_k \dashv (k-1, k-1)$, if $(n-k)$ is odd. Otherwise, if $(n-k)$ is

even $C^{\vdash}(n,k) = C^{\vdash}_n(n-1,k-1).C_{n-1} \dashv (n-2,k-1)\ldots.C_k \vdash (k-1,k-1)$. Note that by induction, the sequence $C^{\vdash}(n,k)$, $1 \leq k \leq n$, obtained this way has exactly once each subset of the set $\{1,\ldots,n\}$ of the cardinality k. Similarly, given sequences $C^{\vdash}(n,k)$ and $C^{\dashv}(n,k)$, $0 < k \leq n$, a sequence C_n is derived as follows. $C_n = C^{\vdash}(n,n).C^{\vdash}(n,n-1).C^{\dashv}(n,n-2)\ldots.C^{\dashv}(n,2)$, if n is even. Otherwise, if n is odd, $C_n = C^{\vdash}(n,n).C^{\vdash}(n,n-1).C^{\dashv}(n,n-2)\ldots.C^{\vdash}(n,2)$. By construction, the sequence C_n contains each non-empty subset of the set $\{1,\ldots,n\}$ of the cardinality $k > 1$ exactly once. Moreover, for each two consecutive subsets p_j and p_{j+1} of C_n two cases are possible: (i) if $|p_{j+1}| < |p_j|$ then $p_{j+1} \backslash p_j$ is a singleton; (ii) if $|p_{j+1}| = |p_j|$ then the sets $p_j \backslash p_{j+1}$ and $p_{j+1} \backslash p_j$ are singletons. Therefore, C_n defines a linear order $\succ_n$ over all nonempty subsets of states of the set $\{1,\ldots,n\}$ which are not singletons. For example, for $n = 5$, C_n defines a sequence $\{1,2,3,4,5\} \succ_5 \{2,3,4,5\} \succ_5 \{1,3,4,5\} \succ_5 \{1,2,4,5\} \succ_5 \{1,2,3,5\} \succ_5 \{1,2,3,4\} \succ_5 \{1,2,3\} \succ_5 \{2,3,4\} \succ_5 \{1,3,4\} \succ_5 \{1,2,4\} \succ_5 \{1,2,5\} \succ_5 \{2,3,5\} \succ_5 \{1,3,5\} \succ_5 \{1,4,5\} \succ_5 \{2,4,5\} \succ_5 \{3,4,5\} \succ_5 \{4,5\} \succ_5 \{3,5\} \succ_5 \{2,5\} \succ_5 \{1,5\} \succ_5 \{1,4\} \succ_5 \{2,4\} \succ_5 \{3,4\} \succ_5 \{2,3\} \succ_5 \{1,3\} \succ_5 \{1,2\}$.

Given the order $\succ_n$, we derive a class of observable nondeterministic FSMs (an FSM $\mathbf{M}_n$ with n states), such that a shortest adaptive test case is of the height $2^n - n - 1$. The derived FSM has states $1,\ldots,n$ ($n > 2$), $L = 2^n - n - 1$ inputs, and $\binom{n}{2} + n + 1$ outputs. For each i, $0 < i < L$, consider consecutive subsets of C_n, $p = p_i$ and $p' = p_{i+1}$ in order to derive the FSM transitions under the input $i_{p/p'}$ which is used for providing the corresponding $(L-(i-1))$-distinguishability of the set p. The L-th input, named i_{dist}, has to distinguish only states of the tail subset of C_n. The outputs $0, o_1, o_2, \ldots, o_n$ define io-successors for providing the corresponding distinguishability of the set p while the output $o_{(a,b)}$, $0 < a < b \leq n$, is used in order to show that states a and b have the same $io_{(a,b)}$-successor under the corresponding input i. Transitions are derived using the two cases, *Case-1* that applies when $|p| > |p'|$ and *Case-2* applied when $|p| = |p'|$.

Case-1: $|p| > 2$, and $p' = p\backslash\{r\}$, $r \in p$. For every state j, $j \in p'$, there are transitions: $(j, i_{p/p'}, 0, j)$ and $(j, i_{p/p'}, o_t, j)$ for every $t \in p\backslash\{j,r\}$. For state $r \in p$, there are transitions: $(r, i_{p/p'}, o_t, r)$ for every $t \in p\backslash\{r\}$. For every state j, $j \notin p$, there are the transitions: $(j, i_{p/p'}, 0, j)$ and $(j, i_{p/p'}, o_t, r)$ for some $t \notin p\backslash\{1,r\}$.

Proposition 6 *Given subsets p of cardinality $k > 2$ and $p' = p\backslash\{r\}$, $r \in p$, each $i_{p/p'}o$-successor of p, $o \neq 0$, has the cardinality less than k. Moreover, the $i_{p/p'}0$-successor of p is p' and for any other output o, the $i_{p/p'}o$-successors (if they exist) of any two different states of p do not coincide.*

Proposition 7 *Given subsets p of cardinality $k > 2$, $p' = p\backslash\{r\}$, $r \in p$, and a subset q of the set $\{1,\dots,n\}$, $|q| > 1$ and $q \neq p$, there either exists an $i_{p/p'}o$-successor of q that is equal to q or there exist two different states of the subset q that have the same non-empty $i_{p/p'}o$-successors for some o.*

The latter assures that when considering a subset p an input $i_{p/p'}$ should be applied to get to the subset p' in the corresponding DTC. Moreover, this input $i_{p/p'}$ is "useless" for any other subset q.

Case-2: $\exists j \in p, r \notin p$, such that $p' = p\backslash\{j\} \cup \{r\}$. For every state $s \notin p\backslash\{j\}$ there are transitions $(s, i_{p/p'}, 0, r)$. For every state $s \in p\backslash\{j\}$, there are transitions $(s, i_{p/p'}, 0, s)$. For every state $s \notin p \cup \{r\}$, and state $t \in p$ there are transitions $(s, i_{p/p'}, o_{(s,t)}, s)$ and $(t, i_{p/p'}, o_{(s,t)}, s)$. For every state $b \in p$, there are transitions $(b, i_{p/p'}, o_c, b)$ for each $c \in p\backslash\{b\}$.

Proposition 8 *The $i_{p/p'}0$-successor of p is p' and $\forall o \neq 0$ the cardinality of the $i_{p/p'}o$-successor of p is less than $|p|$.*
Given any subset $q \neq p$, $|q| > 1$, and $i_{p/p'}$, either for some output o there exist two equal non-empty $i_{p/p'}o$-successors of two different states of q or for some output o, the $i_{p/p'}o$-successor of q is q.

Similar to the propositions above, the latter states that the input $i_{p/p'}$ is 'helpful' only for distinguishing states of the subset p'.

Finally, for the linear order relation $\succ_n$ and the corresponding chain C_n with the tail subset $\{j,k\}$, $j < k$, we add a proper input i_{dist} that distinguishes states j and k, i.e., the set $\{j,k\}$ is 1-distinguishing while for any other pair of states there exists an output o such that the non-empty io-successors of the states of the pair coincide. For example, this can be done as proposed in Table 3.3: $\forall s \notin \{j,k\}$ there are transitions $(s, i_{dist}, 1, 0)$ and $(s, i_{dist}, 1, o_1)$; at state j there is a transition $(j, i_{dist}, 1, 0)$ while at state k there is a transition $(k, i_{dist}, 1, o_1)$.

Table 3.3: $\mathbf{M}_n$ transitions over the input i_{dist}

State	1	...	$j-1$	j	$j+1$	...	$k-1$	k	$k+1$	...	n
Input i_{dist}	$1/0, o_1$	...	$1/0, o_1$	$1/0$	$1/0, o_1$	...	$1/0, o_1$	$1/0_1$	$1/0, o_1$	...	$1/0, o_1$

By construction, the derived FSM $\mathbf{M}_n$ has $O(n^3 2^n)$ transitions and for this FSM, the following property holds.

Theorem 2 *The FSM* $\mathbf{M}_n$ *has a shortest adaptive test case of length* $2^n - n - 1$.

Therefore, DTC can also reach exponential length for nondeterministic specifications and therefore, test generation with the guaranteed fault coverage against such specifications can be somewhat unpractical. That is the reason, why another area of research of the author, covers the possibilities for the complexity reduction, on the one hand, preserving the fault coverage or the necessary properties for example, for state identification, and on the other, reducing the length of the related sequences or searching for the sequence existence check problems that are in P. Note that the second (habilitation, Doctor of Sciences) thesis of the author, presented in Russia in 2016 was exactly devoted to decreasing the complexity of 'gedanken' experiments with nondeterministic FSMs [Kus16]. Below, we discuss just some examples when such complexity can be decreased. Note also, that some of the results presented below are rather recent and have been obtained after the habilitation [Kus16].

3.4 Making it more practical – possibilities to reduce the complexity, discussing particular cases

When talking about particular cases of nondeterministic FSMs, we can first note a couple of 'good' classes[11] already discussed above.

- *Complete observable non-initialized FSMs for the final state identification*: indeed, as previously established, for these FSMs preset HSs and SSs might not be so promising, however the existence of the adaptive HS and SS can be done in polynomial time and the height of the corresponding test case is polynomial (w.r.t. the number of FSM states).

- *Complete observable non-initialized merging-free FSMs for the initial state identification*: similar to the case above, this class is promising as the existence check of an adaptive DS can be done in polynomial time, and the corresponding test case has a polynomial height as well.

[11] In our case, 'good' means an 'easy' (solved in polynomial time) existence check and/or derivation problem of the corresponding state identification or testing task.

3.4.1 Nondeterministic FSMs with 'good' projections

Given a specification FSM $\mathbf{S}$, it can be weakly or non-initialized, depending on the test purpose and testing methodology. One needs to perform the test generation against this specification in an efficient way. We first start with an efficient related state identification for $\mathbf{S}$, which can be nondeterministic and maybe even non-observable. Solving this task right away can be complicated, as also discussed above, however maybe this specification machine $\mathbf{S}$ allows a 'good' projection and then the state identification can be rather simple. A *projection* is a submachine of the specification $\mathbf{S}$ that is obtained via deleting some transitions[12]. Later on, we check if the projection belongs to a 'good' FSM class, for example, it is deterministic complete and reduced, or it is in one of the classes mentioned in the items above, and if this is the case then a corresponding experiment can be derived efficiently for the state identification. The preset/adaptive state identification sequence for the projection is also that one for the initial specification FSM. In this case, not only polynomial length of the corresponding state identification sequence is the purpose but also a fast construction of such a projection.

It is known that $n(n-1)/2$ is the tight upper bound for the height of an ADS for deterministic FSMs [LY94], therefore a deterministic projection is of interest. We call a transition $(s,i,o,s') \in h_S$ *deterministic* if for any transition $(s,i,o',s'') \in h_S$ we have $o' = o$ and $s'' = s'$. For a given FSM $\mathbf{S} =< S,I,O,h_S >$, we define the *deterministic projection* $\mathbf{S}^d =< S,I^d,O^d,h_S^d >$ of $\mathbf{S}$ as follows. S^d and S have the same set S of states. For a transition $(s,i,o,s') \in h_S$, $(s,i,o,s') \in h_S^d$, if and only if (s,i,o,s') is a deterministic transition in $\mathbf{S}$. I^d and O^d consist of the inputs and outputs used in the transitions in h_S^d. Intuitively, $\mathbf{S}^d$ is the same FSM as $\mathbf{S}$ where the deterministic transitions are preserved but all other transitions are removed. $\mathbf{S}^d$ is thus a deterministic FSM by definition. Moreover, it is easy to see that an adaptive DS for $\mathbf{S}^d$ or DTC can be directly used as a DTC for $\mathbf{S}$ as well. In the lucky case that $\mathbf{S}^d$ is a completely specified deterministic FSM, the existence check and the adaptive DS construction algorithms given in [LY94] can directly be applied. Therefore, a nondeterministic FSM can still be 'good' if it allows a deterministic projection and this projection possesses a DTC. Note that the deterministic projection is unique and can be computed in a linear time w.r.t. the number of transitions of the specification FSM.

Similarly, in an FSM $\mathbf{S}$, a transition from state s under input i is *observable* if for any two transitions (s,i,o,s'), $(s,i,o',s'') \in h_S$ we have $(s'' \neq s' \implies o' \neq o)$. We define the *observable projection* $\mathbf{S}^o =< S,I^o,O^o,h_S^o >$ of $\mathbf{S}$ that contains all observable transitions

[12]Note that the related results on the projections have been obtained together with the colleagues from Tomsk State University, Sabanci University and published in [YYK16, KY15a].

and only them. Note that the derivation of the FSM $\mathbf{S}^o$ requires not more than $|h_S||S||I||O|$ steps. If we manage to get not only observable but also complete and merging-free projection, then the DTC existence check and derivation becomes simpler. Any DTC, which will have a polynomial height in this case, of the projection $\mathbf{S}^o$ will also be the one of the specification FSM $\mathbf{S}$.

As an example, consider an FSM $\mathbf{S}$ in Figure 3.10. This FSM is non-observable, and in fact, 'bad' transitions are those under input i_3. Deleting these transitions brings to a deterministic projection $\mathbf{S}^d$ which is shown in Figure 3.11. This projection also happens to be a complete FSM with the set $I^d = \{i_1, i_2\}$ of inputs.

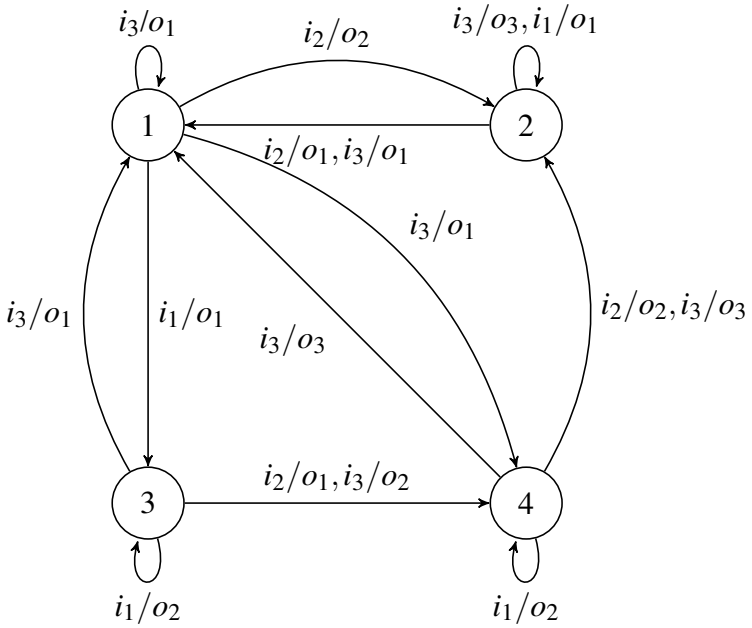

Figure 3.10: Complete non-observable nondeterministic FSM $\mathbf{S}$

A DTC for the FSM $\mathbf{S}^d$ can be of height 2; it is shown in Figure 3.12. It is also a DTC for the initial FSM $\mathbf{S}$.

Note, that such projections can be also applied not only to solve state identification problems but to simplify the test suite generation directly (whenever applicable).

Consider a fault model $< \mathbf{S}, \cong, FD = \{I_1, I_2, \ldots, I_k\} >$ where the potential faulty implementations are explicitly enumerated. The specification $\mathbf{S}$ is an initialized nondeterministic (possibly, non-observable) FSM, and $\cong$ is an adaptive indistinguishability relation that is sometimes referred to as r-compatibility relation. In this case, one can make use of the observable projections $\mathbf{S}^o$ and I^o of the FSMs $\mathbf{S}$ and $I \in FD$. In particular, one can derive a test suite distinguishing the mutants I_1, I_2, ..., I_k from the specification $\mathbf{S}$ via an adap-

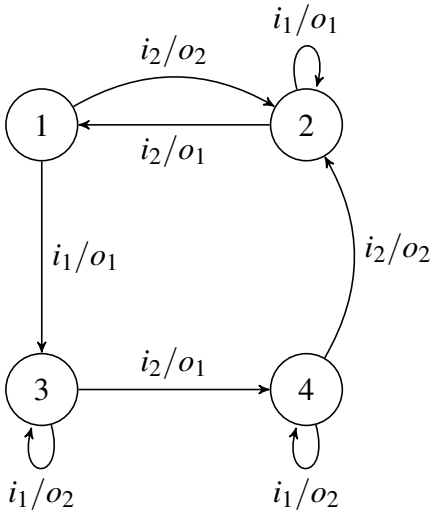

Figure 3.11: Deterministic projection $\mathbf{S}^d$ of the FSM $\mathbf{S}$

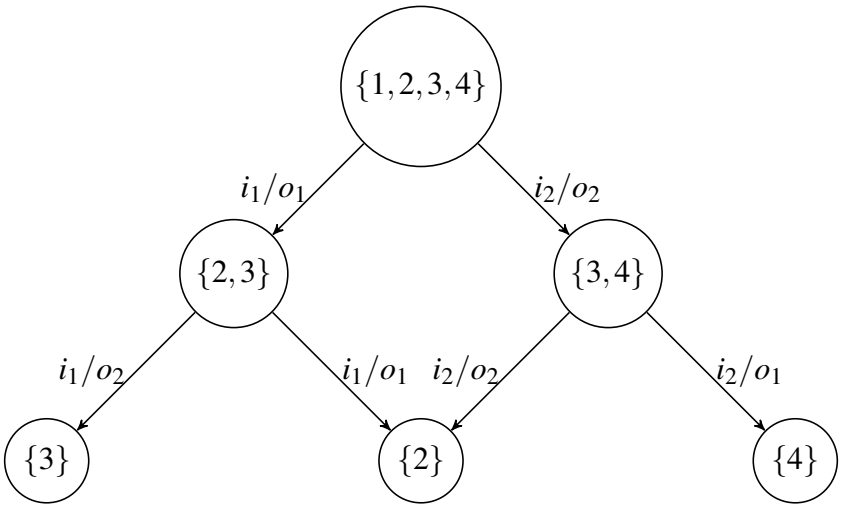

Figure 3.12: **DTC** for FSM $\mathbf{S}$ in Figure 3.10

tive sequence (if exists). The latter can be done through considering the corresponding direct sum $\mathbf{S}^o \oplus I^o$ for which an adaptive distinguishing sequence for state pair $\{s_0, p_0\}$ is derived, where s_0 is the initial state of the specification $\mathbf{S}$ while p_0 is that one of the implementation I. Each such DTC is added to the test suite TS. If for some I in FD there is no adaptive distinguishing sequence for corresponding observable projections then the test generation algorithm returns the answer *'An exhaustive test suite cannot be derived using observable projections'*. Otherwise, the set of all DTCs is an exhaustive test suite w.r.t. the fault model $< \mathbf{S}, \cong, FD >$.

An observable projection of the specification FSM represents a heuristic approach (published in [YKL+17]) for decreasing the complexity of the derivation of exhaustive test

cases for nondeterministic, possibly partial and non-observable FSMs. For two initialized observable possibly partial FSMs $\mathbf{S}^o$ and I^o with n and m states the length of an adaptive distinguishing sequence when it exists is at most mn [ACY95]. Therefore, this test generation procedure is polynomial with respect to mn, i.e., it solves the problem of an exhaustive test suite derivation for the fault model $<\mathbf{S},\cong,FD>$ in polynomial time with respect to the cardinality of the set FD. Note that in this case, FD contains the most probable mutants on the observable transitions of $\mathbf{S}$, since this method can only identify such mutants. Note that the faults that can potentially appear in the observable part of the non-observable system specification that includes all deterministic transitions, in fact reflect the critical, sometimes non-flexible system requirements. Deterministic transitions claim that no optionality of the output responses can be considered at a given state under a given input. Observable transitions, on the other hand, allow such optionality but only when the corresponding nondeterminism remains observable, i.e., when the next state is unique for each possible output response. Such deterministic and observable projections can exist for real protocol specifications, and as an example in [YKL+17] the Reader can find the File Transfer Protocol (FTP) specification and related mutants.

3.4.2 Probabilistic approach for test suite minimization against nondeterministic specifications

Another option to decrease the complexity of the test suite generation against a nondeterministic specification is to introduce some additional knowledge about this nondeterministic behavior. In [KYL21], together with the colleagues from ISP RAS and ADS, we propose to introduce the probabilities to the nondeterministic transitions. The latter in some cases, for observable FSMs, allows to minimize the test suite preserving the test suite exhaustiveness at a certain level.

Given the specification machine $\mathbf{S}=<S,I,O,h_S,s_0>$, we augment each nondeterministic transition $(s,i,o,s')\in h_S$ with the probability p. The probabilistic specification is thus the FSM $\mathbf{S}=<S,I,O,h_S,s_0,pr>$, where pr is the function that defines the probability for the output o to be produced at state s under input i, $pr : S\times I\times O\longrightarrow[0,1]$. Note that, we restrict the assignation of pr in such a way that $\forall s\in S\,\forall i\in I\sum_{o\in O} pr(s,i,o)=1$. The function pr can be extended over input/output sequences from $(IO)^*$; given an input/output sequence $\alpha/\beta=(\alpha'/\beta').(i/o)$, $pr(s_0,\alpha,\beta)$ = $pr(s_0,\alpha',\beta')*pr(s,i,o)$, where s is the α'/β'-successor of the state s_0 of the specification FSM $\mathbf{S}$; if the trace α'/β' is not defined at state s_0 then this probability equals 0. For the defined sequence γ such a successor is

unique due to the observability of the specification FSM $\mathbf{S}$; as usual $pr(s, \varepsilon, \varepsilon) = 1$.

Consider a fault model where $< \mathbf{S}, \cong, \{I_1, I_2, \ldots, I_k\} >$ and $\cong$ is a non-separability relation now, i.e., differently from the case above, we now consider only preset test cases. We adjust the non-separability relation for the augmented probabilistic specification FSM $\mathbf{S}$[13]. We thus define a *probabilistic separability* for a given implementation I_j and the specification $\mathbf{S}$. Given P as a user defined probability[14], a sequence $\alpha \in I^*$ is a $P\text{-}probably\ separating$ sequence for I_j and $\mathbf{S}$, if $\sum_{\beta \in out(I_j, \alpha) \cap out(\mathbf{S}, \alpha)} pr(s_0, \alpha, \beta) \leq 1 - P$. I_j is not probabilistic, and $pr(s_0, \alpha, \beta)$ is the probability to observe β when α is applied at the initial state s_0 of $\mathbf{S}$.

Correspondingly, for a fault model $< \mathbf{S}, \cong, FD = \{I_1, I_2, \ldots, I_k\} >$, we say that the test suite $P\text{-}TS$ is $P\text{-}probably\ exhaustive$ if $\forall I_j \in FD\ \exists \alpha \in P\text{-}TS$ such that α is a P-probably separating sequence for I_j and $\mathbf{S}$. We aim at deriving such test suites for user defined probabilities via *filtering* a given exhaustive test suite TS for the fault model $< \mathbf{S}, \cong, FD = \{I_1, I_2, \ldots, I_k\} >$.

In other words, given a user defined probability P, we propose to derive a test suite $P\text{-}TS \subseteq TS$, aiming at reducing $|P\text{-}TS|$ (in size), and which is P-probably exhaustive for $< \mathbf{S}, \cong, FD = \{I_1, I_2, \ldots, I_k\} >$. In order to do so, we propose to build a matrix M whose rows are assigned with the test sequences from TS while columns correspond to all the implementations from FD. $m_{i,j}$ contains the maximal guaranteed probability $p_{i,j}$ for the sequence α_i (in lexicographical order) to separate the implementation I_j (also in lexicographical order) from the specification FSM $\mathbf{S}$. This probability is calculated as $p_{i,j} = 1 - \sum_{\beta \in out(I_j, \alpha_i) \cap out(\mathbf{S}, \alpha_i)} pr(s_0, \alpha_i, \beta)$.

By construction, each column of the matrix M contains at least one 1, as the test suite TS is exhaustive. After M is derived what is left to do is to build a minimal cover of it, where a subset $P\text{-}TS$ corresponding to the rows, covers all columns $\{I_1, I_2, \ldots, I_k\}$, such that each probability $p_{i,j} \geq P$ where P is user defined. The latter means that for each potential faulty implementation from FD there exists at least one test sequence from $P\text{-}TS$ that P-probably separates it from the specification $\mathbf{S}$.

We omit the discussion about how such a row cover can be constructed - it can be done through an explicit combinatorial enumeration or various (combinatorial) optimization strategies can be applied. The solution to the problem always exists and in the worst

[13] Note that the notion of a probabilistic FSM has been introduced before, however the notion of distinguishability was defined differently (as non-equivalence, for example) [HM09], [ACY95], [VTdlH$^+$05].

[14] A level of certainty that a sequence separates the specification and an implementation

case scenario, when nothing could be minimized, $P\text{-}TS = TS$. Otherwise, if we are lucky enough, the overall test suite length can be minimized while the fault coverage can be preserved with the defined level P of certainty.

We note that other optimization strategies can be applied to simplify the state identification and test generation tasks, those can include various heuristics (some we discussed for example, in [KY15a]) or some scalable representations can be employed to accelerate the distinguishablity (see for example, [KYTS15]). These approaches keep developing as in fact, as discussed above, even if the general complexity is high for the related problems, many specific FSM classes still remain practical and at the same time allow a lower complexity. We will check some of the scalable representations for the FSMs in the next chapter.

Chapter 4

Decreasing the abstraction level - Testing against logic circuits: fault models and test derivation strategies

In the previous chapter, we discussed various contributions to testing *stateful* systems - we assumed that the output of the SUT at the next time instance depends not only on the input applied to it but also on its internal state. Moreover, we generally considered that this state remains unknown and thus, went deeper into the state identification problems. However, not all systems are like that; at least in the area of communication networks, *stateless* behavior is also met quite often. Logic circuits can effectively model both - it only depends if they have latches or not. Moreover, logic circuits can also serve as scalable representations to solve various analysis problems over FSMs, including again, the problems of the state identification (see for example, one of our recent works [TWJ+21]). Therefore, it is interesting to decrease the abstraction level when testing discrete event systems, and consider not high level descriptions for system behavior, but rather low, in terms of the corresponding logic circuit where inputs and outputs are represented by the Boolean vectors and states (if present) are 'stored' in the corresponding latches.

Given an FSM, a corresponding logic circuit that models its behavior can be obtained through the use of logic synthesis solutions (see, for example [VKBSV97, BM10, BT09, Sas93, DMBSV85, Tri16], among others). In this case, Boolean functions for the output and transition functions, should be derived, and there are various related research questions that have been widely studied and yet, novel improved solutions are needed and thus keep appearing. For example, when it comes to the logic synthesis, there are questions of

the resulting circuit optimality, including numerous criteria (number of gates, path length, delay, etc.) or state encoding (which partially relates to the first one). At the same time, for a given logic circuit a relevant FSM can be obtained via a direct simulation of the circuit behavior at each state under each input vector. Note that these approaches work under the *synchronous* behavior assumption, i.e., a clock signal makes the combinational logic produce its outputs at once, as well as the latches to change their states (just like in FSMs considered in the previous chapter, the output is produced simultaneously with the state change).

We do not focus on this FSM-circuit relationship in this manuscript but rather note that it is very well studied and can be useful not only for design but also for testing reasons, when an SUT has and has not internal states. Yet, we note that the faults that can be detected at the FSM level, not necessarily are detected when it comes to related logic circuit implementations, and vice versa. Indeed, for the logic circuit the faults of interest also differ - if in the previous chapter we mostly focused on detecting transition and/or output faults then now, when it comes to logic circuits, we can talk about a permutation of two logic wires, for example. We observed that in some cases the fault models developed for testing logic circuits can be improved, and this chapter is devoted to study of such fault models, correlation between them and related test suite generation. At the same time, when it comes to the communication systems, logic circuits can be quite adequate and efficient models for testing reasons, and in the next chapter we discuss one of related case studies.

4.1 Background: logic circuits for describing combinational and sequential behavior

A *logic circuit* consists of logic gates. Each logic gate has input (-s) and a single output. Outputs of some gates are connected to inputs of the others. The inputs of some gates that are not connected to any other gate output are claimed to be primary inputs of the circuit while the outputs of some gates are claimed as primary outputs. *Combinational circuits* are feedback-free and these circuits have no latches, i.e., the circuit output significantly depends only on its current input. Sequential circuits, on the contrary, have logic gates and latches or other types of flip-flops that can be considered as the internal circuit memory.

Note that the model of a logic circuit is of interest of the author, alongside with other models discussed before, such as FSMs and Automata. In fact, we utilized combina-

Table 4.1: Example LUT

x_0	x_1	x_2	x_3	z_0	z_1
1	1	1	1	1	1
0	1	1	1	1	1
1	1	0	0	1	0

tional circuits for testing SDN switches [LKB+18] as well as for estimating the quality of web/electronic services [KYC+14b, KPY+14]. Moreover, we evaluated the related FPGA implementations when a logic circuit is designed to implement a supervised machine learning algorithm [LLK+18]. Sequential circuits are used as scalable representations for FSMs and thus, we studied the related fault models and proposed the test generation strategies at the corresponding abstraction level.

We now come back to the definition of the circuits of interests and their representations that we use (-d) in our work. As usual, in a circuit, each logic gate implements a Boolean function. Most common 2-input gates are AND/OR/XOR/NAND/NOR/XNOR that implement conjunction/disjunction/xor and their inversions. There are also 1-input gates such as NOT/BUFF that implement the inversion and the equality function, correspondingly. There can be more complex gates that can have more inputs as well.

A logic circuit can be represented in various ways. For example, a circuit can be represented as a graph, and, in particular, an And-Invertor graph, such that logic elements are only represented by AND and NOT gates. In this case, these gates are called AIG nodes. An example of an AIG graph generated from logic synthesis and verification system ABC [BM10] is shown in Figure 4.1.

A combinational circuit implements or represents a system of Boolean functions. A circuit accepts a Boolean vector as an input and produces the Boolean vector as an output according to the corresponding system of Boolean functions. Each combinational circuit can be described by a Look-up-Table (LUT). A LUT contains a set of input/output pairs of a given circuit: if for the input **i** the circuit produces an output **o**, then the pair **i**/**o** is included into the LUT. An example LUT for a system of partially specified Boolean functions is given in Table 4.1. In fact, this LUT is used as an illustrative example in [KYC+14b] and it represents a very small dataset encoded as Boolean vectors for the Quality of Experience (QoE) level of 3 users with a given booking service.

The AIG in Figure 4.1 is the result of the logic synthesis by ABC, given the LUT in

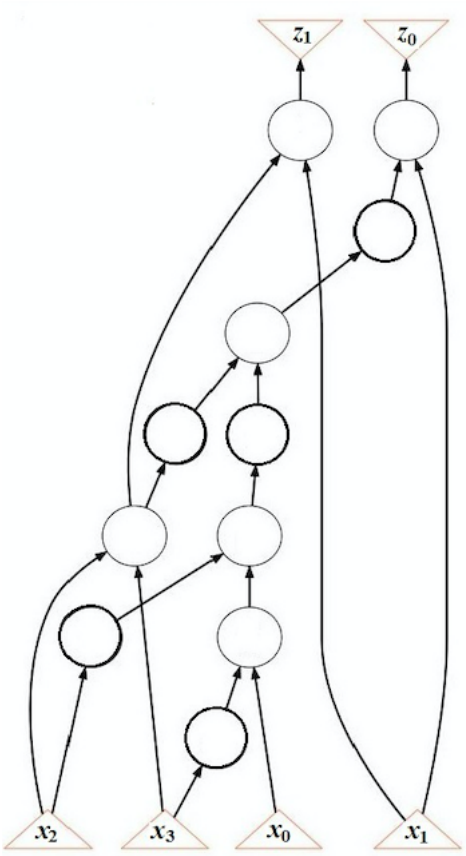

Figure 4.1: An example circuit in the form of AIG

Table 4.1 described in the special PLA format (NOT nodes are taken in bold).

Note that there are similar formats if we aim at synthesizing a sequential circuit, for example, from a corresponding FSM description. One of those can be KISS (or *kiss2*) format which is widely used as well in the description of various logic synthesis and verification benchmarks [Nij]. In fact, a KISS file contains a transition table of a *structural* FSM where inputs and outputs are encoded as Boolean vectors, however the states can remain abstract.

4.2 Novel results in testing logic circuits

4.2.1 Test generation based on logic circuit verification

We now turn to the test suite generation when the specification $\mathcal{S}$ is represented by a logic circuit which can be sequential or combinational. Some of the methods and techniques, as well as testing assumptions, are rather similar to those when testing against FSMs. For example, in the previous chapter, we discussed how to derive a test suite using explicit enumeration of potential mutants of the specification, i.e., under white box testing assumption. In this case, for each mutant, a corresponding sequence that distinguishes this mutant from the specification, is added to the test suite (if such a sequence exists). We now discuss how this strategy can be applied when the specification is represented by a corresponding logic circuit.

White box testing for Verilog descriptions

We hereafter consider another commonly used format for a logic circuit. In fact, we propose to derive the test cases directly from a circuit description in a hardware description language such as VHDL or Verilog. We use Verilog and the logic synthesis and verification solution ABC accepts it well. As an example consider a Verilog description in Figure 4.2 for a circuit $\mathcal{S}_1$, represented as an AIG, with a set $X = \{x_0, x_1, x_2, x_3\}$ of inputs, and a set $Z = \{z_0, z_1\}$ of outputs. This Verilog description is very simple and intuitive, it represents the AIG from Figure 4.1.

Let an SUT specification be given as a logic circuit which for example, is described in HDL or Verilog. For further test generation, we first define the distinguishability for the

```
module s1_for_ver (
    x0, x1, x2, x3,
    z0, z1  );
  input  x0, x1, x2, x3;
  output z0, z1;
  wire n6, n7, n8, n9, n10, n11, n12, n13, n14;
  assign n6 = ~x2;
  assign n7 = ~x3;
  assign n8 = x2 & x3;
  assign n9 = ~n8;
  assign n10 = x0 & n7;
  assign n11 = n6 & n10;
  assign n12 = ~n11;
  assign n13 = n9 & n12;
  assign n14 = ~n13;
  assign z0 = x1 & n14;
  assign z1 = x1 & n8;
endmodule
```

Figure 4.2: Verilog description example

circuits of interest. For the sake of simplicity, we start with combinational circuits. Similar to FSMs, two combinational circuits $\mathcal{S}_1$ and $\mathcal{S}_2$ are equivalent, if for each input vector **i** they produce the same output **o**. When talking about sequential circuits the equivalence relation is defined in more complex way, namely, for each input sequence applied at the current (initial) states, output responses of the circuits are the same[1]. If circuits $\mathcal{S}_1$ and $\mathcal{S}_2$ are not equivalent, then there exists an input vector **i** (an input sequence α for sequential circuits), such that the circuit $\mathcal{S}_1$ produces an output $\mathbf{o}_1$ (an output sequence β_1), the circuit $\mathcal{S}_2$ produces an output $\mathbf{o}_2$ (an output sequence β_2), and $\mathbf{o}_1 \neq \mathbf{o}_2$ ($\beta_1 \neq \beta_2$). In this case, the corresponding input (input sequence) is a distinguishing vector/pattern (sequence).

We are interested in deriving such distinguishing sequences for two or more Verilog descriptions and for that matter we rely on the well known Satisfiability problem (SAT) for a corresponding Conjunctive Normal Form (CNF)[2]. The CNF is derived for a miter of two circuits $\mathcal{S}_1$ and $\mathcal{S}_2$. Miter has the set of inputs $I = \{i_1, \ldots, i_n\}$, just like $\mathcal{S}_1$ and $\mathcal{S}_2$, and the set $\{m_1, \ldots, m_p\}$ of outputs. The function f_j that is implemented at the output j of the miter is the XOR of output functions g_j and h_j of the circuits $\mathcal{S}_1$ and $\mathcal{S}_2$ correspondingly. Circuits $\mathcal{S}_1$ and $\mathcal{S}_2$ are equivalent if each output of the miter always equals 0; otherwise, the corresponding input **i** that returns an output $\mathbf{o} \neq (0, \ldots, 0)$ is a distinguishing pattern for the circuits $\mathcal{S}_1$ and $\mathcal{S}_2$. Miter is a known technique that is used in logic synthesis and verification for checking if two circuits are equivalent (see, for example some related works of R. Brayton's research group [MCBE06, CMBK07, BEM12] who in fact developed the system ABC that we utilize).

We propose to use this miter-based technique combined with the white box test generation strategy, to derive an exhaustive test suite against a logic circuit specification [KYTS15]. Note again that if in Chapter 3 we were interested in detecting transition and/or output faults in the corresponding FSM, then now the fault domain should also be changed accordingly, i.e., now the mutants will be derived against the specification circuit.

We thus consider the fault model $< \mathcal{S}, \equiv, FD >$ where $\mathcal{S}$ is the Verilog-description of an SUT, $\equiv$ is the equivalence relation and the fault domain FD contains all possible mutants of interest for the description $\mathcal{S}$. Hereafter, as mutants, we consider the Verilog specifications with faults of specific types, such as for example replacements of one gate by another,

[1] The Reader may refer to [JVCH10, Pix92] for the related notions and algorithms in the area of hardware equivalence.

[2] Note, that in circuit design and verification, many tasks such as, for example, synthesis and/or optimization are often reduced to SAT or QBF solving (see, for example [ZJM21, BEK+14, BKS14]), due to the existence of quite efficient solvers, nowadays.

classical stuck-at faults, permutation of input wires[3], etc. As an example, consider a circuit $\mathcal{S}_2$ that can be obtained from the circuit $\mathcal{S}_1$ (Figures 4.1 and 4.2) by changing the input identifier of the AND gate z_1, namely, n_8 is replaced with x_2. The mutant $\mathcal{S}_2$ is distinguished from the specification $\mathcal{S}_1$ via the input **i** = $(1,1,1,0)$. Indeed, the circuit $\mathcal{S}_1$ produces the output $\mathbf{o}_1$ = $(0,0)$ while the circuit $\mathcal{S}_2$ produces the output $\mathbf{o}_2$ = $(0,1)$. Note that first, second, and higher order mutants of the Verilog description $\mathcal{S}$ can be considered, i.e., mutants with single, double, etc. faults.

We derive an exhaustive test TS for the $<\mathcal{S},\equiv,FD>$ in the exact same iterative manner as we used for the FSM specifications; just that in this case we are looking for an input pattern **i** (a sequence of length 1 for combinational circuits), satisfying the corresponding CNF. Deriving mutants one by one, we first check whether sequences (patterns) of already derived test suite can distinguish a current mutant $\mathcal{M}$ from the Verilog description $\mathcal{S}$. If this is not the case then we check whether there exists an input pattern **i** distinguishing Verilog descriptions $\mathcal{S}$ and $\mathcal{M}$. If such sequence exists then we add it to the test suite TS. If not, or the time, given for solving the corresponding SAT problem is over, then we stop the procedure or can turn to another mutant. The latter depends on the test objective, since in order to get an exhaustive test suite, we can only add such mutants $\mathcal{M}$ to the set FD that were distinguished from $\mathcal{S}$ in *realistic* time.

Note that we tried this technique using the system ABC and it usually returns an exhaustive test suite without 'dropping' some mutants, since a counter example for a satisfiable CNF is provided very fast. However, for some mutants the verdict can be inconclusive due to the lack of time when using SAT solving for equivalent circuits, that is quite rare.

Such white box testing approach against logic circuits seems to be promising since all the ABC operators work fast enough even for huge circuits. In particular, in our experiments we relied on the commands **miter**, **iprove** and **write_counter** to derive the miter, check if the result is SAT and if so, save the corresponding input **i**. It is also possible to use the equivalence check for the sequential circuits, namely **dprove**, but this obviously takes longer, in general. Some experiments with those circuits for different types of faults are presented in the next section.

[3]More details on the faults of interest are given in the next section.

4.2.2 (Novel) fault models when testing against the logic circuits and correlation between them

We consider three types of faults that can occur in the circuit implementation, namely Single Stuck-At Faults (SSFs), Single Bridge Faults, and Hardly Detectable Faults. Note that the first ones are very well studied (see, for example [Pat05]) and moreover the test suites derived against SSFs are known to detect many other faults. In [KLY16], we empirically verify this fact, namely we try to see how good are the test suites built against SSFs when it comes to other circuit mutants. We perform a similar investigation in [LKB+18] when studying this fault model w.r.t. the faults that can occur in forwarding devices for programmable networks (results presented in Chapter 5, accordingly).

For the fault models of interest, the mutants considered are the following:

- Single Stuck-At Fault Mutants that occur when one circuit gate gets 'stuck' at a given logical value ("1" or "0");
- Single Bridge Fault (SBF) Mutants that occur when a given input of a given logical gate is wrongly wired (bridged), i.e., taking the input from the wrong gate;
- Hardly Detectable Fault (HDF) Mutants that occur when a single gate changes its output for a single input.

We rely on the ABC tool, as discussed above with the **dprove** function to derive the test suites. The experimental results presented below concern the sequential benchmarks from the ITC'99 package (Second Release) [CRS00]. In [KLY16], the Reader can find the results on the test length and test size for the circuits b01, ..., b10 from the ITC'99[4]. We omit these results in here, considering that the correlation between the fault models can be more interesting. Such correlation is investigated as follows: after all test suites, under white box assumption, are built, each test suite is used to try to distinguish the mutants of two other different types. For example, the test suite derived for SSFs is used to try to 'kill' the mutants for the HDFs and the SBFs. The results of the experiments are shown in Figures 4.3, 4.4, 4.5, respectively.

As can be seen from the graphs, even well studied SSFs and related test suites against them can sometimes provide a low fault coverage for sequential circuits. Indeed, less than 40% of the fault coverage for some cases of the hard detectable faults is quite unpromising; we admit however that the bridges are generally detected easier.

[4]Circuit parameters and their brief descriptions are also given in the related publication [KLY16].

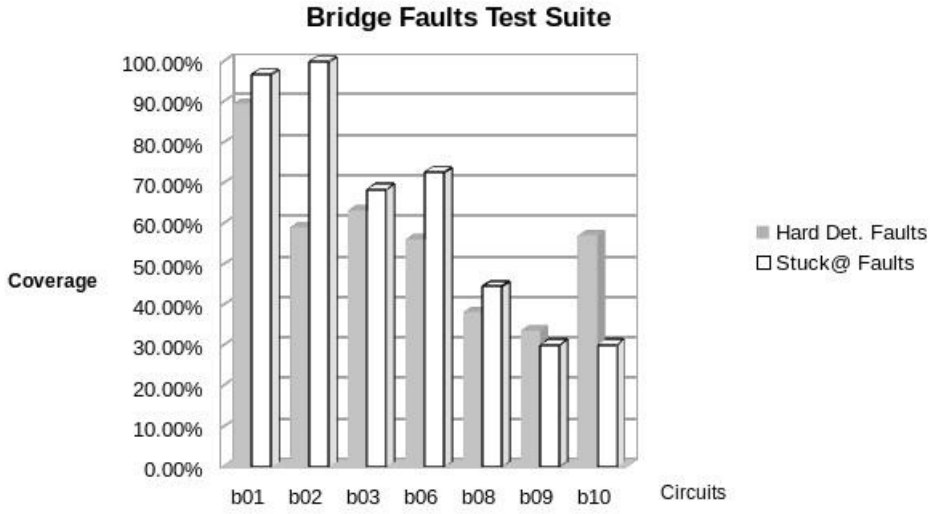

Figure 4.3: SBF test suite fault coverage against SSFs and HDFs

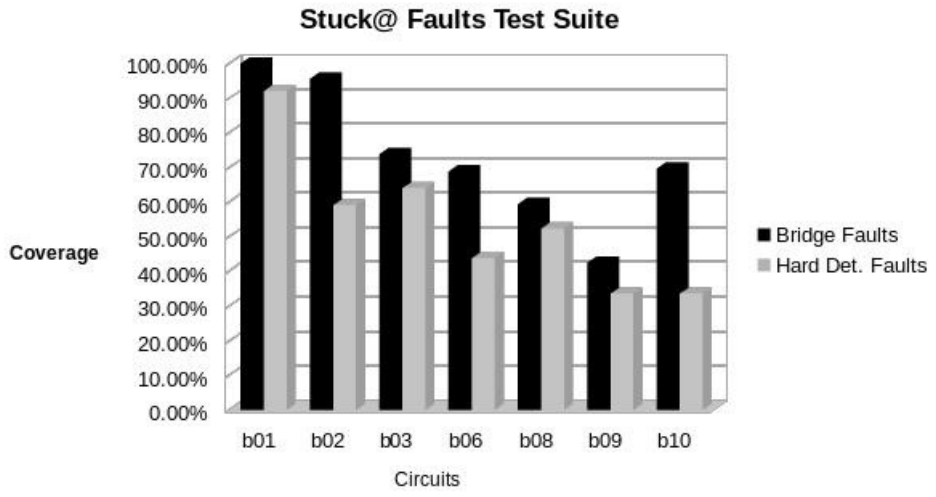

Figure 4.4: SSF test suite fault coverage against HDFs and SBFs

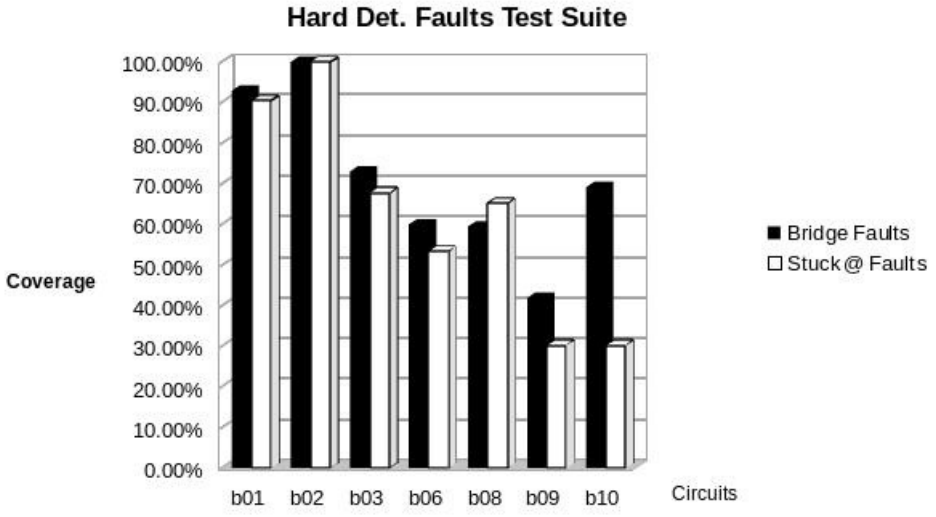

Figure 4.5: HDF test suite fault coverage against SSFs and SBFs

The experiments briefly presented above aim to motivate the necessity of studying different specifications when testing discrete event systems, as well as considering various abstraction levels. Indeed, for a software or hardware system that is developed for further production, the test cases should be provided and *collected* at different levels. In our work, we study two levels for sequential systems where FSMs or automata represent the high level description, while the logic circuits correspond to the low abstraction level. Note that not always the test suites that are *good* at one level, are *sufficiently good* at another level. In other words, it is interesting to study how faults detected (or guaranteed to be detected) at one level can correlate and be detected when moving to another abstraction level. We performed a small empirical study on that matter together with our colleagues from ISP RAS.

In particular, our colleagues from ISP (Russia), work on Extended FSM based testing strategies for hardware systems, and we decided to compare the fault coverage of the test suites for logic circuits and EFSMs. The experimental evaluation on the same circuits b01, . . . , b10 from the ITC'99 was presented in [SLK+16]. The goal of the experiments was very similar to that one above: which fault coverage can be achieved at a circuit level given the test suite derived from an Extended FSM for the sequential circuit in question? And vise versa: for the SSF and other logic circuit test suites, which faults can they detect over EFSMs? Below, we briefly present the test generation strategies and some results.

The EFSM based strategy employs a so called RETGA functional test generation method [IAS15]. The method has two phases: i) automatic extraction of the EFSM spec-

ification from the target HDL description; ii) generation of a test suite that covers each transition of the EFSM model (transition tour). For this test generation strategy we estimated, on the one hand, the statement or branch coverage in the hardware description language for the circuit of interest, and on the other, the fault coverage for the set of mutants of three types, i.e., SSFs, HDFs and SBFs. In fact, we united the mutants derived and considered this set together as a set of 'low level' mutants. We omit the details of the results of the experiments, inviting the Reader to check the publication, and note that for some circuits, for example b02, more than 80% of the low level mutants were detected. At the same time, for this circuit a 100% branch and statement coverage has been obtained with the RETGA tests. Nevertheless, for the circuit b10 a 100% branch and statement coverage for an HDL description does not bring good results for the low level mutants. Indeed, not even 40% of those are detected for this circuit with the RETGA-based test suite. Therefore, the test suites derived for the high abstraction level not necessarily detect the faults at lower abstraction level.

Similar experiments we performed also for the tests derived against the low level mutants. For the same circuit b02 we have more than 88% of the mutants coverage and 100% for the branch and statement coverage. However, again, this is not always the case, and for the circuit b10, we have $64,88\%$ of the low level mutant detection[5] and $93,24\%$ for the statement coverage ($85,71\%$ for the branch one). Somehow we can conclude that the low level tests reach higher levels of mutant coverage than high level ones, but they are longer, at the same time. Therefore, the conclusion can only be repeated - both abstraction levels are necessary and it would only be better if the test suites derived against the FSMs / automata can be combined with those against logic circuits, i.e., the SUT should pass both (as well as many others that are available) before being put into production / deployment.

We however note that the fault models are not limited with those discussed in the last two chapters - it is mostly the authors' interests that go around these very models. Nevertheless, depending on the application areas, we tried defining other original fault models and some of them we discuss in the next Chapter.

[5]For some mutants, a distinguishing sequence was not found using ABC.

Chapter 5

Testing and verification of dynamic and programmable networks

This chapter summarizes some of the application areas of several fault models considered above. As mentioned already, we try not only to solve the fundamental tasks in the area of model based testing, but also study some critical systems and investigate which fault models and test generation strategies could be appropriate. The author takes advantage of being part of the Networks and Mobile Multimedia Services Department at TSP. This fact presents the opportunities for collaboration, where the application areas are networks and in particular, programmable and dynamic networks. In such networks, some changes of the network topology or related network parameters are possible, during the system functioning. We therefore, investigated how the components of such networks can be tested and verified and which guarantees and at which levels can be provided.

5.1 Application areas: dynamic networks and SDN

Dynamic networks As mentioned above, such networks allow the network parameters associated to links to be changing their values at run-time. We say that a *static network* is a computer network where each link has a set of parameters that do not change, for example bandwidth (capacity) or delay. The parameters of the links may change in *dynamic networks*[1]. Static networks can be modeled as (directed) weighted graphs $(V, E, p_1, \ldots, p_k)$,

[1] We assume the network topology does not change in dynamic networks, and according to our industrial partner ADS with whom we work on this subject, this assumption is quite realistic.

where V is a set of nodes, $E \subseteq V \times V$ is a set of directed edges, and p_i is a link parameter function $p_i : E \to \mathbb{N}$, for $i \in \{1, \ldots, k\}$; without loss of generality, we assume that the parameter functions map to non-negative integers (denoted by $\mathbb{N}$) or related values can be encoded with them. Similarly, dynamic networks can be modeled as such graphs, however, p_i maps an edge to a non-empty set of integer values, i.e., $p_i : E \to 2^{\mathbb{N}} \setminus \emptyset$, where $2^{\mathbb{N}}$ denotes the power-set of $\mathbb{N}$. Note that a dynamic network snapshot, at a given time instance, is a static network itself.

Two main tasks of interest (mainly stated by the ADS again) for these kind of networks, in the context of verification and testing, are the following. i) Given a network topology and the functions p_i for the dynamic network parameters, one should verify that this topology satisfies the properties of interest. For example, one should check that each node has at least one coming and one outgoing edge, or that the topology is not fully mesh, for instance, etc. ii) Once a network is implemented, for example, as an emulated environment, can we check at run-time that the properties of interest are not violated? For example, can we check that the values of the dynamic parameters do not exceed a given constant? ADS is interested in these tasks in the framework of Satellite communications, and currently in the thesis of E. Petersen, supervised jointly with Jorge López and Djamal Zeghlache (thesis director), we are trying to apply the formal methods' strategies to the validation of such dynamic networks.

Software defined networking (SDN) These networks represent another case study for the model based testing and verification strategies. The *dynamicity* in this case is different: again, the topology is fixed, but given such a topology, one can implement various paths or various (virtual) networks on it. In dynamic networks, we assume that each node can generate packets and can also forward them, i.e., can act as a host and a switch at once. In SDN, we assume that the nodes are split into these two types, and for the sake of simplicity we assume that the related sets are disjoint. SDN allows *to program* a network through the requests to the application or the controller, and the request is implemented on the data plane. Such *programming* is performed via the necessary rules installed in the switches which, depending on the traffic type of the packet of interest, will forward it to the defined port. These forwarding rules are 'pushed' to the switches by the controller that has a global view of the network; switches are configured remotely and dynamically through interfaces using protocols such as the Open Flow (OF) protocol [MAB+08]. The SDN *resource* topology (data plane) or *resource network connectivity topology (RNCT)* is represented as an undirected graph $G = (V, E)$ where $E \subseteq \{\{a, b\} | a \in V \ \& \ b \in V\}$ without multiple edges and loops. The set $V = Ho \cup Sw$, $Ho \cap Sw = \emptyset$, of nodes represents network

devices such as *hosts* (the set Ho) and *switches* (the set Sw). Edges of the graph (the set E) represent connections (links) between two nodes in G and each link can transmit packets in both directions.

Several tasks of interest can be listed when it comes to verification and testing for SDN. i) What was requested at the application layer, is it exactly implemented on the data plane? ii) Does the controller/application/switch behave as requested by their specifications? iii) How is the interoperability between the entities listed? Can their be any races considered, for example? Indeed, what if two applications compete to access the controller and that the latter pushed the rules accordingly; will these races affect the final implementation of the requests on the data plane? iv) Finally, similar to the previous case study, given the network topology, one can check certain properties of it. Moreover, if together with the topology a virtual network request is also specified, one can also assure that this request is indeed implementable and *'makes sense'* before proceeding with its actual implementation. Some of these questions were studied during the thesis of A. Berriri (thesis director - Djamal Zeghlache) [Ber19] and also in collaboration with the ISP RAS researchers.

We present the problems listed above and some related solutions in a top down approach: first, we will discuss how to validate the provided specifications. Note that in the contributions previously presented, we never drew any attention to this question, namely, since the first general schema in Figure 2.1, we assumed that the specification of the SUT is provided and moreover it is valid, i.e., we can test the conformance of the SUT to this valid specification. In Chapter 4, we however considered extracting some specifications from the HDL code, but again we did not verify them. This might be a negative point and a future direction for the author to consider. Yet, we note that a large spectrum of works in testing and verification community are devoted (for almost half of the century) to the validation of the related specifications (see, for example, [HHL$^+$91, JL93, Hol82, vB75] for validating the protocol specifications using various means). We tried to apply some validation strategies in dynamic and programmable networks' area as we realized some network requests cannot be valid at all under certain environments[2]. Therefore, the first section is devoted to such verification with some very known formal verification means. Having validated the topology at the top, we come afterwards to *touching real* implementations that can be treated as SUTs and to which active and passive test generation strategies can be applied. Finally, when some critical network components are tested in isolation, we come to the discussion of their interoperability. On the one hand, we define some original fault models where the SUT is a composition of various network entities, and on the other,

[2]A simple example is a request to implement a virtual network that contradicts the physical resources, e.g., to request a virtual link when a physical one is not even provided.

we proactively test the possible races in the composition of interest.

5.2 Verifying the topologies and requests in dynamic and programmable networks

We start this validation part with a bit of motivation. Assume that we analyze certain virtual network requests and further their implementations. We will limit ourselves to the requests written in TOSCA language (Topology and Orchestration Specification for Cloud Applications) [DMT16]. In a well formatted TOSCA request (correct syntax, i.e., parsed successfully), a number of semantic inconsistencies can be present; we discussed several types of those in our work [LKYZ17]. If an inconsistent user request for a virtual network that for example, contains contradicting declarations, is deployed in the virtualization platform, where SDN can be one of the underlying technologies, unexpected or undesirable results might occur. Different constraints might be violated, such as for example service level agreements. As the consistency of the system is compromised by such requests, in one way or another the request is not properly implemented or it can even threaten the security and safety of the whole system.

Therefore, various properties of the virtual requests should be checked before any implementation on the data plane. First of all, any request should be free of *functional* inconsistencies, it should be *implementable* within a given topology. Moreover, *resource / dependency* issues should be also verified beforehand. Those can represent, for example, restrictions on the Virtual Network Function (VNF) neighboring connections or on the resources used by VNFs. Note also that such request validation should be performed at different abstraction levels. In fact, even if a request is expressed in terms of a set of virtual paths, without any labels such as VNFs on the nodes, not taking into account any dependencies between them, it still needs to be *pre-validated*. We do not draw much attention of the Reader to this issue, but just note that we studied these challenges and in fact, submitting even two paths as a request to an SDN controller can lead to more paths that can unintentionally appear on the data plane. In [BKY+21], we established the sufficient conditions when a given set of paths can be implemented without any undesired paths, i.e., no loops would appear for example, but this is only done under the assumption that all the packets have the same traffic type.

Similar motivation leads to validate the specifications for dynamic networks, where only the values of dynamic parameters are changing over the edges. It is very possible that

the topology description is not consistent, for example, that the links are not symmetric when they should be, or the network density is bigger or smaller than a given constant. Again, many of these properties can be directly verified on graphs, while for some a model checker can be employed. In [PLK+20], together with our ADS partners, we propose to utilize an SMT-solver for that matter and thus, describe the properties as Many Sorted First Order Logic (MSFOL) formulas. We encode the set of nodes V as an object of array sort; the sort of V is thus an array whose indices and values are integers. Directed edges are nothing more than records (tuples with sorts), particularly, pairs of integers. Dynamic parameters of the network, such as bandwidth or delay, are encoded according to their respective functions p_i. If the values of the parameters are integers and they belong to a finite set, then we enumerate the values (intervals in the simplest cases), depending of the source/destination node. For each parameter and each function, one can provide its own formula ϕ_i; their conjunction $\phi_{\mathcal{N}}$ (together with the topology formula as well), is considered afterwards for the verification. In fact, $\phi_{\mathcal{N}}$ represents the model of the dynamic network in question.

Having the properties to be checked, π as a conjunction of them, and the model $\phi_{\mathcal{N}}$, the model checking process is quite straightforward. First, we check that both $\phi_{\mathcal{N}}$ and π are satisfiable; otherwise, either the model (whose verification can be performed beforehand as well) or the properties have inconsistencies. Further, if the formula $\phi_{\mathcal{N}} \wedge \pi$ is satisfiable, then we conclude that the properties π hold for the model $\phi_{\mathcal{N}}$. If the formula is not satisfiable then there is a conflict between $\phi_{\mathcal{N}}$ and π. This approach was implemented using the solver z3 [DMB08]. The Reader can refer to [PLK+20] to see the properties that were checked for two dynamic parameters. We admit that the verification process is sometimes not very scalable, but it partly depends on the network type. For a network of 30 nodes, some properties take seconds, some can take up to 10 minutes to be verified, which also depends on the formula π.

As any networks and their components, dynamic networks can also be verified at runtime, i.e., a continuous monitoring or passive testing of an implemented dynamic network, can be performed. We note that such monitoring strategies of distributed architectures are also of the interest of the author, and apart from dynamic and programmable networks, we tried other distributed architecture scenarios, such as for instance Cloud Computing. For example, in the thesis of P. Carvallo (supervised jointly with Ana Cavalli, financed by Montimage company, thesis director - Stephane Maag) [Car18], we studied the security issues related to potential insider threat and discussed the relevant monitoring strategies. At the same time, in [LKZ17, LKZ19], we provided some active and passive testing solutions for the placement modules that assign virtual machines to hosts, in cloud infrastructures

[PB15]. We omit these results hereafter, inviting the Reader to check the cited thesis and publications. Instead, we show some online monitoring possibilities over the case study of a dynamic network.

Given a dynamic network, when monitoring its behavior, one can check that static network instances do not violate its dynamic description, i.e., the network model $\phi_{\mathcal{N}}$ introduced above. We can consider a corresponding conformance relation $\preceq$ which is similar to a reduction; a static network conforms to its dynamic description, if it has exactly the same topology and for each dynamic parameter at each edge e, its value belongs to the allowed set $p_i(e)$. In order to verify this relation $\preceq$ at run-time, the behavior of an SUT can be monitored for checking that each link $(v_a, v_b) \in V \times V$ is implemented correctly, and that each value of the i-th parameter belongs to the set $p_i((v_a, v_b))$ [3]. At run-time we verify that the value of $p_i((v_a, v_b))$ does not violate $\phi_{\mathcal{N}}$, and if this is not the case, an alert can be produced to signal the link (v_a, v_b) itself as well as the i-th parameter which was wrongly assigned when implementing the static instance.

Such passive testing or monitoring strategies in networks area are indeed efficient, however it is hard to provide any guarantees when it comes to this testing mode. The reason is that, differently from *active* testing where we build the sequences to be applied (preset or adaptive), in the passive mode, some of the sequences / traces might not even be observed during the testing time and thus, we cannot conclude about the SUT behavior on such *not shown* traces. We therefore try to study and apply both modes, passive and active, when it comes to networks' application area, and below we present the latter.

5.3 MBT for SDN enabled switches

In this section, we summarize the results presented in [LKB+18] that concern testing one of the critical components in SDN networks, namely an SDN switch. As mentioned above, a switch processes the received packets according to the installed forwarding rules. A *forwarding rule* consists of three parts: a packet matching part, an action part, and a location / priority part. The *matching part* describes the values a received network packet should have in order for the rule to be applied. The *action part* indicates how to process the matched network packets; the *location / priority part* controls the rule hierarchy using tables and priorities. A *rule* R defined in the switch configuration can therefore be represented

[3] We assume that the points of observation in this case can be placed at each node $v \in V$ and each link (v_a, v_b), correspondingly.

by the following implication: $(p_1 \in V_1 \ \& \ p_2 \in V_2 \ \& \ \ldots \ \& \ p_i \in V_i \ \& \ \ldots \ \& \ p_n \in V_n) \implies output\_ports = \{o_1, o_2, \ldots, o_m\}$. In this case, p_i refers to an input parameter, o_i refers to an output port and the sets V_1, V_2, ..., V_n define a range or an interval for each switch parameter p_1, p_2, ..., p_n, correspondingly[4].

We would like to check that the switch is implemented correctly, i.e., that it indeed implements the set of predefined rules. In order to introduce a fault model, given the implication above, we introduce two types of possible mutants, i.e., which mistakes can happen to a rule R when being pushed to a switch under test.

An *output mutant* for the rule R is defined as follows: $(p_1 \in V_1 \ \& \ p_2 \in V_2 \ \& \ \ldots \ \& \ p_i \in V_i \ \& \ \ldots \ \& \ p_n \in V_n) \implies output\_ports = \{o'_1, \ldots, o'_{m'}\}$, and $\{o'_1, \ldots, o'_{m'}\} \neq \{o_1, \ldots, o_m\}$.

A *parameter value mutant* for the rule R is defined as follows: $(p_1 \in V_1 \ \& \ p_2 \in V_2 \ \& \ \ldots \ \& \ p_i \in V'_i \ \& \ \ldots \ \& \ p_n \in V_n) \implies output\_ports = \{o_1, o_2, \ldots, o_m\}$, and $V'_i \neq V_i$.

We assume that the switch implementation has no faults if each packet is processed exactly in the way that the switch configuration requires. Moreover, if for a given packet there is no rule in the switch configuration with the corresponding matching part then this packet is simply dropped by the switch, i.e., forwarded nowhere. The fault model in this case is $< Switch, =, FD >$ where $Switch$, the specification, is the set of switch rules; $=$ is the conformance relation represented by the equality, and FD is the fault domain where the potential switch implementations are explicitly enumerated. We thus consider a white box testing assumption; potential faulty implementations correspond to the mutants introduced above. Note also that the system specification can be complete or partial. The set $Switch$ of switch rules is said to be *complete* if for each preamble $(p_1 \in V_1 \ \& \ p_2 \in V_2 \ \& \ \ldots \ \& \ p_n \in V_n)$ there exists a rule $R \in Switch$ such that $R = ((p_1 \in V_1 \ \& \ p_2 \in V_2 \ \& \ \ldots \ \& \ p_n \in V_n) \implies output\_ports = \{o_1, o_2, \ldots, o_m\})$. Otherwise, the specification $Switch$ is *partial*.

Note that the specification $Switch$ represented by rules of type R has no memory and thus can be effectively modeled as a combinational circuit. Moreover, it is intuitive to consider Boolean representations for values transmitted in network packets as they represent data in binary strings. A combinational circuit LC can be derived in different ways; in [LKB+18] a straightforward approach for this purpose is proposed. We rely on the use of logic synthesis solutions, in particular, ABC again, from LUT for a system of (partially specified) Boolean functions. Once a logic circuit LC that simulates the behavior of the switch with the rules $Switch$ is derived, one can apply different techniques for test generation. We decided to apply those, discussed in Chapter 4, i.e., we consider the SSF, HDF, SBF faults

[4]The defined intervals are assumed to contain integers, without loss of generality.

in circuit LC and discuss which fault coverage it can bring against output and parameter value mutants over switch rules. It turns out, some guarantees can in some cases be given when it comes to SSFs.

Proposition 9 *If $Switch$ is complete and $\exists\, i \in \{1,\dots m\}$ such that $\exists!\ R \in Switch$, $R = ((p_1 \in V_1\ \&\ p_2 \in V_2\ \&\ \dots\ \&\ p_n \in V_n) \implies output\_ports = \{o_1, o_2, \dots, o_m\})$ and $o_i \notin \{o_1, o_2, \dots, o_m\}$, then each output fault in the rule R is detected by an exhaustive test suite w.r.t. SSFs.*

Whenever $Switch$ is not complete, the behavior of LC over the undefined patterns can be specified in different ways. In our approach and in our experiments, we use ABC, which sets the corresponding outputs to 0. This fact allows to guarantee the fault coverage for output mutants of the rules when initially the specification $Switch$ is not complete.

Proposition 10 *If for a set of rules $Switch$ $\exists\, i \in \{1,\dots m\}$ such that $\exists!\ R \in Switch$, $R = ((p_1 \in V_1\ \&\ p_2 \in V_2\ \&\ \dots\ \&\ p_n \in V_n) \implies output\_ports = \{o_1, o_2, \dots, o_m\})$ and $o_i \in \{o_1, o_2, \dots, o_m\}$, then each output fault in the rule R is detected by an exhaustive test suite w.r.t. SSFs.*

Therefore, if in the set $Switch$ of switch rules, each output port is used in at most one rule, then an exhaustive test suite w.r.t. SSFs is also exhaustive w.r.t. rule output mutants. The above statements do not necessarily hold for the parameter value mutations. Such faults can in some cases be detected by other test suites such as HDFs or SBFs.

We checked the related fault coverage empirically, performing the experiments with an Open vSwitch (OVS) [PPK+15]. The Reader can refer to [LKB+18] for the details of experimental setup and results. We just mention that the topology was small, containing 4 switches, all connected to ONOS controller and 4 hosts, each connected to its own switch. One of the switches was declared to be an SUT and a circuit LC was derived using ABC, to model it. We generated 45 mutants of various orders for the rule R of the specification (1 got to be equivalent so we deleted it) and designed the test suites using the SSF, HDF and SBF strategies. We also tried to re-synthesize the circuit LC, to obtain other mutants and thus, get a richer test suite. As re-synthesis result (LC') is not always minimal, indeed, the circuit can have more gates, for example, and this can help increasing the fault coverage. Note that the resulted circuit LC' is an AIG. The experimental results are shown in Table 5.1, where the last column refers to the union of all 3 test suites, i.e., *ACF* stands for 'all circuit faults'.

Table 5.1: Fault Coverage for digital circuit fault models

Circuit	SSF	SBF	HDF	ACF (total)
LC	79%	45%	18%	86%
LC'	95%	97%	95%	100%

We conclude that test suites derived based on logic circuit fault models have high fault coverage for SDN-enabled switch faults. An interesting aspect is that the fault coverage highly increases when the original circuit specification is transformed into an AIG.

Note that similar to the run-time verification of dynamic networks (or other architectures), the same can be done with the SDN switches, when the specification circuit LC is obtained. The simulation of LC can in some cases be much faster than the search of the particular switch rule and its further application to conclude about the expected output port(-s). Therefore, the task of the switch monitoring that verifies that the packets are forwarded to the exact ports specified by $Switch$, can be reduced to the problem of the LC simulation, i.e., obtaining an output pattern for a given input one. We performed such experiments as well, to see how fast this simulation is. We measured the time that the switch takes to process a packet as the difference between the packet ingress and the packet egress. And for the Open vSwitch under test it was ~ 0.29ms. The time taken to simulate a single pattern in LC was measured to be ~ 0.003408ms, on average. This not only encourages to use this formalism in model based monitoring for forwarding devices, but maybe even consider the synthesis of those, using logic synthesis solutions. Such logic circuit based design for forwarding devices is one of the directions to consider in the future.

We mention again, that all this modeling and test generation was performed under the *stateless switch* assumption. If we assume that a switch can have memory, for example, it would not drop the packets but would ask the controller what to do with them, or any other *intelligence* would be introduced, then maybe stateful models would be more appropriate (sequential circuits, FSMs, etc.).

5.4 MBT for SDN frameworks and related fault models

It is important to test each network component in isolation, but at the same time, it is also important to test and verify their interoperability. In this section, we consider a bigger system under test, assuming that maybe network components contain certain inconsistencies

but for the end-user or a tenant what counts is the final result. In other words, maybe both a controller and a switch have some bugs but these bugs are masked one by another and what matters is that the user request for a certain network or a path is implemented correctly. We therefore propose to test the SDN framework, including the controller(-s), switches, and connections between them. A tester then sends specific requests to the SDN controller asking for different paths to be implemented in the RNCT.

We assume that the SDN infrastructure is *functioning correctly* when each requested path and only it is created. Hereafter, we consider a virtual path (or simply a path) as a sequence of directed edges whose head and tail nodes are hosts and all other intermediary nodes are switches. As in fact, mostly connectivity issues are tested, similar to [ETSY04], the fault model this time has only two items, $<=, FD>$ where the conformance relation is again the equality; no specification is included in the fault model.

The following types of faults can be considered: a requested edge can be directed to a wrong node, additional edges can appear as well as some edges can disappear. Thus, a fault domain FD contains all possible paths of the RNCT. A test case is thus a path of the RNCT and a test suite is a finite set of paths. We are interested in exhaustive test suites such that any difference between a requested and implemented path can be detected. Such test generation strategies for black and white box testing were proposed in [BLK$^+$18] where for obtaining an appropriate output reaction of the SUT we relied on the traffic generation (see, for example, [ZKVM12, DSO14, FYT$^+$16] for some works on this subject).

For the *Black Box Testing*, as the set of all paths of the RCNT is finite, the simplest way to construct an exhaustive test suite w.r.t. $<=, FD>$ is to consider the set of all such paths. However, this test suite is rather long, and a possibility to reduce the test suite size is to consider equivalence classes for the paths. For example, if we assume that each node processes inputs independently of the node where they come from, two paths can be considered (i, j)-*equivalent* if both paths have a directed edge from node i to node j. That is either all the packets that should be directed from i to j are processed correctly, i.e., are sent from i to j, or are processed wrongly, i.e., are sent anywhere except the j-th node. In this case, the set of paths that contains a path of each (i, j)-equivalent class where (i, j) is an edge in the RCNT, is an exhaustive test suite w.r.t. the corresponding fault model $<=, FD>$.

In order to cover all equivalence classes in an optimal way, an optimization problem can be stated and solved. One option is to consider the Boolean (weighted) matrix and solve the corresponding covering problem [VKBS97] for which many libraries and scalable

software solutions are developed.

If the node processes a packet depending on where it comes from then equivalence classes could be considered w.r.t. path subsequences of length $l \geq 2$. Given a sequence γ of RNCT edges of length l between node i and node j, two paths are considered γ-equivalent if they both contain γ. A test suite is exhaustive if it has at least one path of each equivalence class. A minimal cover of a corresponding Boolean matrix can also be used for optimal test generation.

For the *White Box Testing*, as usual, only the mutants of interest, enumerated explicitly, can be considered. One can define *critical edges*, that need to be tested first; for example, critical edges that include critical services. In this case, in FD, implementations can potentially contain three types of faults that need to be detected. i) An edge is directed to a wrong node, i.e., from the edge of interest (v_i, v_j) to $(v_i, v_{j'})$ where $j \neq j'$. ii) An edge $e = (v_i, v_j)$ is deleted. iii) A non-existing edge (v_i, v_k) is created for a critical edge (v_i, v_j), where $j \neq k$. We propose to generate a test suite so that all the critical edges are covered. For a given critical edge (v_i, v_j), one can first backtrack to find a shortest sequence that starts in a host and finishes at the node v_j, say p_1. Further, a forwardtrack can be performed to find a shortest sequence that starts at the node v_i and finishes at a host, say p_2. The sequence $p_1.p_2$ is then added to the test suite. As each critical edge is covered by construction, the derived test suite is exhaustive w.r.t. $<=, FD>$ where FD has explicitly enumerated mutants of types i)-iii). This somewhat naive approach can be also improved. A test minimization can be performed similar to black box testing approach, via solving a covering problem. In this case, a minimal set of paths that cover all critical edges can be identified.

The fault models considered when testing SDN frameworks can include other parameters of switches, for example, such as input and output ports through which a packet goes. In [YBK+18], we discussed some related equivalence classes for example, defining another equivalence relation when two paths are equivalent if a given switch processes inputs independently of many parameters only paying attention to where a packet came from and what is its destination port. More parameters from the packet header can be included to define other equivalent classes, we are currently working on defining *the most relevant* parameters for various traffic types and respective equivalence classes.

5.5 Formal verification for pro-active testing when detecting SDN races

In this section, we focus on another kind of interoperability issues. In fact, we now try to see when such issues cause misconfigurations on the data plane. One of the reasons is the presence of races in the messages that application, controller, and switch exchange. In [VLK+19], we proposed a proactive testing approach for detecting such races in SDN frameworks. We note that the problem of SDN races has been raised and studied before us. Moreover, there are some existing utilities, such as for example SDNracer [EMB+16]. However, the approaches proposed before rely on the definition of a specific (partial) order between the possible SDN events and further monitoring if the defined order is violated. For example, in [MBE+15] the authors derive a specific *HB graph (happens-before model)* to identify the order of events. Therefore, the detection stays *passive* in this case. Another possibility is to take a *preventive path*, i.e., to derive the SDN components that are carefully synchronized, so that races cannot show up [MHC17]. Despite the fact that test generation based on formal verification has been largely considered before [FWA09], we however are not aware of the application of such test generation strategies for race detection in distributed systems.

As a result of several traineeships of E. Vinarskii at TSP (under joint supervision with D. Zeghlache), we therefore proposed a complementary approach [VLK+19] which is based on proactive testing, i.e., on generation of specific application requests that can lead to a race in an SDN framework. For that reason, we model the SDN components by a simplified version of an Extended Finite Input/Output Automaton, EIOA, for short.

An EIOA $\mathbf{A}$ is a tuple (S, I, O, V, T, s_0), where S is a finite nonempty set of states with the designated initial state s_0; I and O are finite *input and output alphabets*; V is a finite, possibly empty set of *context variables* with the set D_V of vectors of context variables' values if $V \neq \emptyset$; T is a set of transitions. In our case, inputs and outputs are parameterized, i.e., inputs and outputs of the EIOA are pairs $(\textit{input, vector of input parameters' values})$ or $(\textit{output, vector of output parameters' values})$ and D_I (D_O) is the set of vectors of input (output) parameters' values if the set of parameters is not empty. A transition is a 6-tuple (s, a, P, v_p, v_o, s') where $s, s' \in S$ are initial and final states of the transition; $a \in I \cup O$; $P : D_V \times D_I \rightarrow \{True, False\}$ is the *transition predicate*; $v_p : D_V \rightarrow D_V$ is the *context update transition function*. The transition (s, a, P, v_p, v_o, s') is executed only when the transition predicate P evaluates to $true$ and the vectors of context variables' values and output parameters' values are updated according to the functions v_p and v_o after the

transition execution.

We also assume that when dealing with an EIOA, no input is accepted and no output is produced when a transition is executed. However, when both an input and an output are defined at a state, they can 'compete' between themselves, i.e., what the machine does first - accepts the input or produces the output is nondeterministically decided[5]. We hereafter refer to such 'competition' as an *input/output race* (a race between input and output actions). When an EIOA works in isolation, the races can occur within certain states. However, when the machine acts as a component of a multi-agent system, other types of races can take place. In particular, the communication channels can serve as 'tunnels' where the actions 'compete' to be faster for reaching the output, we refer to this type of races as *intra-channel* races. In the given SDN case study, we are concerned with the interactions of three entities, namely the application, controller, and switch. Therefore, potential races in the channels cover either the northbound or southbound interface. The interacting entities are depicted in Figure 5.1. Note that we limit ourselves to 1 application, 1 controller and 1 switch[6].

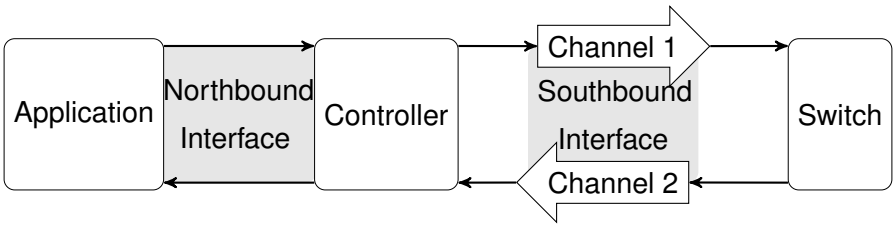

Figure 5.1: SDN Topology considered for the races' detection

For the race detection in EIOAs and their compositions, we rely on the formal verification based on the Linear Temporal Logic (LTL) formulas and the correlation between LTL formulas and EIOA properties. We therefore built the EIOAs modeling the SDN components of interest and described the related behavior in the Promela language to further utilize the SPIN model checker [Hol03]. The Reader can refer to [VLK+19] to find the appropriate descriptions that we hereafter omit, focusing rather on the test generation strategy. Note that as the races can occur in the channels as well, both were also modeled as appropriate EIAOs.

To detect the Input/Output races in a given state of a given automaton, we proposed a probabilistic approach. The strategy is based on a choice of a state s of the EIOA, where both input i and output o *related via some parameter* are defined. The Spin model

[5]Note that such behavior is completely different from the FSMs, considered in Chapter 3, where an input is always followed by an output.

[6]We are currently working on increasing the number of interacting entities.

checker is used for verifying the LTL formula *prohibiting* a race between i and o. For example, let $curr\_mess$ denote an input or an output action (message) at state s while $next\_mess$ denote the action at the next time instance. To guarantee that the output o is never produced before the input i is received, an LTL formula of the following kind can be verified: $\mathbf{G}(\neg((curr\_mess == o) \rightarrow (next\_mess == i)))$. If the formula of interest can be violated for the component of interest, then Spin produces a counterexample α. Note that not each counterexample is feasible for the topology of interest; further checking should be performed by applying a sequence of inputs simulating α, with appropriate timeouts to an implementation (component of interest). As the approach is proactive, we execute the derived counterexample α potentially leading to a race against the implementation of the SDN framework. For that reason, we rely on an automated script; the script is executed at most N times, simulating α as an input to the component of interest. If during the execution of the script the *competition* between an input and output can be observed, then the script returns $TRUE$. Otherwise, the script returns $FALSE$. The approach therefore is implemented as a randomized algorithm that can be *trusted* on a race detection. However, if the returned reply is $FALSE$, we cannot be 100% sure there are no Input/Output races in an SDN component of interest.

We proposed a similar probabilistic approach for the intra-channel races, when for example the rules or requests submitted into a channel by a given component can be permutated. In the case of the experimental evaluation those are the rules submitted by the controller and later on pushed to the switch. In order to detect if there exists a potential permutation in the channel, we try to proactively create this race condition. To do so, the SDN application installs rules with their IDs in chronological order; moreover, each rule has a hard timeout equal to its ID. To derive several LTL formulas we express the dependencies between the rule i and the rule $i+1$. For example, we state that $\mathbf{G}((table[i+1] == 0) \rightarrow (table[i] == 0))$, i.e., the rule i should be deleted before the rule $i+1$. The derived properties together with the original model are then 'fed' to Spin. If Spin does not detect a violation of any of the properties then not a single race of interest can be detected. This does not necessarily guarantee the absence of such races. However, if a counterexample α is produced then its feasibility is first verified; a non-feasible counterexample again results in the '*not detected*' ($FALSE$) conclusion. On the contrary, if α is feasible, and during the N times that the corresponding script was executed, a race was indeed detected, then the result $TRUE$ is returned.

We roughly estimated the probability of success of these randomized algorithms for the specifications derived for ONOS controller and an Open vSwitch. The calculated probability was rather low, i.e., around 30 and 50%. However, experimentally we managed to

indeed observe the permutation of rules, therefore we managed to detect a race between the $PostFlow$ requests in the related channel. We are currently working on the generalization of this approach, trying to define the races through introducing a special order over inputs and outputs in automaton states. The latter can probably allow defining races in longer traces but this point needs to be investigated. Another challenging issue is the test suite exhaustiveness for the races' detection; we are currently working on it as well.

Chapter 6

Conclusions

6.1 Some concluding remarks

We hereafter briefly summarize the results that were presented in the manuscript, trying also to give a certain evaluation and critics to those.

The research directions of the author mostly cover model based testing and verification and their applications to some network components. The main models considered in the work are FSMs and automata as well as their scalable representations as logic circuits. We thus contributed to the testing strategies against these models (active as well as passive) and worked on the tasks that on the one hand, lead to test generation with the guaranteed fault coverage, and on the other, analyzed the related complexity and studied some possible *simplifications*, such as extraction of some classes with the reduced complexity of test derivation.

In the area of FSM/Automata based testing, we in particular, focused on the state identification issues for nondeterministic specifications, as these ones are met quite often *in real life*. More precisely, we proposed novel techniques for deriving preset and adaptive homing/synchronizing and distinguishing sequences that allow identifying current and initial states of the machine. We also evaluated the length of some of the sequences and showed that similar to the deterministic FSMs, in some cases, moving from preset to adaptive strategy can decrease the length of the sequence from exponential one to polynomial. Methods for deriving homing and synchronizing sequences for nondeterministic FSMs can be effectively utilized for deriving those for finite Input/Output automata, where transitions are labelled either by inputs or outputs; in some cases no inputs even need to be applied

to determine the current state (output observations are sufficient). Given the high complexity of the FSM based test generation strategies, we paid a particular attention to some possibilities (special classes of the specifications) when the worst case complexity cannot be reached. We also discussed other approaches to decrease this complexity, such as, for example, considering white box testing approaches instead of black box. Moreover, taking into account specific projections in the latter case, can allow deriving an exhaustive test suite of polynomial length w.r.t. the number of states of the specification machine. When talking about the passive testing, we also proposed certain optimizations/improvements based on the current state identification; indeed, knowing the current implementation state can reduce the number of properties to be checked on- or off-line.

It is not a simple task to assess how interesting are the results that were summarized above, from the practical point of view. On the one hand, we consider more and more complex specifications, where the nondeterministic behavior can be also non-observable, for example. At the same time, we started to work with input/output automata where not each input is followed by an output and proposed related state identification techniques for them. On the other hand, we must say that many and even the majority of these results currently stay just fundamental, and as seen from Chapter 5, sometimes the fault models and the specifications are simpler (no states, for instance) or on the contrary, more complex (they are extended, with predicates, parameters, etc.), and so there is still somewhat a gap between the results of Chapters 3 and applications in Chapter 5, which we plan to overcome step by step in our future work.

In the area of test generation against logic circuits, we studied well known faults models, such as single stuck-at faults, and also introduced some others. We proposed the techniques for deriving distinguishing sequences for such models, and thus, adjusted the white box test generation strategy for deriving the test suites using logic synthesis and verification solutions. Logic circuit based solutions have been also adopted for deriving homing sequences for nondeterministic FSMs; in this case, the problem is reduced not to SAT solving but to QBF solving. We analyzed various fault models against combinational and sequential circuit specifications and applied some of them to testing network components, such as an SDN switch. Due to room limitations, we did not present much of the related results, but we also studied the combinational circuits for quality evaluation/prediction of some electronic services and this model also seems appropriate. We generalized the corresponding logic synthesis strategy later on, for implementing some supervised machine learning technique, with its further FPGA implementation.

As for the drawbacks, some critical remarks on the related results need to be taken

into account. Almost in all our experiments, logic synthesis and verification solutions stay a *software*. In other words, we do take advantage of certain scalability, for example, when it comes to simulation tasks, however we still rely on simulating the circuit in a software and not in a hardware. This part needs to be strengthened, as logic circuits in particular are very natural to be implemented in FPGA, for example, and existing logic synthesis solutions allow Verilog and HDL descriptions, so we need to try to take advantage of this fact in the future. This might help not only to derive the test cases but even to implement the related components in hardware, such as for example, the switches mentioned above.

In the application areas, we considered networks with certain *dynamicity*. In particular, those are either dynamic networks where the topology stays the same but parameters on the links change their values (from a given set) in time; or programmable networks where on a physical or resource topology several virtual networks or paths, can be implemented (can be later reconfigured if necessary). We analyzed the necessity of the validation of the related user requests, on the first place. Later on, partially followed by the user-defined specifications, we proposed some test generation and monitoring strategies for this kind of networks, and estimated the guarantees that can be provided, in terms of the test suite exhaustiveness. We applied some of the test generation strategies based on automata and logic circuit models, to verify the programmable network components in isolation and also when checking their interoperability. In some cases, we introduced the novel fault models that reflect better the related SUTs and their potential faults, in SDN networks, for example.

Evaluating the results related to testing and verification of specific kinds of networks, one can notice what was already mentioned before - the models utilized *in real life* as specifications, are sometimes much simpler or much more complex than those known as (nondeterministic) FSM or automata. Filling this gap, and bringing more MBT solutions to network components before the deployment (and even after) is an interesting challenge for the future research directions (probably, not only of the author).

6.2 Future / current work and perspectives

Some of the research questions raised in Chapter 1, have been answered in the related publications and the replies are partially included in this work. As usual, more questions appeared and keep appearing and they need further investigation.

In the area of FSM/automata state identification, we are trying now to consider rather

non-classical state models, for example, extended FSMs and automata (where both inputs and outputs can be defined at the same state). Not much work has been done in the community in this area, but there are already works on distinguishing EFSM configurations [PBG04], deriving homing sequences for timed machines [TY20], testing against FSMs with time guards, timeouts [EYF09, TEY18], etc. However, such models can differ a lot, depending on the restrictions that can be put on the predicates/guards, and therefore, more research is needed in this area. At the same time, not everywhere the complexity of the related problems has been estimated - some upper bounds on the length of the sequences are not known to be reachable (even for *classical* deterministic specifications, where the Cerny conjecture can serve as one of illustrative examples). The hardness of certain existence check and derivation problems of state identification sequences is not yet proven.

Another example that can be considered here is a homing sequence for a nondeterministic complete observable FSM where we managed to prove the reachability of an exponential upper bound of the order $O(2^{n-1})$ for a machine with n states. However the upper bound that comes from the truncated successor tree is of another order, i.e., $O(2^{n^2})$, therefore this gap needs to be filled in, either moving the left border of the interval or the right one, or both. Determining the corresponding length, and deriving the classes of machines with the *bad* properties to reach such bounds is one of the directions of the current/future work of the author. As there exist also *good* FSM/automata classes for which the worst complexity upper bounds are not reachable, and thus, these machines can serve as *good* specifications for *easy* test suite generation, we continue studying those as well.

At another abstraction level, when a machine is represented by a logic circuit, we need to better study the advantages. First of all, obtaining such a circuit - if the specification was not given this way, is already a complicated task. Second, we are not avoiding any complexity but rather change the problem, expressing for example, homing sequence existence check using QBF formula. Therefore, we need to again study the classes of machines when such a problem reduction is useful. The correlation between the fault models at different abstraction level should be studied thoroughly. Ideally, for a case study of interest if would be nice to have the mutants derived for the faults related to the system and its users, as well as the mutants derived for the specifications at different levels. It would be nice to know what is the relationship between such mutants, not to build a test suite as the union of those for various levels, but using some optimization strategies. We use *subjunctive mood* in here on purpose, as this task does not seem very feasible to the author at this stage for any SUT or any application area, but we started to work on it at least, when comparing traditional fault models against those defined for SDN components.

When it comes to the application areas, we always want to perform more experiments when having access to *real* data. We will try to do this taking advantage of the existing collaborations with the TSP colleagues, as well as our industrial partners. At the same time, we would not like to limit ourselves with dynamic or programmable networks and turn our attention or other distributed systems (as we partially did already with Cloud infrastructures, for example). There is a hope that *good* nondeterministic specifications can find their place in such case studies, and thus test generation strategies with the guaranteed fault coverage will be implementable in *real* time. All these aspects need to be investigated carefully and again it is an avenue for our future work.

Self-references

[BKY+21] Igor B. Burdonov, Alexandre Kossachev, Nina Yevtushenko, Jorge López, Natalia Kushik, and Djamal Zeghlache. Preventive model-based verification and repairing for SDN requests. In Raian Ali, Hermann Kaindl, and Leszek A. Maciaszek, editors, *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2021, Online Streaming, April 26-27, 2021*, pages 421–428. SCITEPRESS, 2021.

[BLK+18] Asma Berriri, Jorge López, Natalia Kushik, Nina Yevtushenko, and Djamal Zeghlache. Towards model based testing for software defined networks. In Ernesto Damiani, George Spanoudakis, and Leszek A. Maciaszek, editors, *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2018, Funchal, Madeira, Portugal, March 23-24, 2018*, pages 440–446. SciTePress, 2018.

[EYK18] Khaled El-Fakih, Nina Yevtushenko, and Natalia Kushik. Adaptive distinguishing test cases of nondeterministic finite state machines: test case derivation and length estimation. *Formal Aspects Comput.*, 30(2):319–332, 2018.

[KEY11] Natalia Kushik, Khaled El-Fakih, and Nina Yevtushenko. Preset and adaptive homing experiments for nondeterministic finite state machines. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *Implementation and Application of Automata - 16th International Conference, CIAA 2011, Blois, France, July 13-16, 2011. Proceedings*, volume 6807 of *Lecture Notes in Computer Science*, pages 215–224. Springer, 2011.

[KEYC16a] Natalia Kushik, Khaled El-Fakih, Nina Yevtushenko, and Ana R. Cavalli. On adaptive experiments for nondeterministic finite state machines. *Int. J. Softw. Tools Technol. Transf.*, 18(3):251–264, 2016.

[KEYC16b] Natalia Kushik, Khaled El-Fakih, Nina Yevtushenko, and Ana R. Cavalli. On adaptive experiments for nondeterministic finite state machines. *STTT*, 18(3):251–264, 2016.

[KKE14] Natalia G. Kushik, Victor V. Kulyamin, and Nina V. Evtushenko. On the complexity of existence of homing sequences for nondeterministic finite state machines. *Program. Comput. Softw.*, 40(6):333–336, 2014.

[KLCY16] Natalia Kushik, Jorge López, Ana R. Cavalli, and Nina Yevtushenko. Improving protocol passive testing through "gedanken" experiments with finite state machines. In *2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016, Vienna, Austria, August 1-3, 2016*, pages 315–322. IEEE, 2016.

[KLY16] N. G. Kushik, J. E. López, and N. V. Yevtushenko. Investigation of correlation of test sequences for reliability testing of digital physical system components. *Russian Physics Journal*, 59(8):1274–1280, Dec 2016.

[KPY+14] Natalia Kushik, Jeevan Pokhrel, Nina Yevtushenko, Ana R. Cavalli, and Wissam Mallouli. Qoe prediction for multimedia services: Comparing fuzzy and logic network approaches. *Int. J. Organ. Collect. Intell.*, 4(3):44–64, 2014.

[Kus13] Natalia Kushik. *Methods for deriving homing and distinguishing experiments for non-deterministic FSMs (in Russian)*. Phd thesis, Tomsk State University, 2013. 137 pages.

[Kus16] Natalia Kushik. *Methods for defining FSM classes with reduced complexity of 'gedanken' experiments (in Russian)*. Habilitation (doctor of sciences), Tomsk State University, 2016. 287 pages.

[KY13] Natalia Kushik and Nina Yevtushenko. On the length of homing sequences for nondeterministic finite state machines. In *Proceedings of International Conference on Implementation and Application of Automata (CIAA)*, pages 220–231, 2013.

[KY15a] Natalia Kushik and Hüsnü Yenigün. Heuristics for deriving adaptive homing and distinguishing sequences for nondeterministic finite state machines. In Khaled El-Fakih, Gerassimos D. Barlas, and Nina Yevtushenko, editors, *Testing Software and Systems - 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25,*

2015, Proceedings, volume 9447 of *Lecture Notes in Computer Science*, pages 243–248. Springer, 2015.

[KY15b] Natalia Kushik and Nina Yevtushenko. Adaptive homing is in P. In Nikolay V. Pakulin, Alexander K. Petrenko, and Bernd-Holger Schlingloff, editors, *Proceedings Tenth Workshop on Model Based Testing, MBT 2015, London, UK, 18th April 2015*, volume 180 of *EPTCS*, pages 73–78, 2015.

[KY15c] Natalia Kushik and Nina Yevtushenko. Describing homing and distinguishing sequences for nondeterministic finite state machines via synchronizing automata. In Frank Drewes, editor, *Implementation and Application of Automata - 20th International Conference, CIAA 2015, Umeå, Sweden, August 18-21, 2015, Proceedings*, volume 9223 of *Lecture Notes in Computer Science*, pages 188–198. Springer, 2015.

[KYBK18] Natalia Kushik, Nina Yevtushenko, Igor B. Bourdonov, and Alexander S. Kossatchev. Deriving synchronizing and homing sequences for input/output automata. *Autom. Control. Comput. Sci.*, 52(7):589–595, 2018.

[KYC14a] Natalia Kushik, Nina Yevtushenko, and Ana R. Cavalli. On testing against partial non-observable specifications. In *9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014, Guimaraes, Portugal, September 23-26, 2014*, pages 230–233. IEEE Computer Society, 2014.

[KYC+14b] Natalia Kushik, Nina Yevtushenko, Ana R. Cavalli, Wissam Mallouli, and Jeevan Pokhrel. Evaluating web service qoe by learning logic networks. In Valérie Monfort and Karl-Heinz Krempels, editors, *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 1, Barcelona, Spain, 3-5 April, 2014*, pages 168–176. SciTePress, 2014.

[KYL21] Natalia Kushik, Nina Yevtushenko, and Jorge López. Testing against nondeterministic fsms: a probabilistic approach for test suite minimization. In Ana Cavalli and Héctor D. Menéndez, editors, *Testing Software and Systems - 33rd IFIP WG 6.1 International Conference, ICTSS 2021, University College London, United Kingdom, 10-12 November 2021, Proceedings*, Lecture Notes in Computer Science. Springer, 2021.

[KYTS15] Natalia Kushik, Nina Yevtushenko, Stanislav N. Torgaev, and Nikita Shatilov. On using ABC for deriving distinguishing sequences for verilog-descriptions.

In *2015 IEEE East-West Design & Test Symposium, EWDTS 2015, Batumi, Georgia, September 26-29, 2015*, pages 1–4. IEEE Computer Society, 2015.

[KYY16] Natalia Kushik, Nina Yevtushenko, and Hüsnü Yenigün. Reducing the complexity of checking the existence and derivation of adaptive synchronizing experiments for nondeterministic fsms. In *Proceedings of the International Workshop on domAin specific Model-based AppRoaches to vErificaTion and validaTiOn, AMARETTO@MODELSWARD 2016, Rome, Italy, February 19-21, 2016.*, pages 83–90, 2016.

[LKB+18] Jorge López, Natalia Kushik, Asma Berriri, Nina Yevtushenko, and Djamal Zeghlache. Test derivation for sdn-enabled switches: A logic circuit based approach. In Inmaculada Medina-Bulo, Mercedes G. Merayo, and Robert M. Hierons, editors, *Testing Software and Systems - 30th IFIP WG 6.1 International Conference, ICTSS 2018, Cádiz, Spain, October 1-3, 2018, Proceedings*, volume 11146 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2018.

[LKYZ17] Jorge López, Natalia Kushik, Nina Yevtushenko, and Djamal Zeghlache. Analyzing and validating virtual network requests. In Jorge S. Cardoso, Leszek A. Maciaszek, Marten van Sinderen, and Enrique Cabello, editors, *Proceedings of the 12th International Conference on Software Technologies, ICSOFT 2017, Madrid, Spain, July 24-26, 2017*, pages 441–446. SciTePress, 2017.

[LKZ17] Jorge López, Natalia Kushik, and Djamal Zeghlache. Quality estimation of virtual machine placement in cloud infrastructures. In Nina Yevtushenko, Ana Rosa Cavalli, and Hüsnü Yenigün, editors, *Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings*, volume 10533 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2017.

[LKZ19] Jorge López, Natalia Kushik, and Djamal Zeghlache. Virtual machine placement quality estimation in cloud infrastructures using integer linear programming. *Softw. Qual. J.*, 27(2):731–755, 2019.

[LLK+18] Jorge López, Andrey Laputenko, Natalia Kushik, Nina Yevtushenko, and Stanislav N. Torgaev. Scalable supervised machine learning apparatus for computationally constrained devices. In Leszek A. Maciaszek and Marten van Sinderen, editors, *Proceedings of the 13th International Conference on Soft-*

ware Technologies, ICSOFT 2018, Porto, Portugal, July 26-28, 2018, pages 552–562. SciTePress, 2018.

[PLK+20] Erick Petersen, Jorge López, Natalia Kushik, Claude Poletti, and Djamal Zeghlache. On using smt-solvers for modeling and verifying dynamic network emulators: (work in progress). In *19th IEEE International Symposium on Network Computing and Applications, NCA 2020, Cambridge, MA, USA, November 24-27, 2020*, pages 1–3. IEEE, 2020.

[SLK+16] Sergey A. Smolov, Jorge López, Natalia Kushik, Nina Yevtushenko, Mikhail M. Chupilko, and Alexander S. Kamkin. Testing logic circuits at different abstraction levels: An experimental evaluation. In *2016 IEEE East-West Design & Test Symposium, EWDTS 2016, Yerevan, Armenia, October 14-17, 2016*, pages 1–4. IEEE Computer Society, 2016.

[TWJ+21] Kuan-Hua Tu, Hung-En Wang, Jie-Hong Roland Jiang, Natalia Kushik, and Nina Yevtushenko. Homing sequence derivation with quantified boolean satisfiability. *IEEE Transactions on Computers*, pages 696–711, 2021.

[VLK+19] Evgenii Vinarskii, Jorge López, Natalia Kushik, Nina Yevtushenko, and Djamal Zeghlache. A model checking based approach for detecting SDN races. In Christophe Gaston, Nikolai Kosmatov, and Pascale Le Gall, editors, *Testing Software and Systems - 31st IFIP WG 6.1 International Conference, ICTSS 2019, Paris, France, October 15-17, 2019, Proceedings*, volume 11812 of *Lecture Notes in Computer Science*, pages 194–211. Springer, 2019.

[WTJK17] Hung-En Wang, Kuan-Hua Tu, Jie-Hong R. Jiang, and Natalia Kushik. Homing sequence derivation with quantified boolean satisfiability. In Nina Yevtushenko, Ana Rosa Cavalli, and Hüsnü Yenigün, editors, *Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings*, volume 10533 of *Lecture Notes in Computer Science*, pages 230–242. Springer, 2017.

[YBK+18] Nina Yevtushenko, Igor B. Burdonov, Alexandre Kossachev, Jorge López, Natalia Kushik, and Djamal Zeghlache. Test derivation for the software defined networking platforms: Novel fault models and test completeness. In *2018 IEEE East-West Design & Test Symposium, EWDTS 2018, Kazan, Russia, September 14-17, 2018*, pages 1–6. IEEE, 2018.

[YK15] Nina Yevtushenko and Natalia Kushik. Decreasing the length of adaptive distinguishing experiments for nondeterministic merging-free finite state ma-

chines. In *2015 IEEE East-West Design & Test Symposium, EWDTS 2015, Batumi, Georgia, September 26-29, 2015*, pages 1–4. IEEE Computer Society, 2015.

[YKK19] Nina Yevtushenko, Victor V. Kuliamin, and Natalia Kushik. Evaluating the complexity of deriving adaptive homing, synchronizing and distinguishing sequences for nondeterministic fsms. In Christophe Gaston, Nikolai Kosmatov, and Pascale Le Gall, editors, *Testing Software and Systems - 31st IFIP WG 6.1 International Conference, ICTSS 2019, Paris, France, October 15-17, 2019, Proceedings*, volume 11812 of *Lecture Notes in Computer Science*, pages 86–103. Springer, 2019.

[YKK21] Nina Yevtushenko, Victor Kuliamin, and Natalia Kushik. Evaluating the complexity of deriving adaptive s'-homing and s'-synchronizing sequences for nondeterministic fsms. *Software Quality Journal*, 2021.

[YKL+17] Hüsnü Yenigün, Natalia Kushik, Jorge López, Nina Yevtushenko, and Ana R. Cavalli. Decreasing the complexity of deriving test suites against nondeterministic finite state machines. In *2017 IEEE East-West Design & Test Symposium, EWDTS 2017, Novi Sad, Serbia, September 29 - October 2, 2017*, pages 1–4. IEEE Computer Society, 2017.

[YYK16] Hüsnü Yenigün, Nina Yevtushenko, and Natalia Kushik. Some classes of finite state machines with polynomial length of distinguishing test cases. In Sascha Ossowski, editor, *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, pages 1680–1685. ACM, 2016.

[YYK17] Hüsnü Yenigün, Nina Yevtushenko, and Natalia Kushik. The complexity of checking the existence and derivation of adaptive synchronizing experiments for deterministic fsms. *Inf. Process. Lett.*, 127:49–53, 2017.

Bibliography

[ACC+04] Baptiste Alcalde, Ana R. Cavalli, Dongluo Chen, Davy Khuu, and David Lee. Network protocol system passive testing for fault management: A backward checking approach. In David de Frutos-Escrig and Manuel Núñez, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2004, 24th IFIP WG 6.1 International Conference, Madrid Spain, September 27-30, 2004, Proceedings*, volume 3235 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2004.

[ACY95] Rajeev Alur, Costas Courcoubetis, and Mihalis Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 363–372, 1995.

[BEK+14] Roderick Bloem, Uwe Egly, Patrick Klampfl, Robert Konighofer, and Florian Lonsing. Sat-based methods for circuit synthesis. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pages 31–34, 2014.

[BEM12] Robert Brayton, Niklas Een, and Alan Mishchenko. Using speculation for sequential equivalence checking. In *Proceedings of the International Workshop on Logic and Synthesis, IWLS 2012*, pages 139–145, 2012.

[Ber19] Asma Berriri. *Model based testing techniques for software defined networks. (Méthodes de test basées sur les modèles pour la validation des réseaux logiciels (SDN))*. PhD thesis, University of Paris-Saclay, France, 2019.

[BKS14] Roderick Bloem, Robert Könighofer, and Martina Seidl. Sat-based synthesis methods for safety specs. In Kenneth L. McMillan and Xavier Rival, editors, *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, volume 8318 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2014.

[BM10] Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 24–40, 2010.

[BT09] Alexander Barkalov and Larysa Titarenko. *Logic Synthesis for FSM-Based Control Units*, volume 53 of *Lecture Notes in Electrical Engineering*. Springer, 2009.

[Car18] Pamela Carvallo. *Security in the Cloud: an anomaly-based detection framework for the insider threats. (Sécurité dans le cloud: framework de détection de menaces internes basé sur l'analyse d'anomalies)*. PhD thesis, University of Paris-Saclay, France, 2018.

[CGP03] Ana R. Cavalli, Caroline Gervy, and Svetlana Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Inf. Softw. Technol.*, 45(12):837–852, 2003.

[Cho78] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978.

[ÇKY+18] Berk Çirisci, Muhammed Kerem Kahraman, Cagri Uluc Yildirimoglu, Kamer Kaya, and Hüsnü Yenigün. Using structure of automata for faster synchronizing heuristics. In Slimane Hammoudi, Luís Ferreira Pires, and Bran Selic, editors, *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 544–551. SciTePress, 2018.

[CMBK07] Satrajit Chatterjee, Alan Mishchenko, Robert K. Brayton, and Andreas Kuehlmann. On resolution proofs for combinational equivalence. In *Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007*, pages 600–605. IEEE, 2007.

[CMDO09] Ana Cavalli, Stephane Maag, and Edgardo Montes De Oca. A passive conformance testing approach for a manet routing protocol. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 207–211. ACM, 2009.

[CRM12] Xiaoping Che, Felipe Lalanne Rojas, and Stephane Maag. A logic-based passive testing approach for the validation of communicating protocols. In

ENASE'12: 7th International Conference on Evaluation of Novel Approaches to Software Engineering, 2012.

[CRS00] F. Corno, M.S. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *IEEE Design Test of Computers*, 17(3):44–53, 2000.

[DEM+10] Rita Dorofeeva, Khaled El-Fakih, Stéphane Maag, Ana R. Cavalli, and Nina Yevtushenko. Fsm-based conformance testing methods: A survey annotated with experimental evaluation. *Inf. Softw. Technol.*, 52(12):1286–1297, 2010.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DMBSV85] G. De Micheli, R.K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(3):269–285, 1985.

[DMT16] Palma D., Rutkowski M., and Spatzier T. Tosca simple profile in yaml version 1.0. oasis committee specification 01. `http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd02/TOSCA-Simple-Profile-YAML-v1.0-csprd02.html`, 2016.

[DSO14] Lebrun David, Vissicchio Stefano, and Bonaventure Olivier. Towards test-driven software defined networking. In *2014 IEEE Network Operations and Management Symposium*, pages 1–9, 2014.

[EDYvB12] Khaled El-Fakih, Rita Dorofeeva, Nina Yevtushenko, and Gregor von Bochmann. Fsm-based testing from user defined faults adapted to incremental and mutation testing. *Program. Comput. Softw.*, 38(4):201–209, 2012.

[EGGC09] Jose Pablo Escobedo, Christophe Gaston, Pascale Le Gall, and Ana R. Cavalli. Observability and controllability issues in conformance testing of web service compositions. In Manuel Núñez, Paul Baker, and Mercedes G. Merayo, editors, *Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4, 2009. Proceedings*, volume 5826 of *Lecture Notes in Computer Science*, pages 217–222. Springer, 2009.

[EMB+16] Ahmed El-Hassany, Jeremie Miserez, Pavol Bielik, Laurent Vanbever, and Martin T. Vechev. Sdnracer: concurrency analysis for software-defined networks. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 402–415, 2016.

[Epp90] David Eppstein. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3):500–510, 1990.

[EPYvB03] Khaled El-Fakih, Svetlana Prokopenko, Nina Yevtushenko, and Gregor von Bochmann. Fault diagnosis in extended finite state machines. In Dieter Hogrefe and Anthony Wiles, editors, *Testing of Communicating Systems, 15th IFIP International Conference, TestCom 2003, Sophia Antipolis, France, May 26-28, 2003, Proceedings*, volume 2644 of *Lecture Notes in Computer Science*, pages 197–210. Springer, 2003.

[ETSY04] Khaled El-Fakih, Vadim Trenkaev, Natalia Spitsyna, and Nina Yevtushenko. FSM based interoperability testing methods for multi stimuli model. In *Testing of Communicating Systems, 16th IFIP International Conference, TestCom 2004, Oxford, UK, March 17-19, 2004, Proceedings*, pages 60–75, 2004.

[EYF09] Khaled El-Fakih, Nina Yevtushenko, and Hacène Fouchal. Testing timed finite state machines with guaranteed fault coverage. In Manuel Núñez, Paul Baker, and Mercedes G. Merayo, editors, *Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4, 2009. Proceedings*, volume 5826 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2009.

[FvBK+91] Susumu Fujiwara, Gregor von Bochmann, Ferhat Khendek, Mokhtar Amalou, and Abderrazak Ghedamsi. Test selection based on finite state models. *IEEE Trans. Software Eng.*, 17(6):591–603, 1991.

[FWA09] Gordon Fraser, Franz Wotawa, and Paul Ammann. Testing with model checkers: a survey. *Softw. Test. Verification Reliab.*, 19(3):215–261, 2009.

[FYT+16] Seyed Kaveh Fayaz, Tianlong Yu, Yoshiaki Tobioka, Sagar Chaki, and Vyas Sekar. BUZZ: testing context-dependent policies in stateful networks. In *13th USENIX Symposium on Networked Systems Design and Implementation*, pages 275–289, 2016.

[Gil62] A. Gill. *Introduction to the theory of finite-state machines*. McGraw Hill, 1962.

[GvBD93] Abderrazak Ghedamsi, Gregor von Bochmann, and Rachida Dssouli. Multiple fault diagnostics for finite state machines. In *Proceedings IEEE INFOCOM '93, The Conference on Computer Communications, Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies, Networking: Foundation for the Future, San Francisco, CA, USA, March 28 - April 1, 1993*, pages 782–791. IEEE Computer Society, 1993.

[Hen64] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proceedings of Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, 1964.

[HHL+91] K.C. Huang, W.S. Hsieh, C.S. Lu, M.S. Yang, T.S. Nain, and Ihnen Lin. Implementation and design of pvd: An interactive protocol specification and validation environment. *Microprocessing and Microprogramming*, 32(1):281–288, 1991. Euromicro symposium on microprocessing and microprogramming.

[HM09] Robert M. Hierons and Mercedes G. Merayo. Mutation testing from probabilistic and stochastic finite state machines. *J. Syst. Softw.*, 82(11):1804–1818, 2009.

[Hol82] Gerard J. Holzmann. A theory for protocol validation. *IEEE Trans. Computers*, 31(8):730–738, 1982.

[Hol03] Gerard Holzmann. *The Spin model checker: primer and reference manual*. Addison-Wesley Professional, 2003.

[HYC12] Iksoon Hwang, Nina Yevtushenko, and Ana R. Cavalli. Tight bound on the length of distinguishing sequences for non-observable nondeterministic finite-state machines with a polynomial number of inputs and outputs. *Inf. Process. Lett.*, 112(7):298–301, 2012.

[IAS15] Melnichenko I., Kamkin A., and Smolov S. An extended finite state machine-based approach to code coverage-directed test generation for hardware designs. *Proceedings of ISP RAS*, 27:161–182, 2015.

[IS04] Masami Ito and Kayoko Shikishima-Tsuji. Some results on directable automata. In *Theory Is Forever, Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*, pages 125–133, 2004.

[JL93] Ajin Jirachiefpattana and Richard Lai. Verifying estelle specifications: numerical petri nets approach. In *1993 International Conference on Network Protocols, ICNP 1993, San Francisco, CA, USA, October 19-22, 1993, Proceedings*, pages 334–341. IEEE Computer Society, 1993.

[JVCH10] Jie-Hong Roland Jiang, Tiziano Villa, Yves Crama, and Peter L. Hammer. Hardware equivalence and property verification. In Yves Crama and Peter L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 599–674. Cambridge University Press, 2010.

[Kön12] Hartmut König. *Protocol Engineering*. Springer, 2012.

[KPKR95] S. Kajihara, I. Pomeranz, K. Kinoshita, and S.M. Reddy. Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(12):1496–1504, 1995.

[KPY99] I. Koufareva, Alexandre Petrenko, and Nina Yevtushenko. Test generation driven by user-defined fault models. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *Testing of Communicating Systems: Method and Applications, IFIP TC6 12th International Workshop on Testing Communicating Systems, September 1-3, 1999, Budapest, Hungary*, volume 147 of *IFIP Conference Proceedings*, pages 215–236. Kluwer, 1999.

[LB10] Florian Lonsing and Armin Biere. DepQBF: A dependency-aware QBF solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):71–76, 2010.

[LMM16] Jorge López, Stéphane Maag, and Gerardo Morales. Behavior evaluation for trust management based on formal distributed network monitoring. *World Wide Web*, 19(1):21–39, 2016.

[LNS+97] David Lee, Arun N. Netravali, Krishan K. Sabnani, Binay Sugla, and Ajita John. Passive testing and applications to network management. In *1997 International Conference on Network Protocols (ICNP '97), 28-31 October 1997, Atlanta, GA, USA*, page 113. IEEE Computer Society, 1997.

[Lop19] Jorge Lopez. FSMHSGen. `https://github.com/jorgelopezcoronado/FSMHSGen`, 2019.

[LSKP96] D. Lee, K.K. Sabnani, D.M. Kristol, and S. Paul. Conformance testing of protocols specified as communicating finite state machines-a guided random walk based approach. *IEEE Transactions on Communications*, 44(5):631–640, 1996.

[LY94] D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[MA01] R.E. Miller and K.A. Arisha. Fault identification in networks by passive testing. In *Proceedings of the 34th Annual Simulation Symposium*, pages 277–284, 2001.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[Mar14] Pavel Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory Comput. Syst.*, 54(2):293–304, 2014.

[MBE+15] Jeremie Miserez, Pavol Bielik, Ahmed El-Hassany, Laurent Vanbever, and Martin T. Vechev. Sdnracer: detecting concurrency violations in software-defined networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15, Santa Clara, California, USA, June 17-18, 2015*, pages 22:1–22:7, 2015.

[MCBE06] Alan Mishchenko, Satrajit Chatterjee, Robert K. Brayton, and Niklas Eén. Improvements to combinational equivalence checking. In Soha Hassoun, editor, *2006 International Conference on Computer-Aided Design, ICCAD 2006, San Jose, CA, USA, November 5-9, 2006*, pages 836–843. ACM, 2006.

[MHC17] Jedidiah McClurg, Hossein Hojjat, and Pavol Cerný. Synchronization synthesis for network programs. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, pages 301–321, 2017.

[MHN18] Mercedes G. Merayo, Robert M. Hierons, and Manuel Núñez. Passive testing with asynchronous communications and timestamps. *Distributed Comput.*, 31(5):327–342, 2018.

[Mil98] R.E. Miller. Passive testing of networks using a cfsm specification. In *1998 IEEE International Performance, Computing and Communications Conference. Proceedings (Cat. No.98CH36191)*, pages 111–116, 1998.

[MMS15] Anzhela Yu. Matrosova, Eugeniy Mitrofanov, and Toral Shah. Multiple stuck-at fault testability of a combinational circuit derived by covering ROBDD nodes by invert-and-or sub-circuits. In *2015 IEEE East-West Design & Test Symposium, EWDTS 2015, Batumi, Georgia, September 26-29, 2015*, pages 1–4. IEEE Computer Society, 2015.

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. In *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[Nat86] B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Proceedings of Symposium on Foundations of Computer Science (SFCS)*, pages 132–142, 1986.

[Nij] Radboud University Nijmegen. Automata wiki. `https://automata.cs.ru.nl/BenchmarkCircuits/Kiss`.

[NPR18] Omer Nguena-Timo, Alexandre Petrenko, and S. Ramesh. Checking sequence generation for symbolic input/output FSMs by constraint solving. In *Proceedings of International Colloquium on Theoretical Aspects of Computing (ICTAC)*, pages 354–375, 2018.

[PAGO19] Alexandre Petrenko, Florent Avellaneda, Roland Groz, and Catherine Oriat. FSM inference and checking sequence construction are two sides of the same coin. *Softw. Qual. J.*, 27(2):651–674, 2019.

[Pat05] Janak H. Patel. Stuck-at fault: A fault model for the next millennium? `http://web.stanford.edu/class/ee386/public/stuck_at_fault_6per_page`, 2005.

[PB15] Fabio Lopez Pires and Benjamín Barán. Virtual machine placement literature review. *CoRR*, abs/1506.01509, 2015.

[PBG04] Alexandre Petrenko, Sergiy Boroday, and Roland Groz. Confirming configurations in EFSM testing. *IEEE Trans. Software Eng.*, 30(1):29–42, 2004.

[Pix92] Carl Pixley. A theory and implementation of sequential hardware equivalence. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 11(12):1469–1478, 1992.

[PM64] J. F. Poage and E. J. McCluskey. Derivation of optimum test sequencies for sequential machines. In *1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 121–132, 1964.

[PNR16] Alexandre Petrenko, Omer Nguena-Timo, and S. Ramesh. Multiple mutation testing from FSM. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2016.

[PPK+15] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130. USENIX Association, 2015.

[PY05] Alexandre Petrenko and Nina Yevtushenko. Conformance tests as checking experiments for partial nondeterministic FSM. In *Formal Approaches to Software Testing, 5th International Workshop, FATES 2005, Edinburgh, UK, July 11, 2005, Revised Selected Papers*, pages 118–133, 2005.

[PY11] Alexandre Petrenko and Nina Yevtushenko. Adaptive testing of deterministic implementations specified by nondeterministic fsms. In Burkhart Wolff and Fatiha Zaïdi, editors, *Testing Software and Systems*, pages 162–178, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[PYvB96] Alexandre Petrenko, Nina Yevtushenko, and Gregor von Bochmann. Fault models for testing in context. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX: Theory, application and tools, IFIP TC6 WG6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI, Kaiserslautern, Germany,*

8-11 October 1996, volume 69 of *IFIP Conference Proceedings*, pages 163–178. Chapman & Hall, 1996.

[RT15] M. N. Rabe and L. Tentrup. CAQE: A certifying QBF solver. In *Proceedings of International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 136–143, 2015.

[San05] Sven Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems: Advanced Lectures*, pages 5–33. Springer Berlin Heidelberg, 2005.

[Sas93] (Editor) Tsutomu Sasao. *Logic Synthesis and Optimization*. The Kluwer International Series in Engineering and Computer Science. Springer, Boston, MA, 1993.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[SD88] Krishan Sabnani and Anton Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285–297, 1988.

[SEY07] Natalia Spitsyna, Khaled El-Fakih, and Nina Yevtushenko. Studying the separability relation between finite state machines. *Softw. Test. Verification Reliab.*, 17(4):227–241, 2007.

[SV18] Hanan Shabana and Mikhail V. Volkov. Using sat solvers for synchronization issues in non-deterministic automata. *CoRR*, abs/1801.05391, 2018.

[SV19] Hanan Shabana and Mikhail V. Volkov. Using SAT solvers for synchronization issues in partial deterministic automata. In *Proceedings of International Conference on Mathematical Optimization Theory and Operations Research (MOTOR)*, pages 103–118, 2019.

[TB73] B.A. Trakhtenbrot and Ya.M. Barzdin'. *Finite Automata: Behavior and Synthesis*. North-Holland, Amsterdam, 1973.

[TEY18] Aleksandr Tvardovskii, Khaled El-Fakih, and Nina Yevtushenko. Deriving tests with guaranteed fault coverage for finite state machines with timeouts. In Inmaculada Medina-Bulo, Mercedes G. Merayo, and Robert M. Hierons, editors, *Testing Software and Systems - 30th IFIP WG 6.1 International Conference, ICTSS 2018, Cádiz, Spain, October 1-3, 2018, Proceedings*, volume 11146 of *Lecture Notes in Computer Science*, pages 149–154. Springer, 2018.

[Tra20] A. N. Trahtman. Cerny-starke conjecture from the sixties of XX century. *CoRR*, abs/2003.06177, 2020.

[Tre96] Jan Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Softw. Concepts Tools*, 17(3):103–120, 1996.

[Tri16] Stavros Tripakis. Fundamental algorithms for system modeling, analysis, and optimization. `https://ptolemy.berkeley.edu/projects/embedded/eecsx44/lectures/02-discrete-systems.pdf`, 2016.

[TY20] A. S. Tvardovskii and N. V. Yevtushenko. Deriving homing sequences for finite state machines with timed guards. *Model. Anal. Inform. Sist.*, 27(4):376–395, 2020.

[Vas73] M. P. Vasilevskii. Failure diagnosis of automata. *Cybernetics*, 9(4):653–665, 1973.

[vB75] Gregor von Bochmann. Communication protocols and error recovery procedures. *ACM SIGOPS Oper. Syst. Rev.*, 9(3):45–50, 1975.

[vBP94] Gregor von Bochmann and Alexandre Petrenko. Protocol testing: Review of methods and relevance for software testing. In Thomas J. Ostrand, editor, *Proceedings of the 1994 International Symposium on Software Testing and Analysis, ISSTA 1994, Seattle, WA, USA, August 17-19, 1994*, pages 109–124. ACM, 1994.

[VKBS97] Tiziano Villa, Timothy Kam, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Explicit and implicit algorithms for binate covering problems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 16(7):677–691, 1997.

[VKBSV97] Tiziano Villa, Timothy Kam, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. *Synthesis of Finite State Machines*. Springer, 1997.

[Vol08] Mikhail V. Volkov. Synchronizing automata and the cerny conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008.

[VTdlH+05] Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite-state machines-part I. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1013–1025, 2005.

[VYB+12] T. Villa, N. Yevtushenko, R. Brayton, A. Mishchenko, A. Petrenko, and A. Sangiovanni-Vincentelli. *The Unknown Component Problem - Theory and Applications*, chapter 2, 3, pages 9–72. Springer, 2012.

[WGL+16] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Software Eng.*, 42(8):707–740, 2016.

[ZJM21] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko. A circuit-based sat solver for logic synthesis. In *Proceedings of the 30th International Workshop on Logic & Synthesis (IWLS)*, 2021.

[ZKVM12] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, pages 241–252, 2012.