Online Testing of Nondeterministic Systems with the Reactive Planning Tester

Jüri Vain, Marko Kääramees, and Maili Markvardt
Department of Computer Science,
Tallinn University of Technology, Estonia
E-mail: vain@ioc.ee

ABSTRACT

We describe a model-based construction of an online tester for black-box testing. Contemporary model-based online test generators focusing mainly on computationally cheap but far from optimal planning strategies cover just a fraction of the wide spectrum of test control strategies. Typical examples of those used are simple random choice and anti-ant. Exhaustive planning during online testing of nondeterministic systems looks out of reach because of the low scalability of the methods in regard to the model size. The reactive planning tester (RPT) studied in the chapter is targeted to fill the gap between these two extremes. The key idea of RPT lies in offline static analysis of the IUT model to prepare the data and constraints for efficient online reactive planning. The external behavior of the IUT is modelled as an output observable nondeterministic EFSM with the assumption that all transition paths are feasible. A test purpose is attributed to the transitions of the IUT model by a set of Boolean variables called traps that are used to measure the progress of the test run. We present a way to construct a tester that at runtime selects a suboptimal test path from trap to trap by finding the shortest path that covers unvisited traps within planning horizon. The principles of reactive planning are implemented in the form of the decision rules of selecting the shortest paths at runtime. Based on an industrial scale case study, namely the city lighting system controller, we demonstrate the practical use of the RPT for systems with high degree of nondeterminism, deep nested control loops, and requiring strictly bounded tester response time. Tuning the planning horizon of the RPT allows a trade-off to be found between close to optimal test length and scalability of tester behavior with computationally feasible expenses.

1 INTRODUCTION

Model-Based Testing is the automatic generation of efficient test procedures/vectors using models of system requirements and specified functionality. Specific activities of the practice are (1) Build the model, (2) Generate expected inputs (3) Generate expected outputs, (4) Run tests, (5) Compare actual outputs with expected outputs, and (6) Decide on further actions (whether to modify the model, generate more tests, or stop testing, estimate reliability (quality) of the software (DACS Gold Practice Website, 2010).

1.1 On-line Testing

On-line testing is widely considered to be the most appropriate technique for model-based testing (MBT) of embedded systems where the implementation under test (IUT) is modelled using nondeterministic models (Veanes, Campbell, & Schulte, 2007; Veanes, Campbell, Grieskamp, Schulte, Tillmann, & Nachmanson, 2008). Nondeterminism of IUT models stems from the physical nature of the IUT, particularly, its internal parallel processes, timing conditions, and hardware-related asynchrony of executing the processes. Other sources of model nondeterminism are the higher abstraction level of the model compared to IUT implementation and the ambiguities in the specifications of the IUT. Often, the term *on-the-fly* is used in the context of on-line testing to describe the test generation and execution algorithms that compute and send successive stimuli to IUT incrementally at runtime. Computation of test stimuli is directed by the test purpose and the observed outputs of the IUT.

The state-space explosion problem experienced by many model-based offline test generation methods is avoided by the on-line techniques because only a limited part of the state-space needs to be kept track of at any point in time when a test is running. However, exhaustive planning would be difficult on-the-fly because of the limitations of available computational resources at the time of test execution. Thus, developing a planning strategy for industrial strength online testing should address in the first place the trade-off between reaction time and on-line planning depth to reach the practically feasible test cases.

The simplest approach to on-the-fly selection of test stimuli in model-based on-line testing is to apply so called random walk strategy where no computation sequence of IUT has an advantage over the others. The test is performed usually to discover violations of input/output conformance relation IOCO (Tretmans, 1999) or timed input/output conformance relation TIOCO (Brinksma & Tretmants, 2001) between the IUT and its model. Random exploration of the state space may lead to test cases that are unreasonably long and nevertheless may leave the test purpose unachieved. On the other hand, the long test cases are not completely useless, some unexpected and intricate bugs that do not fit under well-defined test coverage criteria can be detected when a test runs hours or even days.

In order to overcome the deficiencies of long lasting testing usually additional heuristics, e.g. “anti-ant” (Li & Lam, 2005; Veanes, Roy, & Cambell, 2006), dynamic approach of DART system (Godefroid, Halleux, Nori, Rajamani, Schulte, Tillmann, & Levin, 2008), inserted assertions (Korel & Al-Yami, 1996), path fitness (Derderian, Hierons, Harman, & Guo, 2010), etc. are applied for guiding the exploration of the IUT state space. The extreme of guiding the selection of test stimuli is exhaustive planning by solving at each test execution step a full constraint system set by the test purpose and test planning strategy. For instance, the witness trace generated by model checking provides possibly optimal selection of the next test stimulus. The critical issue in the case of explicit state model checking algorithms is the size and complexity of the model leading to the explosion of the state space, specially in cases such as “door lock behavior“ or deep nested loops in the model (Hamon, Moura, & Rushby, 2004). Therefore, model checking based approaches are used mostly in offline test generation.

In this chapter we introduce the principle of reactive planning for on-the-fly selection of test stimuli and the reactive planning tester (RPT) synthesis algorithm for offline construction of those selection rules.The RPT synthesis algorithm assumes that the IUT model is presented as an output observable nondeterministic state machine (Luo, Bochmann, & Petrenko, 1994; Starke, 1972). At first, the synthesis method is introduced for extended finite state machine (EFSM) models of IUT in which all transition sequences are feasible and the EFSM can be transformed to ordinary finite state machine (FSM) model. In (Duale & Uyar, 2004; Hierons, Kim, & Ural, 2004) it has been shown how to transform an EFSM to one that has no infeasible paths. This has been achieved only for EFSMs in which all variable updates and transition guards are linear. In general, the problem of determining whether a path in an EFSM is feasible is undecidable. Therefore, we limit our approach to EFSMs which have only linear updates and transition guards. Later on, in Section 3.4, the synthesis algorithm will be generalised to tackle with EFSM models without the paths feasibility constraint. As will be shown using experimental results in the end of the chapter the reactive planning paradigm appears to be a practical trade-off between using simple heuristics and exhaustive planning in on-line model-based testing.

2 PRELIMINARIES OF ON-LINE TESTING WITH MODEL-BASED PLANNING

2.1 Reactive Model-Based Planning in Testing

The concept of a *reactive planning* as presented in (Williams & Nayak, 1997) is motivated by the need for model-based autonomy in applications which must cope with highly dynamic and unpredictable environments. Reactive planning operates in a timely fashion and is applicable in agents operating in these conditions (Lyons & Hendriks, 1992). Reactiveness of on-line testing means that tester program has to react to observed outputs of the IUT and to possible changes in the test goals on-the-fly. It tries to take the system towards the state that satisfies the desired test goals. Like generally in reactive planning, the model-based test executive uses a formal specification of the system to determine the desired state sequence in three stages - mode identification (MI), mode reconfiguration (MR) and model-based reactive planning (MRP) (Williams & Nayak, 1997). MI and MR set the planning problem, identifying initial and target states, while MRP reactively generates a plan soluion. MI is a phase where the current state of the system model is identified. In the case of a deterministic model transition, MI is trivial, it is just the next state reachable by applying the right IUT input. In the nondeterministic case, MI can determine the current state by looking at the output of the system provided the output is observable. In the current approach, the MR and the MRP phases are combined into one since both the goal and the next step toward the goal are determined by the same decision procedure as will be explained in detail in Section 3.5.1. Selection of IUT inputs taking closer to satisfying the test goal is based on the cost of applying a given input. Further, we characterize this cost using the so called gain function.

The rationale behind the reactive planning method proposed in this approach lies in combining computationally hard offline planning with time bounded online planning phases. Off-line phase is meant to shift the combinatorially hard planning as much as possible in test preparation phase where the results of static analysis of given IUT model and the test goal are recorded in the format of compact planning rules that are easy to apply later in on-line phase. While the reactive planning tester is synthesised, the rules are encoded in the tester model and applied when the test is running. Thus, the rules synthesized must ensure also proper termination of the test case when a prescribed test purpose is satisfied.

2.2 Model-Based Testing with EFSMs

In this approach, we assume that the IUT model is represented as output observable EFSM. A test purpose (or goal) is a specific objective or a property of the IUT that the tester is set out to test. Test purpose is specified in terms of test coverage items. We focus on test purposes that can be defined as a set of “traps” associated with the transitions of the IUT model (Hamon, Moura, Rushby, 2004). The goal of the tester is to generate a test sequence so that all traps are visited at least once during the test run.

The proposed tester synthesis method outputs also the tester model as EFSM where the rules for online planning are encoded in the transition guards as a conjuncts called *gain guard*. The gain guard evaluates **true** or **false** at the time of the execution of the tester determining if the transition can be taken from the current state or not. The value **true** means that taking the transition with the highest gain is the best possible choice to reach some unvisited traps from the current state. Since at each execution step of the tester model only the guards associated with the outgoing transitions of the current state are evaluated, the number of guard conditions to be evaluated at once is relatively small. To implement such a gain guided model traversal, the gain guard is defined using (model and goal specific) *gain functions* and the standard function *max* over the gain function values. The gain functions define the gain that is a quantitative measure needed to compare alternative choices of test stimuli on-the-fly. For each transition of the tester model that generates a stimulus, that can be chosen by test executive, a non-negative gain function is defined that depends on the current bindings of the EFSM context variables. The gain function of a transition defines a value that depends on the distance-weighted reachability of the unvisited traps from the given transition. The gain guard of the tester’s model transition is **true** if and only if that transition is a prefix of the test sequence with highest gain among those that depart from the current state. If gain functions of several enabled transitions evaluate to the same maximum value the tester selects one of these transitions using either random selection or “least visited first” principle. Each transition in the model is considered to have a weight and the cost of test case is proportional to the length of whole test sequence. Also, the current value (**true** when visited, otherwise **false**) of each trap is taken into account in gain functions.

2.3 Extended Finite State Machine

The synthesis of the RPT-tester is based on a non-deterministic EFSM model of the IUT.

Definition 1: An *extended finite state machine,* M is defined as a tuple (S, V, I, O, E), where S is a finite set of states, $s_0 \in S$ is an initial state, V is a finite set of variables with finite value domains, I is the finite set of inputs, O is the finite set of outputs, and E is the set of transitions. A configuration of M is a pair (s, σ) where $s \in S$ and $\sigma \in \Sigma$ is a mapping from V to values, and Σ is a finite set of mappings from variable names to their possible values. The initial configuration is (s_0, σ_0), where $\sigma_0 \in \Sigma$ is the initial assignment. A transition $e \in E$ is a tuple $e = (s, p, a, o, u, q)$, where s is the source state of the transition, q is the target state of the transition $(s, q \in S)$, p is a transition guard that is a logic formula over V, a is the input of M $(a \in I)$, o is the output of M $(o \in O)$, and u is an update function over V.

A deterministic EFSM is an EFSM where the output and next state are unambiguously determined by the current state and the input. A nondeterministic EFSM may contain states where the reaction of the EFSM in response to an input is nondeterministic, i.e. there are more than one outgoing transitions that are enabled simultaneously.

2.4 Modelling the IUT

Denote the EFSM model of IUT by M_S. It can be either deterministic or nondeterministic, it can be strongly connected or not. If the model is not strongly connected then we assume that there exists a reliable reset that allows the IUT to be taken back to the initial state from any state. Since there exists a transformation from EFSM to FSM (Henniger, Ulrich, & König, 1995) for EFSM models were the variables have finite, countable domains, we present the further details of RPT synthesis method at first using simpler FSM model notation. In practice this leads to a vast state space if we use FSMs even for a small system. Though, as will be demonstrated in Section 3.6 by means of RPT and adjustable planning horizon a FSM based test synthesis method can also scale well to handle industrial size testing problems. The transformation from EFSM to FSM is automatic in the method implementation (TestCast Generator Website, 2010) and hidden from the user.

It is essential that the tester can observe the outputs of the IUT for detecting the next state after a nondeterministic transition of the IUT. Therefore, we require that a nondeterministic IUT is output observable (Luo, Bochmann, & Petrenko, 1994; Starke, 1972) which means that even though there may be multiple transitions taken in response to a given input, the output identifies the next state of IUT unambiguously. An example of an output observable nondeterministic IUT model is given in Figure 1. The outgoing transitions e_0 and e_1 $(e_3$ and $e_4)$ of the state s_1 (s_2) have the same input a_0 (a_3), but different outputs o_0 or o_1 $(o_3$ or $o_4)$.

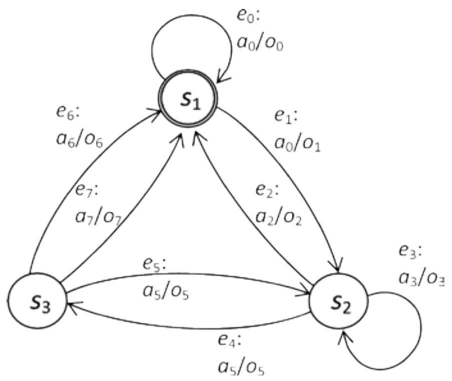


Figure 1: An output observable nondeterministic IUT model

2.5 Modelling the Test Purpose

A test purpose is a specific objective or a property of the IUT that the tester is set out to test. In general, test purposes are selected based on the correctness criteria stipulated by the specification of the IUT. The goal of specifying test purposes is to establish some degree of confidence that the IUT conforms to the specification. In model-based black-box testing the formal model of the IUT is derived from its I/O specification and is the starting point of the automatic test case generation. Therefore, it should be possible to map the test purposes derived from the specifications of the IUT into test purposes defined in terms of the IUT model. Examples of test purposes are “test a state change from state A to state B in a model“, “test whether some selected states of a model are visited“, “test whether all transitions in a model are visited at least once“, etc. All of the test purposes listed above are specified in terms of the structural elements (coverage items) of the model that should be traversed (covered) during the execution of the test.

For synthesising a tester that fulfills a particular test purpose we extend the original model of the IUT with so called traps. The traps are attached to the transitions of the IUT model and they can be used to define which model elements should be covered by the test. Signaling about a trap traversal is implemented by means of trap predicate (in case of FSM just by a Boolean trap variable) and trap update functions. A trap is initially set to $false$. The trap update functions are attached to the trap labeled transitions and computed when the transition is executed in the course of the test run. They set the traps to $true$ which denotes that the traps are covered.

The extended model of the IUT, M'_S is a tuple $(S_S, V'_S, I_S, O_S, E'_S)$. The extended set of variables V'_S includes variables of the IUT and the trap variables ($V'_S = V_S \cup T$), where T is a set of trap variables. E'_S is a set of transitions where each element of E'_S is a tuple (s, p', a, o, u', q), where p' is a transition guard that is a logic formula over V'_S, and u' is an update function over V'_S. For the sake of brevity we further denote the model of the IUT that is extended with trap variables also by M_S.

Figure 2 presents an example where the IUT model given in Figure 2 is extended with trap variables. The example presents a *visit all transitions* test purpose, therefore the traps are attached to all transitions, $T = \{t_0, \ldots, t_7\}$. In this example $V'_S = T$ and $p_k \equiv true$, $u_k \equiv t_k := true$ for each transition e_k, $k \in \{0, ..., 7\}$.

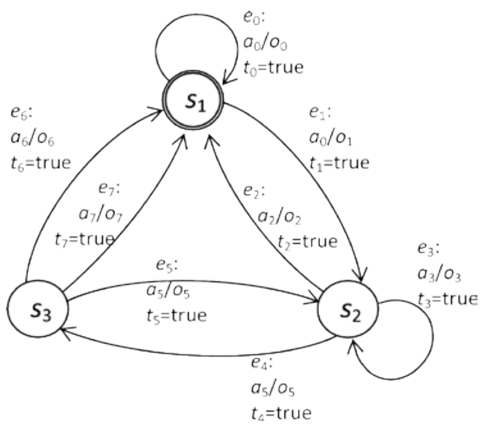

e_0: a_0/o_0 t_0=true
s_1
e_6: a_6/o_6 t_6=true
e_1: a_0/o_1 t_1=true
e_7: a_7/o_7 t_7=true
e_2: a_2/o_2 t_2=true
e_3: a_3/o_3 t_3=true
e_5: a_5/o_5 t_5=true
s_3
s_2
e_4: a_5/o_5 t_4=true

Figure 2: IUT model extended with trap variables.

2.6 Model of the Tester

The tester model M_T is synthesised from the IUT model M_S that is decorated with traps and their updates. The control structure of M_T is derived from the structural elements of M_S – states, transitions, variables, and update functions. We synthesise a tester EFSM M_T as a tuple $(S_T, V_T, I_T, O_T, E_T)$, where S_T is the set of tester states, V_T is the set of tester variables, I_T is the set of tester inputs, O_T is the set of tester outputs and E_T is the set of tester transitions. Necessary condition for the IO conformance of M_S and M_T is that their IO alphabets comply, $I_T = O_S$ and $O_T = I_S$, and the set of context variables of the tester is equal to the set of the context variables of the extended IUT model ($V_T = V'_S$).

The tester M_T has two types of states - active and passive. The set of active states S_T^a ($S_T^a \subset S_T$) includes the states where the tester has enabled transitions and by output functions of these transitions the tester selects stimuli to IUT, i.e., controls the test execution. The set of passive states S_T^p ($S_T^p \subset S_T$) includes the states of $M_T$where the tester is ready to receive reactions from IUT. The transitions $e_T \in E_T$ of the tester automaton are defined by a tuple $(s_T, p_T, a_T, o_T, u_T, q_T)$, where p_T is a transition guard that is a formula of logic over V_T and u_T is an update function over V_T. We distinguish observable and controllable transitions of the M_T. An observable transition e^o is a transition with a passive source state. It is defined by a tuple $(s_T, p_T \equiv true, a_T, o_T \equiv nil, u_T, q_T)$, where s_T is a passive state, the transition is always enabled ($p_T \equiv true$), and it does not generate any output symbol. A controllable transition e^c is a transition with an active source state of the M_T . It is defined by a tuple $(s_T, p_T, a_T \equiv nil, o_T, u_T \equiv nil, q_T)$, where s_T is an active state, the transition needs not receiving an input symbol, $p_T \equiv p_S \wedge p_g(V_T)$ is a guard of e^c constructed as a conjunction of the corresponding guard p_S of the extended IUT model M_S and the gain guard $p_g(V_T)$.

The purpose of the gain guard $p_g(V_T)$ is to guide the execution of M_T so that in each state only the outgoing transition is enabled that is a prefix of a path with maximum gain. In other words, the gain guards enable transitions that are the best in the sense of the path length from the current state towards fulfilling a still unsatisfied subgoal of the test purpose. We construct the gain guards $p_g(V_T)$ offline by analysing reachability of traps from each transition of M_T. The gain guards take into account the number and distance-weighted reachability (gain) of still unvisited traps. The tester model M_T can be non-deterministic in the sense that when there are many transitions with equal positive gain, the selection of the transition to be taken next is made either randomly from the best choices or by the principle "Least visited first".

3 SYNTHESIS OF ON-LINE PLANNING TESTER FOR FSM MODELS OF IUT

3.1 Synthesis of On-Line Planning Tester in Large

We describe the tester synthesis procedure at first based on the FSM model of IUT. The test purpose is expressed in terms of trap variable updates attached to the transitions of the IUT model. We also introduce the parameters that define the RPT planning constraints.

The RPT synthesis comprises three basic steps (Figure3): (i) extraction of the RPT control structure, (ii) constructing gain guards that includes also construction of gain functions, and (iii) reduction of gain guards according to the parameter "planning horizon" that defines the depth of the reachability tree to be pruned.

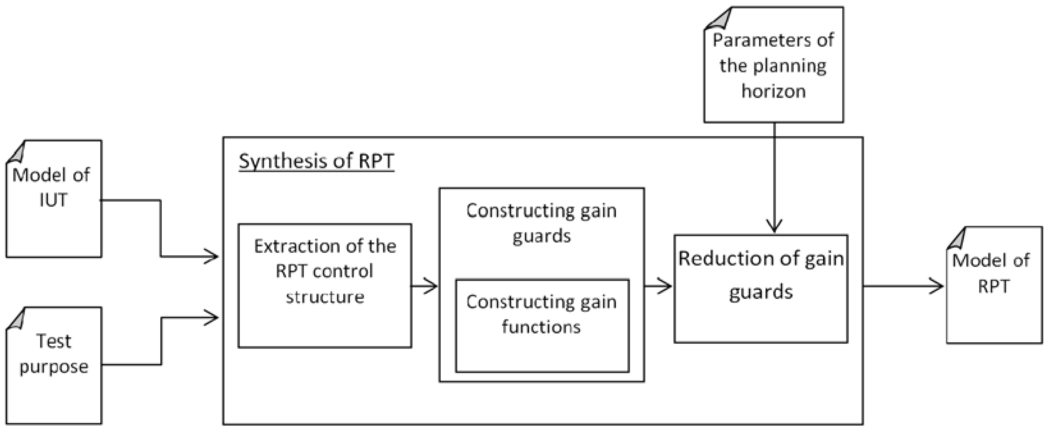

Parameters of the planning horizon
Model of IUT
Synthesis of RPT
Constructing gain guards
Extraction of the RPT control structure
Constructing gain functions
Reduction of gain guards
Model of RPT
Test purpose

Figure 3: RPT synthesis workflow

In the first step, the RPT synthesiser analyses the structure of the IUT model and generates the RPT control structure. In the second step, the synthesizer finds possibly successful IUT runs regarding the test goal. The tester should make its choice in each current state based on the structure of the tester model and the bindings of the trap variables representing the test goal. The decision rules for on-the-fly planning are derived by performing reachability analysis from the current state to all trap-equipped transitions by constructing the shortest path trees. The decision rules are defined for controllable-by-tester transitions of the model and are encoded in transition guards as conjuncts called gain guards. The gain functions that are terms in the decision constraints are derived from the reduced shortest path trees (RSPT) on IUT dual automaton. A shortest-paths tree is constructed for each controllable transition. The root vertex of the tree corresponds to the controllable transition being characterised with the gain function, other vertices present transitions equipped with traps. In case there are branches without traps in the tree that terminate with terminals labelled with traps, the branches are substituted with hyper-edges having weights equal to the length of that branch. By the given construction the RSPT represents the shortest paths from the root transition it characterises to all reachable trap-labelled transitions in the tester model. The gain function also allocates weights to the traps in the tree, and closer to the root transition the higher weight is given to the trap. Thus, the gain value decreases after each trap in the tree gets visited during the test execution.

Since the RSPT on IUT dual automaton has the longest branch proportional to the length of Euler's contour on that automaton graph the gain function's recurrent structure may be very complex. Last step of the synthesis reduces the gain functions pruning the RSPT up to some depth that is defined by parameter "planning horizon". In Sections 3.2 to 3.5 the RPT synthesis steps are described in more detail.

3.2 Deriving the Control Structure of the Tester

The tester model is constructed as a dual automaton of the IUT model where the inputs and outputs are inverted. The tester construction algorithm, Algorithm 1, has the following steps. The states of the IUT model are transformed into the active states of the tester model in step 1. For each state s of the IUT, the set of outgoing transitions $E_S^{out}(s)$ is processed in steps 2 to 5. Each transition of the IUT model is split into a pair of consecutive transitions in the tester model - a controllable transition $e_T^c \in E_T^c$ and an observable transition $e_T^o \in E_T^o$, where E_T^c and E_T^o are respectively the subset of controllable and subset of observable transitions of the tester model. A new intermediate passive state s_p is added between them (steps 6 – 8 of Algorithm 1).

Algorithm 1 Build control structure of the tester

```
1:  $E_T^c \leftarrow \emptyset$; $E_T^o \leftarrow \emptyset$; $S_T^a \leftarrow S_S$; $S_T^p \leftarrow \emptyset$; $I_T \leftarrow O_S$;
    $O_T \leftarrow I_S$; $V_T \leftarrow V_S$
2:  for all $s \in S_S$ do
3:    find $E_S^{out}(s)$
4:    while $E_S^{out}(s) \neq \emptyset$ do
5:      get $e = (s, p, a, o, u, q)$ from $E_S^{out}(s)$
6:      add $s_p$ to $S_T^p$ {passive state}
7:      add $(s, p, \emptyset, a, \emptyset, s_p)$ to $E_T^c$ {controllable transition}
8:      add $(s_p, true, o, \emptyset, u, q)$ to $E_T^o$ {observable trans.}
9:      $E_S^{out}(s) \leftarrow E_S^{out}(s) - \{e\}$
10:     find $E_S^{out}(s, a, p)$ from $E_S^{out}(s)$
11:     $E_S^{out}(s) \leftarrow E_S^{out}(s) - E_S^{out}(s, a, p)$
12:     while $E_S^{out}(s, a, p) \neq \emptyset$ do
13:       get $e = (s, p, a, o, u, q)$ from $E_S^{out}(s, a, p)$
14:       add $(s_p, true, o, \emptyset, u, q)$ to $E_T^o$ {observable trans.}
15:       $E_S^{out}(s, a, p) \leftarrow E_S^{out}(s, a, p) - \{e\}$
16:     end while
17:   end while
18: end for
19: for all $e \in E_T^c$ do
20:   construct gain function $g_e(V_T)$
21: end for
22: construct dual graph $G$ of the tester model $M_T$
23: for all $e \in E_T^c$ do
24:   construct gain guard $p_{g_e}(V_T)$
25:   $p \leftarrow p \wedge p_{g_e}(V_T)$
26: end for
```

Let $E_S^{out}(s, a, p)$ denote the subset of the nondeterministic outgoing transitions of the state s where the IUT input is a and the guard is p. The algorithm creates one controllable transition e_T^c for each set $E_S^{out}(s, a, p)$from state *s* to the passive state s_p of the tester model (step 7). The controllable transition e_T^c does not have any input and the input of the corresponding transition of the IUT becomes an output of e_T^c.

For each element $e \in E_S^{out}(s, a, p)$ a corresponding observable transition e_T^o is created in steps 8 and 14, where the source state s of e is replaced by s_p, the guard is set to $true$ and the output of the IUT transition becomes the input of the corresponding tester transition.

The processed transition e of the IUT is removed from the set of outgoing transitions $E_S^{out}(s)$ (step 9). From the unprocessed set $E_S^{out}(s)$ the subset $E_S^{out}(s, a, p)$ of remaining nondeterministic transitions with the same input a and a guard equivalent to p is found (step 10). For each $e \in E_S^{out}(s, a, p)$ an observable transition e_T^o is created (steps 12-16).

The gain functions for all controllable transitions of the tester are constructed using the structure of the tester (steps 19–21). Finally, for each controllable transition, a gain guard $p_g(V_T)$ is constructed (step 24) and the conjunction of $p_g(V_T)$ and the guard of e_T^c is set to be the guard of the corresponding transition of the tester (step 25).

The details of the construction of the gain functions and gain guards are discussed in the next subsection.

An example of the tester EFSM created by Algorithm 1 is shown in Figure 4. The active states of the tester have the same label as the corresponding states of the IUT and the passive states of the tester are labelled with $s_4, \ldots, s_9$. The controllable (observable) transitions are shown with solid (dashed) lines. For example, the pair of nondeterministic transitions e_0, e_1 of the IUT (see Figure 1) produces one controllable transition (s_1, s_4) and two observable transitions from the passive state s_4 of the tester.

For this example $V_T = T$, where T is the set of trap variables. For example, in Figure 4, $p_2^c(T)$ denotes the gain guard of the tester transition e_2^c. Gain guards attached to the controllable transitions of the tester (for example $p_2^c(T)$, $p_{34}^c(T)$) guide the tester at runtime to choose the next transition depending on the current trap variable bindings in T.

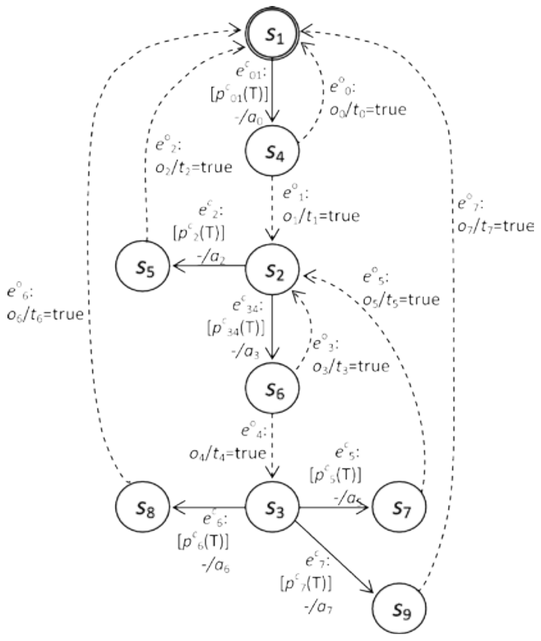


Figure 4: The EFSM model of the tester for the IUT in Figure 1

3.3 Constructing the Gain Guards of Transitions

A gain guard $p_g(V_T)$ of a controllable transition of the tester is constructed to meet the following requirements:

- The next move of the tester should be locally optimal with respect to achieving the test purpose from the current state of the tester.

- The tester should terminate after all traps are achieved or all unvisited traps are unreachable from the current state.

The gain guard evaluates to $true$ or $false$ at the time of the execution of the tester determining if the transition can be taken from the current state or not. The value $true$ means that taking the transition is the best possible choice to reach unvisited traps from the current state. The tester makes its choice in the current state based on the structure of the tester model, the bindings of the trap variables representing the test purpose, and the current bindings of the context variables. We need some measure of quantitative benefit to compare different alternative choices. For each controllable transition $e \in E_T^c$, where E_T^c is the set of all controllable transitions of the tester model, we define a non-negative gain function $g_e(V_T)$ that depends on the current bindings of the context variables. The gain function has the following properties:

- $g_e(V_T) = 0$, if taking the transition e from the current state with the current variable bindings does not lead closer to any unvisited trap. This condition indicates that it is useless to fire the transition *e* (P1)
- $g_e(V_T) > 0$, if taking the transition e from the current state with the current variable bindings visits or leads closer to at least one unvisited trap. This condition indicates that it is useful to fire the transition e. (P2)
- For transitions e_i and e_j with the same source state, $g_{e_i}(V_T) > g_{e_j}(V_T)$, if taking the transition e_i leads to an unvisited trap with smaller cost than taking the transition e_j. This condition indicates that it is cheaper to take the transition e_i rather than e_j to reach the unvisited traps. (P3)

A gain guard for a controllable transition e with the source state s of the tester is defined as

$$p_{g_e}(V_T) \;\equiv\; g_e(V_T) = \max_{e_k} g_{e_k}(V_T) \;\wedge\; g_e(V_T) > 0. \tag{1}$$

where g_{e_k} denotes the value of the gain function of the transition $e_k \in E_T^{out}(s)$, where $E_T^{out}(s) \subset E_T^c$ is the set of outgoing transitions of the state s.

The first predicate in the logical formula (1) ensures that the gain guard is $true$ if and only if it is the guard of the transition that leads to some unvisited trap from the current state with the highest gain compared to the gains of the other outgoing transitions of the current state. The second conjunct blocks test runs that do not serve the test purpose, i.e. it evaluates to $false$ when all unvisited traps from the current state are unreachable or all traps are visited already.

3.4 Gain Function

In this subsection, we describe how the gain functions are constructed. The required properties of a gain function were specified in the previous subsection (P1 - P3). Each transition of the IUT model is considered to have unit weight and the cost of the test case is proportional to the length of the test sequence(s) that cover all traps. The gain function of a transition computes a value that depends on the distance-weighted reachability of the unvisited traps from the given transition.

For the sake of efficiency, we implement a heuristic in the gain function that favors the selection of the path that visits more unvisited traps and is shorter than the alternative ones. Intuitively, in the case of two paths visiting the same number of transitions with unvisited traps and having the same lengths the path with more traps closer to the beginning of the path is preferred.

In this subsection, $M = (S, V, I, O, E)$ denotes the tester model equipped with trap variables and $e \in E$ is a transition of the tester. We assume that the trap variable $t \in T$ is initialised to $false$ and set to $true$ by the trap update function u_t associated with the transition e. Therefore, reaching a trap is equivalent to reaching the corresponding transition. A transition e_j is reachable from the transition e_i if there exists a path $\langle e_i, ..., e_j \rangle$ on the reachability tree of the model such that $e_i, e_j \in E$. For time being we ignore transition guards defined on context variables of the EFSM models.

Shortest-Paths Tree

In order to find the reachable transitions from a given transition we reduce the reachability problem of the transitions to a single-source shortest paths problem of a graph (Cormen, 2001). We create a dual graph $G = (V_D, E_D)$ of the tester model as a graph where the vertices V_D correspond to the transitions of the EFSM of the tester, $V_D \cong E$. The edges E_D of the dual graph represent the pairs of subsequent transitions sharing a state in the tester model. If the transition e_i of the tester model is an incoming transition of a state and the transition e_j is an outgoing transition of the same state, there is an edge (e_i, e_j) in the dual graph from vertex e_i to vertex e_j, $(e_i, e_j) \in E_D$.

The analysis of the transition sequences of the tester model M is equivalent to the analysis of the paths of vertices in the dual graph G. In Figure 5, there is the dual graph of the tester model depicted in Figure 4. For example, after taking the transition e^c_{01} in Figure 4, it is possible that either e^o_0 or e^o_1 follows. In the dual graph in Figure 5, this is represented by the existence of the edges to e^o_0 and e^o_1 from the vertex e^c_{01}.

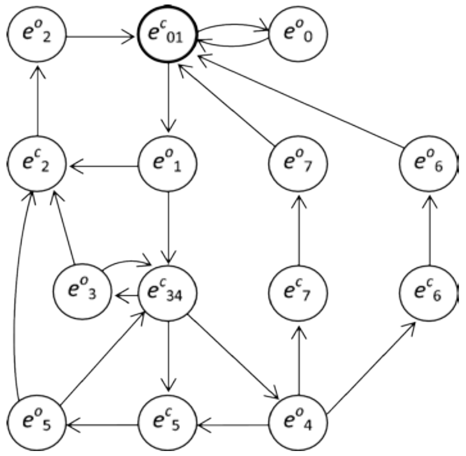


Figure 5: The dual graph of the tester model in Figure 4

In the dual graph, the shortest-paths tree from e is a tree with the root e that contains the shortest paths to every other vertex that is reachable from e. The shortest-paths tree with the root e derived from the graph G is denoted by $SPT(e, G)$. The shortest-paths tree from a given vertex of the dual graph can be found using known algorithms from the graph theory. Running a single source shortest-paths algorithm $|E^c|$ times results in the shortest paths from each controllable transition to every reachable transition.

The dual graph G is an unweighted graph (in this paper we assume that all transitions are uniformly priced). The breadth-first-search algorithm (see, for example (Cormen, 2001)) is a simple shortest-paths search algorithm that works on unweighted graphs. For a vertex e of the dual graph G the algorithm produces a tree that is the result of merging the shortest paths from the vertex e to each vertex reachable from it. As we constructed the dual graph in a way that the vertices of the dual graph correspond to the transitions of the tester model, the shortest path of vertices in the dual graph is the shortest sequence of transitions in the tester model. Each shortest path contains only distinct vertices. Note that the shortest paths and the shortest-paths trees of a graph are not necessarily unique.

The tree $SPT(e, G)$ represents the shortest paths from e to all reachable vertices of G. We assume that the traps of the IUT model are initialised to $false$ and a trap variable t is set to $true$ by an update function u associated with the transition of the IUT model. Therefore, the tree $SPT(e, G)$ represents also the shortest paths starting with the vertex e to all reachable trap assignments. Not all transitions of the tester model contain trap variable update functions. To decide the reachability of traps by the paths in the tree $SPT(e, G)$ it suffices to analyse the reduced shortest-paths tree (RSPT), denoted by $TR(e, G)$. RSPT $TR(e, G)$ includes the root vertex e and only such vertices of $SPT(e, G)$ that contain trap updates. We construct $TR(e, G)$ by replacing those sub-paths of $SPT(e, G)$ that do not include trap updates by hyper-edges. A hyper-edge denotes the shortest sub-path between two vertexes t_i and t_j in the shortest-paths tree such that t_i and t_j are labelled with trap assignments and any other vertex on that path is not. Thus, the reduced shortest-paths tree $TR(e, G)$ contains the shortest paths from root e to all reachable transitions labelled with trap updates in the dual graph G. In $TR(e, G)$ we label each vertex that contains a trap variable update u_t by the corresponding trap t and replace each sub-path containing vertices without trap updates by a hyper-edge (t_i, w, t_j) where t_i is the label of the origin vertex, t_j is the label of the destination vertex and w is the length of that sub-path. Also, during the reduction we remove those sub-paths (hyper-edges) that end in the leaf vertices of the tree that do not contain any trap variable updates.

Figure 6 (left) shows the shortest-paths tree $SPT(e^c_{01}, G)$ with the root vertex e^c_{01} for the dual graph in Figure 5. For example, the path $\langle e^c_{01}, e^o_1, e^c_{34}, e^o_4, e^c_6, e^o_6\rangle$ from the root vertex e^c_{01} to the vertex e^o_6 in the shortest-paths tree in Figure 6 is the shortest sequence of transitions beginning with the transition e^c_{01} that reaches e^o_6 in the example of the tester model in Figure 4.

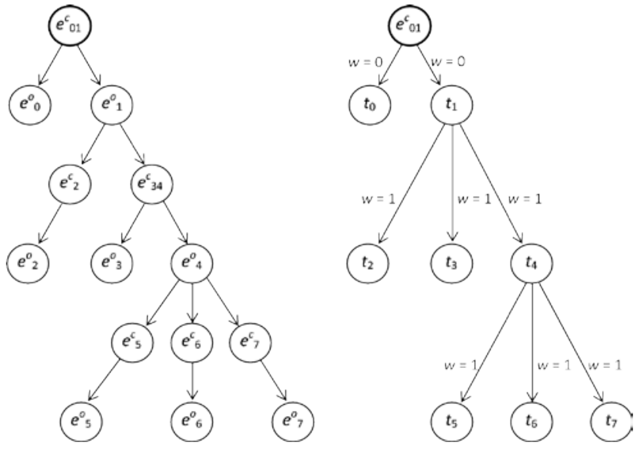


Figure 6: The shortest-paths tree $SPT(e^c_{01}, G)$ (left) and the reduced shortest-paths tree $TR(e^c_{01}, G)$ (right) from the transition e^c_{01} of the graph shown in Figure 5

The reduced shortest-paths tree $TR(e^c_{01}, G)$ from the vertex e^c_{01} to the reachable traps of the dual graph in Figure 5 is represented in Figure 6 (right). All vertices except the root of the reduced shortest-paths tree $TR(e^c_{01}, G)$ are labelled with the trap variables, and the hyper-edges between the vertices are labelled with their weights. The tree $TR(e^c_{01}, G)$ contains the shortest paths beginning with the transition e^c_{01} to all traps in the tester model in the Figure 4. For example, the tree $TR(e^c_{01}, G)$ shows that there exists a path beginning with the transition e^c_{01} to the trap t_6, and this path visits traps t_1 and t_4 on the way.

Algorithm for Constructing the Gain Function

The return type of the gain function is non-negative rational $\mathbb{Q}^+$. That follows explicitly from the construction rules of the gain function (see steps below) and from the fact that the corpus of rational numbers is closed under addition and the max operator. The gain function construction algorithm for transition e of the tester automaton M (having dual graph G) is following:

1. Construct the shortest-paths tree $SPT(e, G)$ for the transition e of the dual graph G of the tester control graph.

2. Reduce the shortest-paths tree $SPT(e, G)$ as described in subsection

3.4 (the reduced tree is denoted by $TR(e, G)$): Compute the lengths w of the minimal trap-free sub-paths between pairs of trap-labelled vertexes t_i and t_j of $SPT(e, G)$ and substitute these sub-paths with hyper-edges (t_i, w, t_j) labelled with weight w.

3. Represent the reduced tree $TR(e, G)$ as a set of elementary sub-trees of height 1, where each elementary sub-tree is specified by the production rule of the form

$$\nu_i \to |_{j\in\{1,\ldots,k\}}\nu_j, \tag{2}$$

   where the non-terminal symbol ν_i denotes the root vertex of the sub-tree and each ν_j (where $j \in \{1, \ldots, k\}$) denotes a leaf vertex of that sub-tree, k is the branching factor, and ν_0 corresponds to the root vertex e of the reduced tree $TR(e, G)$.

4. Rewrite the right-hand sides of the productions constructed in step 3 as arithmetic terms, thus getting the production rule in the form

$$\nu_i \to (\neg t_i)^{\uparrow} \cdot \frac{c}{d(\nu_0, \nu_i) + 1} + \max_{j=1,k}(\nu_j), \tag{3}$$

   where $t_i^{\uparrow}$ denotes the trap variable t_i of lifted type $\mathbb{N}$, c is a constant for the scaling of the numerical value of the gain function, and $d(\nu_0, \nu_i)$ the distance between vertexes ν_0 and ν_i in the labelled tree $TR(e, G)$. The distance is defined by the formula

$$d(\nu_0, \nu_i) = l + \sum_{j=1}^{l} w_j$$

   where l is the number of hyper-edges on the path between ν_0 and ν_i in $TR(e, G)$ and w_j is the value of the weight w corresponding to the concrete hyper-edge.

5. For each symbol ν_i denoting a leaf vertex in $TR(e, G)$ define a production rule:

$$\nu_i \to (\neg t_i)^{\uparrow} \cdot \frac{c}{d(\nu_0, \nu_i) + 1} \tag{4}$$

6. Apply the production rules (3) and (4) starting from the root symbol ν_0 of $TR(e, G)$ until all non-terminal symbols ν_i are substituted with the terms that include only terminal symbols $t_i^{\uparrow}$ and $d(\nu_0, \nu_i)$, ($i \in \{0, \ldots, n\}$, where n is the number of trap variables in $TR(e, G)$). The root vertex $\nu_0 = e$ of the labelled tree $TR(e, G)$ may not have a trap label. Instead of a trap variable t_i, we use a constant $true$ as the label resulting $(\neg true)^{\uparrow} = 0$ in the rule (3).

Table 1: Application of the production rules to the elementary sub-trees of height 1 of the reduced shortest-paths tree $TR(e_{01}^c, G)$

Sub-tree	Production rule (2)	Production rule (3) for non-leaf or production rule (4) for leaf vertex
e_{01}^c	$e_{01}^c \to t_0, t_1$	$e_{01}^c \to 0 \cdot c/1 + max(t_0, t_1)$
t_0	$t_0 \to$	$t_0 \to \neg t_0 \cdot c/2$
t_1	$t_1 \to t_2, t_3, t_4$	$t_1 \to \neg t_1 \cdot c/2 + max(t_2, t_3, t_4)$
t_2	$t_2 \to$	$t_2 \to \neg t_2 \cdot c/4$
t_3	$t_3 \to$	$t_3 \to \neg t_3 \cdot c/4$
t_4	$t_4 \to t_5, t_6, t_7$	$t_4 \to \neg t_4 \cdot c/4 + max(t_5, t_6, t_7)$
t_5	$t_5 \to$	$t_5 \to \neg t_5 \cdot c/6$
t_6	$t_6 \to$	$t_6 \to \neg t_6 \cdot c/6$
t_7	$t_7 \to$	$t_7 \to \neg t_7 \cdot c/6$

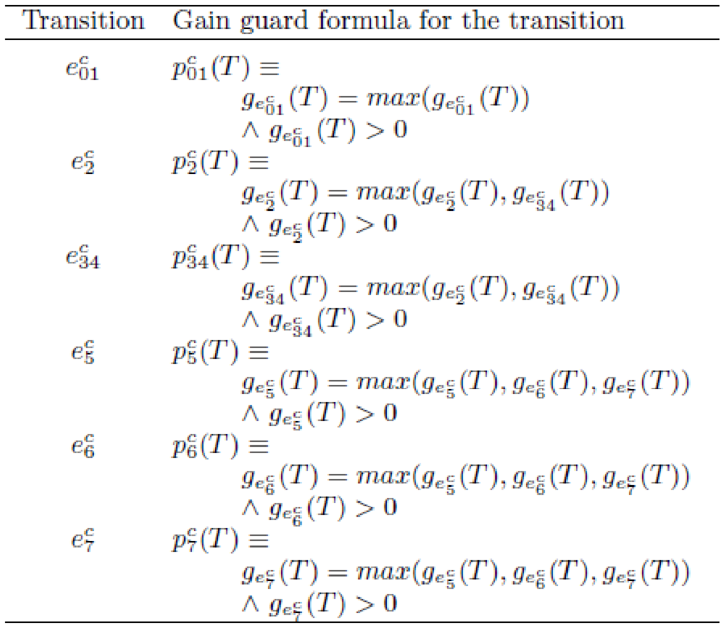
Table 2: Gain functions of the controllable transitions of the tester model

Transition	Gain function for the transition
e_{01}^c	$g_{e_{01}^c}(T) \equiv c \cdot max($ $\neg t_0/2,$ $\neg t_1/2 + max(\neg t_2/4, \neg t_3/4, \neg t_4/4 +$ $max(\neg t_5/6, \neg t_6/6, \neg t_7/6)))$
e_2^c	$g_{e_2^c}(T) \equiv c \cdot (\neg t_2/2 + max($ $\neg t_0/4,$ $\neg t_1/4 + max(\neg t_3/6, \neg t_4/6 +$ $max(\neg t_5/8, \neg t_6/8, \neg t_7/8))))$
e_{34}^c	$g_{e_{34}^c}(T) \equiv c \cdot max($ $\neg t_3/2 + \neg t_2/4 + max(\neg t_0/6, \neg t_1/6),$ $\neg t_4/2 + max(\neg t_5/4, \neg t_6/4, \neg t_7/4))$
e_5^c	$g_{e_5^c}(T) \equiv c \cdot (\neg t_5/2 + max($ $\neg t_2/4 + max(\neg t_0/6, \neg t_1/6),$ $\neg t_3/4,$ $\neg t_4/4 + max(\neg t_6/6, \neg t_7/6)))$
e_6^c	$g_{e_6^c}(T) \equiv c \cdot (\neg t_6/2 + max($ $\neg t_0/4,$ $\neg t_1/4 + max(\neg t_2/6, \neg t_3/6, \neg t_4/6 +$ $max(\neg t_5/8, \neg t_7/8))))$
e_7^c	$g_{e_7^c}(T) \equiv c \cdot (\neg t_7/2 + max($ $\neg t_0/4,$ $\neg t_1/4 + max(\neg t_2/6, \neg t_3/6, \neg t_4/6 +$ $max(\neg t_5/8, \neg t_6/8))))$

It has to be pointed out that the gain function characterizes the expected gain only within the planning horizon. The planning horizon is determined by the maximum length of the paths in the reduced shortest-paths tree.

Table 1 shows the results of the application of the production rules (2), (3) and (4) to the vertexes of the reduced shortest-paths tree $TR(e_{01}^c, G)$ in Figure 6 (right). As the root e_{01}^c is not labelled with a trap variable, the transition e_{01}^c does not update any trap, a constant $true$ is used in the production rule (3) in the place of the trap variable resulting $(\neg true)^{\uparrow} = 0$ in the first row of Table 1. Application of the production rules (3) and (4) to the tree $TR(e^c_{01}, G)$ starting from the root vertex e^c_{01} results in the gain function given in the first row of Table 2. Table 2 presents the gain functions for the controllable transitions of the tester model (Figure 4). The gain guards for all controllable transitions of the tester model are given in Table 3. The type lifting functions of the traps have been omitted from the tables for the sake of brevity.

Table 3: Gain guards of the transitions of the tester model



Transition	Gain guard formula for the transition
e^c_{01}	$p^c_{01}(T) \equiv$ $g_{e^c_{01}}(T) = max(g_{e^c_{01}}(T))$ $\wedge\ g_{e^c_{01}}(T) > 0$
e^c_2	$p^c_2(T) \equiv$ $g_{e^c_2}(T) = max(g_{e^c_2}(T), g_{e^c_{34}}(T))$ $\wedge\ g_{e^c_2}(T) > 0$
e^c_{34}	$p^c_{34}(T) \equiv$ $g_{e^c_{34}}(T) = max(g_{e^c_2}(T), g_{e^c_{34}}(T))$ $\wedge\ g_{e^c_{34}}(T) > 0$
e^c_5	$p^c_5(T) \equiv$ $g_{e^c_5}(T) = max(g_{e^c_5}(T), g_{e^c_6}(T), g_{e^c_7}(T))$ $\wedge\ g_{e^c_5}(T) > 0$
e^c_6	$p^c_6(T) \equiv$ $g_{e^c_6}(T) = max(g_{e^c_5}(T), g_{e^c_6}(T), g_{e^c_7}(T))$ $\wedge\ g_{e^c_6}(T) > 0$
e^c_7	$p^c_7(T) \equiv$ $g_{e^c_7}(T) = max(g_{e^c_5}(T), g_{e^c_6}(T), g_{e^c_7}(T))$ $\wedge\ g_{e^c_7}(T) > 0$

3.5 Adjustable Planning Horizon

Since the gain functions are constructed based on RSPTs their complexity is in direct correlation with the size of RSPT. In that way, the all transitions coverage criterion sets the number of traps equal to the number of transitions in the IUT model. Considering the fact that the number of transitions in the full-scale IUT model may reach hundreds or even more, the gain functions generated using RSPTs may grow over a size feasible to compute at test execution time. To keep the on-line computation time within acceptable limits RSPT pruning is added to the RPT synthesis technique. The planning horizon defines the depth of the RSPT to be pruned. Although the pruning of RSPT makes on-line planning incomplete it makes the RPT method fully scalable regardless of the size of IUT model and the test goal. Moreover, there is an option to set the planning horizon automatically offline when specifying the upper limit to the size of RSPT pruned. Pruning of RSPT reduces the resolution capability of RPT gain functions. In order to resolve the potentially rising priority conflicts between transitions having equal maximum gain values, RPT uses either random or anti-ant choice mechanisms. Both conflict resolution approaches are demonstrated on the City Lighting Controller case study and details discussed in Section 5.4.

3.6 Complexity of Constructing the Tester Based on EFSM Models with Feasibility Assumption

The complexity of the synthesis of the reactive planning tester based on EFSM models of IUT where all paths are feasible is determined by the complexity of the construction of the gain functions. For each gain function the complexity of finding the shortest-paths tree for a given transition in the dual graph of the tester model by breadth-first-search is $O(|V_D|+|E_D|)$ (Cormen, 2001), where $|V_D|=|E_T|$ is the number of transitions and $|E_D|$ is the number of transition pairs of the tester model. The number of transition pairs of the tester model is mainly defined by the number of transition pairs of the observable and controllable transitions which is bounded by $|E_S|^2$. For all controllable transitions of the tester the upper bound of the complexity of the offline computations of the gain functions is $O(|E_S|^3)$.

At runtime each choice by the tester takes no more than $O(|E_S|^2)$ arithmetic operations to evaluate the gain functions for the outgoing transitions of the current state.

4 PERFORMANCE EVALUATION OF RPT USING CASE STUDY EXPERIMENTS

The experiments are made to prove the feasibility of the RPT method and to compare its performance with the random choice and anti-ant methods using an industry scale case study.

4.1 The Case Study

The testing case study developed under the ITEA2 D-MINT project (ITEA2 project “Deplyment of Model-Based Technologies to Industrial Testing” Website, 2010) evaluates the model-based testing technology in the telematics domain. The IUT of the case study is a Feeder Box Control Unit (FBCU) of the street lighting control system. The most important functionality of the FBCU is to control street lighting lamps either locally, depending on the local light sensor and calendar, or remotely from the monitoring centre. In addition, the controller monitors the feeder box alarms and performs power consumption measurements. The communication between the controller and monitoring centre is implemented using GSM communication. The RPT performance evaluation experiments are performed on the powering up procedure of the FBCU.

4.2 Model of the IUT

The model implements the power-up scenario of the FBCU. The strongly connected state model of the FBCU includes 31 states and 78 transitions. The model is non-deterministic. Pairs of non-deterministic transitions depart from seven states of the model and a triple of non-deterministic transitions departs from one state of the model. The minimum length of the sequences of transitions from the initial state to the farthest transition is 20 transitions, i.e. the largest depth of the RSPT for any transition is 20. The model is similar to the model of well known digital door lock example that has several nested loops. There are several possibilities to fall from the successful scenario back to the first states if something goes wrong in the scenario.

4.3 Planning of Experiments

In order to demonstrate the algorithms in different test generation conditions we varied the test coverage criterion. The tests were generated using two different coverage criteria - all transitions and a single selected transition. The single transition was selected to be the farthest one from the initial state. The location of the single transition was selected on the limit of the maximum planning horizon. Different RPT planning horizons (0 to 20 steps) were used in the experiments. In case the RPT planning resulted in several equally good subsequent transitions in the experiment with the selected coverage criterion and planning horizon we used alternatively the anti-ant and random choice methods for choosing the next transition. If the planning horizon is zero then RPT works like pure random choice or anti-ant method depending on the option selected in the experiment.

As a characteristic of scalability we measured the length of test sequences and time spent on-line on each planning step. The planning time is indicative partially only because it depends on the performance of the RPT executing platform. Still, those measurements give some hints about the scalability of the method with respect to the planning horizon. In addition to the non-deterministic model there is always a random component involved in the RPT planning method. Therefore we performed all experiments in series of 30 measurements and calculated averages and standard deviations over the series.

4.4 Results and Interpretation of the Experiments

The experiments are summarized in Table 4 and in Table 5. The lengths of the test sequences are given in the form average ± standard deviation of 30 experiments. The results in the first row of Table 4 and Table 5 with planning horizon 0 correspond to the results of the pure anti-ant and random choice methods. For estimation of the minimum test sequence length we modified the examined non-deterministic model to the corresponding deterministic model with the same structure. Eliminating the non-determinism in the model by introducing mutually exclusive transition guards and using the maximum planning horizon 20 the reactive planning tester generated the test sequence with length 207 for "all transitions" coverage criteria on the modified deterministic model. The minimum length of the test sequence to reach the "single selected transition" was 20 steps.

Table 4: Average lengths of the test sequences satisfying the "all transitions" test purpose

Length of planning horizon (number of steps)	anti-ant	random choice
0	18345 ± 5311	44595 ± 19550
1	18417 ± 4003	19725 ± 7017
2	5120 ± 1678	4935 ± 1875
3	4187 ± 978	3610 ± 2538
4	2504 ± 815	2077 ± 552
5	2261 ± 612	1276 ± 426
6	2288 ± 491	1172 ± 387
7	1374 ± 346	762 ± 177
8	851 ± 304	548 ± 165
9	701 ± 240	395 ± 86
10	406 ± 102	329 ± 57
11	337 ± 72	311 ± 58
12	323 ± 61	284 ± 38
13	326 ± 64	298 ± 44
14	335 ± 64	295 ± 40
15	324 ± 59	295 ± 42
16	332 ± 51	291 ± 52
17	324 ± 59	284 ± 32
18	326 ± 66	307 ± 47
19	319 ± 55	287 ± 29
20	319 ± 68	305 ± 43

The experiment shows that the reactive planning tester with maximum planning horizon results on average in a test sequence many times shorter and a considerably lower standard deviation than the anti-ant and random choice tester. For the test goal to cover all transitions of the non-deterministic model the RPT generated an average test sequence 1.5 times longer than the minimum possible sequence. The difference from the optimum is mainly due to the non-determinism of the model. Compared to the RPT with the maximum planning horizon the anti-ant and random choice tester generated test sequences that were on average 57 and 146 times longer, respectively.

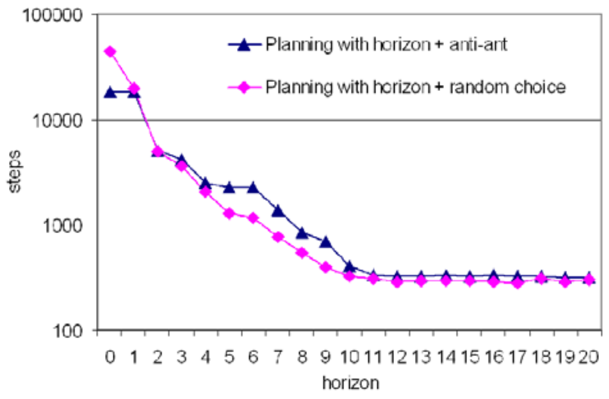
Table 5: Average lengths of test sequences satisfying the test purpose to cover one single transition (the farthest transition from the initial state)

Length of planning horizon (number of steps)	anti-ant	random choice
0	2199 ± 991	4928 ± 4455
1	2156 ± 1154	6656 ± 5447
2	1276 ± 531	2516 ± 2263
3	746 ± 503	1632 ± 1745
4	821 ± 421	1617 ± 1442
5	319 ± 233	618 ± 512
6	182 ± 116	272 ± 188
7	139 ± 74	147 ± 125
8	112 ± 75	171 ± 114
9	72 ± 25	119 ± 129
10	73 ± 29	146 ± 194
11	79 ± 30	86 ± 59
12	41 ± 15	74 ± 51
13	34 ± 8	48 ± 31
14	34 ± 9	40 ± 23
15	25 ± 4	26 ± 5
16	23 ± 2	24 ± 3
17	22 ± 2	21 ± 1
18	21 ± 1	21 ± 1
19	21 ± 1	21 ± 1
20	21 ± 1	21 ± 1

If the test goal is to cover one selected transition (Table 5), the RPT reached the goal with the length of test sequence that is close to optimal. The anti-ant and random choice tester required on average 104 and 235 times longer test sequences. This experiment shows that the anti-ant tester outperforms the random choice tester by more than twice on average with smaller standard deviation. This confirms the results reported in (Li & Lam, 2005).

The dependency of the test sequence length on the planning horizon is shown in Figure 7. Non-smoothness of the curves is caused by the relatively small number of experiments and large standard deviation of the results. The planning horizon can be reduced to half of the maximum planning horizon without significant loss of average test sequence lengths for “all transitions” coverage criterion in this model. Even if planning few steps ahead significantly shorter test sequences were obtained than in case of the random or anti-ant methods. For instance, when the planning horizon is restricted to 2 or 5 steps, the average test sequence length decreases by approximately 4 or 8 times, respectively, compared to the anti-ant and random methods. If the test goal is to cover a single transition, then the test sequence length decreases exponentially to the value of the planning horizon.

At planning horizons less than maximum, there is no clear preference among the methods that could resolve the non-determinism of transition selection. The anti-ant method performs better for all horizon lengths in case of the “single transition” coverage criterion (Figure 7, right) and for small values of horizon length in case of “all transitions” coverage (Figure 7, left). The random choice method performs better on average for horizon lengths from 4 to 10 (Figure 7, left) for this model for the “all transitions” coverage criterion.



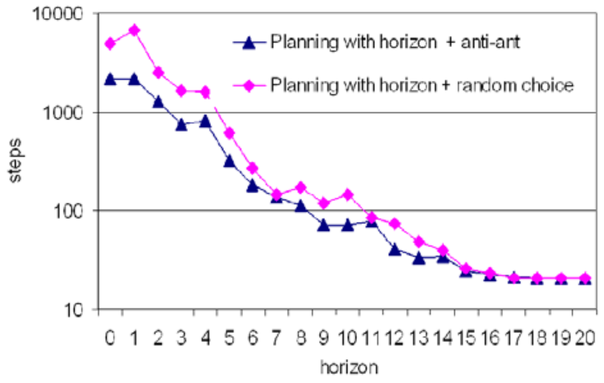


Figure 7: Average test sequence lengths of the test sequences satisfying the all transitions (left) and single transition (right) test goal

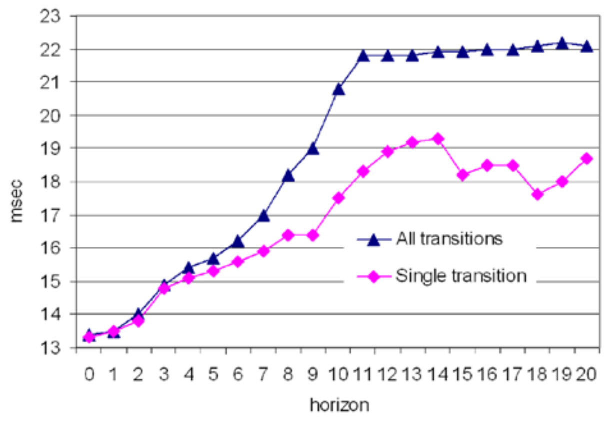


Figure 8: Average time spent for on-line planning of the next step

We also measured the time spent by tester for one on-line planning (selection of a test stimulus). The average duration of a planning step in milliseconds is shown in Figure 8. The computer used for experiments has an Intel Core 2 Duo E6600 processor running at 2.4 GHz. Experiments on the model demonstrate that the growth of planning time with respect to the planning horizon is not more than quadratic. The average time for calculating the gain function values with a maximum planning horizon in one step is less than 9 milliseconds. When the planning horizon is increased to maximum then the average depth of the shortest paths trees remains below the maximum horizon and the average planning time stabilizes.

5 EXTENDING THE REACTIVE PLANNING TESTER FOR EFSM MODELS OF IUT

5.1 Method in general

In this section, we extend the on-line planning tester synthesis method to EFSM models of IUT with restrictions that (i) the state variables must be of finite domain, (ii) the IUT automaton must be output-observable, i.e. the transitions taken by IUT are recognizable by the tester. We recall shortly some informal definitions related to EFSM models. EFSM is a collection of *states* S, *transitions* T, *state variables* V and *input variables* I. States and transitions are labeled by names. Every transition $t \in T$ has a $source_t$ and $target_t$ state and is attributed by a $guard_t$ and $update_t$. A $guard_t$ is a predicate on state and input variables and must evaluate to $true$ for the transition t to be enabled. An $update_t$ is a set of assignments of expressions to state variables. The expressions can contain both state and input variables. The types of the variables and operations allowed in the updates and guards are determined by the underlying solvers used. It is safe to constrain the domain to booleans, finite enumerations with equality and bounded integers with linear arithmetic, but it can be broadened. We do not model input and output symbols separately, the variables of enumeration type can be used for that purpose. External assignment of input variables is assumed whenever an input variable occurs in the guard or update of the transition to be taken. The only condition to outputs is that the automaton must be output-observable, i.e. the transition taken is detectable by the tester. A *configuration* (S, V) is a tuple of a state and state variables. An initial configuration $(S_0, V_0) \subseteq (S, V)$ is a subset of all configurations.

The goal of the test is specified as a set of ***traps*** TR. In the sequel we define a trap tr as a pair (t_{tr}, P_{tr}) where t_{tr} is a transition and P_{tr} is a predicate defined on variables. Covering a trap means taking the transition t_{tr} in a configuration $(source_{t_{tr}}, P_{tr})$, where the trap condition P_{tr} is satisfied in the pre-state of the trap transition t_{tr}. Defining trap in this way allows to express many different coverage criteria, e.g. path, all transitions or state variable border conditions. In order to avoid multi-level indexing, a notation $guard_{tr}$ means the guard of the transition t associated to the trap tr. To model the traps as a part of the EFSM model a boolean variable v_{tr} and update $v_{tr} \leftarrow P_{tr} \vee v_{tr}$ of the transition t_{tr} is added to the EFSM model for every trap $tr \in TR$ and all the trap variables v_{tr} are initialized to $false$.

By a set $Path(t, tr)$ we mean a set of all transition sequences $< t, ..., t_{tr} >$ from transition t to the transition of trap tr , where all the transitions are feasible for the model and P_{tr} is satisfied in the $source_{t_{tr}}$. Covering a trap tr means finding a path in $Path(t_i, tr)$ for transitions t_i leaving from initial states. Length of a $Path(t, tr)$ is the number of transitions in the sequence $< t, ..., t_{tr} >$. Feasibility constraint $Feas(Path(t, tr))$ is a predicate on variables on state $source_t$ such that $Path(t, tr)$ is feasible.

The testing process is divided into the computationally expensive off-line phase where a IUT model is analyzed and the efficient on-line phase where the instances of test input data are generated for guiding the IUT towards the uncovered traps. The off-line constraint and measure generation comprises a breath-first backwards constraint propagation static analysis algorithm. The propagation continues until the fixpoint is reached or the search horizon bound is met. The result of the off-line process is a set of constraints and expected gain measures to make the decisions on-line. More exactly, for every pair of a state s of the IUT EFSM and a trap tr the following is generated:

1. a *shortest path constraint* $C_{s,tr}$ being a sufficient feasibility condition for the shortest paths of $Path(t, tr)$ where $s = source_t$ is pre-state of t; and its length $L_{s,tr}$;

2. a *weakest constraint* $C^*_{s,tr}$ being a sufficient feasibility condition for any path p, p' in $Path(t, tr)$ where $s = source_t$ and the length of the paths does not exceed $L^*_{s,tr}$. $L^*_{s,tr}$ is equal to the search horizon bound or the length of the longest path p that's feasibility condition has a model that is not a model for the feasibility constraint of any other shorter path p'. $\neg(\bigvee_{p'} Feas(p') \rightarrow Feas(p))$ for all p' with $length_{p'} < length_p$. $C^*_{s,tr}$ expresses a fixpoint of $Feas(Path(t, tr))$ in the later case and $L^*_{s,tr}$ is the length of the longest path contributing to the fixpoint calculation.

The exact rules for calculating the constraints are presented in section
5.3.

For every pair of a transition t and trap tr the following is generated:

1. a *shortest path constraint* $C_{t,tr}$ being a sufficient reachability condition for the shortest path of set $Path(t, tr)$ and its length $L_{t,tr}$;

2. a *weakest constraint* $C^*_{t,tr}$, that is a sufficient reachability condition for any path p, p' in $Path(t, tr)$ with length not exceeding $L^*_{t,tr}$. $L^*_{t,tr}$ is equal to the search horizon bound or the length of the longest path p that's feasibility condition has a model that is not a model for the feasibility constraint of any other shorter path p'. $\neg(\bigvee_{p'} Feas(p') \rightarrow Feas(p))$ for all p' with $length_{p'} < length_p$. $C^*_{t,tr}$ expresses a fixpoint of $Feas(Path(t, tr))$ in the later case and $L^*_{t,tr}$ is the length of the longest path contributing to the fixpoint calculation.

3. a *guarding constraint* $C^g_{t,tr}$ on state variables evaluates to $true$ for the transition t if t is the initial transition of a shortest path of $Path(t, tr)$ considering the actual valuation of the state variables.

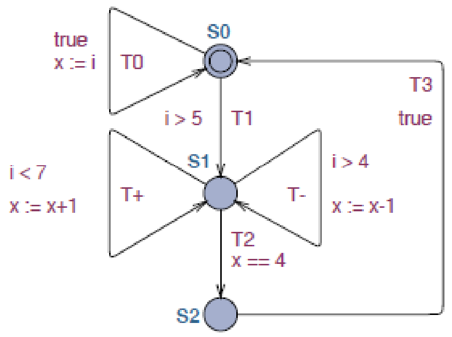
The on-line process takes the generated constraints, distance measures and the IUT model as an input. It does a three step planning on every step of the testing process:

- selects a trap from the set of uncovered traps to be taken next
- selects a transition to guide IUT closer to the trap
- selects an input to take the chosen transition

Computationally demanding parts of the tester like simplification, quantifier elimination and satisfiability checks of the constraints are handled by the state of art SMT solver.

5.2 Simple example

We demonstrate the result of off-line computation and on-line test data generation on a simple model of a double counter in Figure 9 before we explain the method more precisely. The model has one state variable x and input variable i, both of integer type with range $[0, 10]$. Every transition is attributed by a label, guard and optional update. The table shows the constraints generated by the off-line computation for the trap $(T2, true)$. The constraints C on the third column are satisfied only for some values of x and i that make the shortest paths with length L on the EFSM control structure reachable. For example the condition $C_{T+,T2}$ means that the shortest path with length 2 to the trap starting with transition $T+$ is feasible only when the value of x is 5 and input must be chosen to be greater than 5. The weakest conditions C^* on the fifth column give the largest set of values of the variables that can be used for reaching the trap. For any input value satisfying the constraint, there is a path to the trap not longer than L^*. The result of a constraint $C^g_{T1,T2}$ seems unintuitive on the first glimpse. It is clear that a path starting with $T1$ can eventually lead to the trap regardless of the value of x in state $S0$, but it is not reflected in the constraint. The reason is that the calculation reaches a fixpoint for $S1$ on step 4, as can be seen from the values of $C^g_{S0,T2}$ and $L^g_{S0,T2}$ due to presence of transition $T0$. $C^g_{T1,T2}$ expresses condition on the state variables for paths no longer than 4, but it is sufficient for our purposes and there is no need to generate more general constraint. A condition $C^g_{t,tr}$ is satisfied in current valuation of the data variables when the shortest path to trap tr starts with transition t. The conditions are used to guide the tester towards the trap. It can be seen most clearly from the conditions $C^g_{T+,T2}$, $C^g_{T-,T2}$, and $C^g_{T2,T2}$ for the transitions $T+$, $T-$, and $T2$ leaving from state $S1$.



via	to	C^g	C	L	C^*	L^*
T0	T2	$x \neq 4$	$i = 4$	3	$i = 4$	3
T1	T2	$x \geq 3 \wedge x \leq 5$	$i > 5 \wedge x = 4$	2	$i > 5 \wedge x \geq 2 \wedge x \leq 6$	4
T+	T2	$x \leq 3$	$i \leq 6 \wedge x = 3$	2	$i \leq 6 \wedge x \leq 6$	5
T-	T2	$x \geq 5$	$i > 4 \wedge x = 5$	2	$i > 4 \wedge x \geq 1 \wedge x \leq 10$	7
T2	T2	$x = 4$	$x = 4$	1	$x = 4$	1
T3	T2	$true$	$x = 4$	3	$true$	3
S0	T2	-	$x = 4$	2	$true$	3
S1	T2	-	$x = 4$	1	$true$	7
S2	T2	-	$x = 4$	3	$true$	4

Fig. 9 Model of IUT (double counter) and generated constraints

Lets have a look what happens on-line when the real inputs must be generated, assuming that we have all the constraints prepared off-line. We start from state $s0$ with x equal to 5. The guarding constraints are used for choosing a right transitions, but $C^g_{T0,T2}$ and $C^g_{T1,T2}$ are both satisfiable and do not constrain the choice, because a path with length 3 is possible both ways. We have a non-deterministic model and nothing in the model forces $T1$ to be taken, but let us assume that the random choice works for our favor this time. Choosing transition $T1$ gives a concrete instance $i > 5 \wedge 5 \geq 2 \wedge 5 \leq 6$ of constraint $C^g_{T1,T2}$ to be solved and an input $i = 6$ is generated. Guarding constraints $C^g_{T+,T2}$, $C^g_{T-,T2}$, and $C^g_{T2,T2}$ determine that $T-$ is the transition of choice from state $S1$. Just solving $C_{T-,T2}$ for determining the input i can give a value 5 which can trigger $T+$ also. Solving $C^*_{T-,T2} \wedge \neg guard_{T+} \wedge \neg guard_{T2}$ gives value 7 for the input and resulting $T-$ to be taken and x to be equal to 4. Next step does not depend on input, but the guard of $T2$ is satisfied and taken eventually.

5.3 Offline computation

The generation of reachability constraints that guide on-line testing process is carried out off-line. The reachability constraints for transition-trap and state-trap pairs are constructed by backwards breath-first propagation of the constraints starting from the traps. The shortest path constraints C are constructed when the transition or state with constraint not equal to $false$ is encountered first in that propagation. For finding the weakest condition C^* the computation continues and the constraints are weakened at each step until the fixpoint is reached or the search depth bound is reached. The fixpoint is guaranteed to exist as long we restrict the model to be of finite domain, but finding it may be computationally infeasible and the computation is canceled at some traversal depth. In that case, the constraints express the conditions for the paths with length up to the bound.

Algorithm 2 presents the procedure for finding the constraints and path lengths for on-line test navigation. The algorithm employs the monotonic nature of the constraint derivation. It carries over only the changes C^{Δ} discovered at each traversal step and adds the result to the previous value $C^{*'}$ of the constraint C^{*} as a new disjunct (lines 8, 12). State condition change $C^{\Delta}_{s,tr}$ is calculated (line 6) by eliminating all the inputs I from the disjunction of constraint changes of the outgoing transitions of the current state s. Input ellimination is carried out by the existential quantifier ellimination procedure in the simplification procedure. Transition condition change $C^{\Delta}_{t,tr}$ is a conjunction of two constraints (line 11). The first conjunct $guard_t$ is the guard of the transition t. The second conjunct is *the weakest precondition* of the current transition's update $update_t$ and of the condition change $C^{\Delta}_{s,tr}$ of that transition's target state s. The *weakest precondition* calculation is a straightforward substitution in case the update is a collection of evaluations and assignments. The most complicated is the calculation of the guarding constraints C^{g} (line 14). The update of the constraint $C^{g}_{t,tr}$ can be interpreted as the valuation of the state variables that satisfy the transition's constraint change $C^{\Delta}_{t,tr}$ but do not satisfy the constraint $C^{*'}_{source(t),tr}$ of the source state of the transition t and will be used to extend the interpretation set of the $C^{*'}_{source(t),tr}$ in the next iteration. Constraints C_s, C_t for the shortest paths are determined when satisfiable constraint change $C^{\Delta}_{s}, C^{\Delta}_{t}$ is found (line 9, 13). The fixpoint is reached when no weakening happens on the traversal step and it is checked by the constraint satisfiability check (SAT) procedure (line 7). Some simplification procedures are applied to all intermediate results to reduce the size of the formula.

Algorithm 2 Off-line constraint generation

```
find constraints(Trap $tr$)
1.  initialize all constraints related to trap $tr$ to $false$
2.  initialize all lengths related to trap $tr$ to 0
3.  $C_{tr,tr} \leftarrow C^{*}_{tr,tr} \leftarrow C^{\Delta}_{tr,tr} \leftarrow guard_{tr} \wedge P_{tr}$
4.  repeat    // breath-first traversal
5.    foreach state $s$ in the $depth$ level do
6.      $C^{\Delta}_{s,tr} \leftarrow$ simplify($\exists I : \bigvee_{ti} C^{\Delta}_{ti,tr}$) $where\ source_{ti} = s$
7.      if sat($\neg(C^{\Delta}_{s,tr} \Rightarrow C^{*'}_{s,tr})$) then        // if $C^{*}_{s,tr}$changed
8.        $C^{*}_{s,tr} \leftarrow$ simplify($C^{*'}_{s,tr} \vee C^{\Delta}_{s,tr}$); $L^{*}_{s,tr} \leftarrow depth$
9.        if $C'_{s,tr} = false$ then  $C_{s,tr} \leftarrow C^{*}_{s,tr}$;  $L_{s,tr} \leftarrow L^{*}_{s,tr}$
10.       foreach transition $t$ having $s$ as target state do
11.         $C^{\Delta}_{t,tr} \leftarrow guard_t \wedge wp(update_t, C^{\Delta}_{s,tr})$
12.         $C^{*}_{t,tr} \leftarrow$ simplify($C^{*'}_{t,tr} \vee C^{\Delta}_{t,tr}$); $L^{*}_{t,tr} \leftarrow depth + 1$
13.         if $C'_{t,tr} = false$ then  $C_{t,tr} \leftarrow C^{*}_{t,tr}$;  $L_{t,tr} \leftarrow L^{*}_{t,tr}$
14.         $C^{g}_{t,tr} \leftarrow$ simplify($C^{g'}_{t,tr} \vee ((\exists I : C^{\Delta}_{t,tr}) \wedge \neg C^{*'}_{source(t),tr})$)
15. until fixpoint or max depth level is reached
```

Tuning the planning horizon or depth level of the search allows a trade-off to be found between close to optimal (in terms of test length) and scalability of tester behavior with computationally feasible expenses. The discussion about finding a suitable planning horizon is given in Section 5.6.

5.4 On-line computation

The goal of on-line computation during a test run is to find the shortest possible path covering the maximal number of traps while keeping the on-line computation as efficient as possible. The planning, based on pre-computed constraint set, is done repetitively, i.e. before executing each EFSM transition. Planning is performed in three steps (Algorithm 3): (i) the succession of traps is planned; (ii) the path from current state to the next trap is planned; (iii) the data is generated for IUT to guide the IUT along the preferred path.

Algorithm 3 On-line planning

```
1.  $st$=initial state
2.  while exist uncovered traps do  // at state $s$
3.    //(i) select next trap (greedy)
4.    select trap $tr$ with
5.      min($L_{s,tr}$, $L^*_{s,tr}$) and corresponding $C_{s,tr}$ or $C^*_{s,tr}$ satisfiable
6.    //(ii) select next transition
7.    select $t$ with  $source_t = s \wedge C^g_{t,tr}$ satisfiable
8.    //(iii) generate inputs
9.    select input valuation by solving $C_{t,tr}$ or $C^*_{t,tr}$
10.   communicate the inputs to the IUT
11.   detect the target state $s$ of the IUT move
12.   if $s$ does not conform to the model
13.     return(fail)
14. end while
15. return(passed)
```

The next trap to reach from current state is selected in step (i) using the lengths $L_{s,tr}, L^*_{s,tr}$ to traps found off-line. The lengths serve as interval estimates of the distances to traps and are used for planning the order the traps have to be taken. The actual test length depends on the valuation of the variables and cannot be determined off-line. There are several strategies for selecting the order of traps starting with the greedy approach to guide the test towards the closest uncovered trap and ending with the global planning approach that involves solving NP-complete asymmetric traveling salesman problem (ATSP) for finding a shortest path through all traps. This can be computationally quite expensive when the number of traps is large. Still, this is not the issue because the intended order of covering traps can be computed off-line. Fast heuristic approximating ATSP algorithms can be applied also later in on-line phase to refine the plan when the IUT due to its non-determinism deviates from the planned path. Alternatively, the greedy approach does all the planning on-line trying to reach the closest trap from the current state taking into account data constraints. The planning horizon can be parametrically tuned from greedy to global planning by setting how many traps ahead the planning covers.

To guide IUT towards the trap chosen in step (i) the next transition is selected in step (ii) using the guarding constraints $C^g_{t,tr}$ of outgoing from current state s transitions t. The guarding constraints of outgoing transitions are mutually exclusive except when two transitions prefixing two different paths to the same trap tr have equal lengths and non-contradictory data constraints. For checking the constraints $C^g_{t,tr}$ we apply a simple heuristic that chooses constraints in the order of increasing values of $L_{t,tr}$.

In order to take the chosen transition in step (ii) a suitable input must be generated in step (iii). The input is generated by solving the constraint of the path using random choice, border value, or corner value data coverage strategy. The most liberal constraint that can be used is denoted by $C^*_{t,tr}$. This constrains the input to the values that guide IUT towards the trap along the path that is not longer than $L^*_{t,tr}$. It may not be the optimal path and the values satisfying $C^*_{t,tr}$ may trigger also some other transition in the case of non-deterministic automaton. The negations of the guards of neighboring transitions may be conjoined to the constraint $C^*_{t,tr}$ to rule out the non-deterministic choice. Alternatively the constraint $C_{t,tr}$ can be used, if satisfiable, to guide the IUT to the trap along the shortest path. Input generation involves constraint solving which is not in the scope of this paper. We assume that the constraints involving propositional logic and linear inequalities can be solved efficiently by standard methods.

5.5 Example

Inres protocol is a well-known example in the model verification and test generation community. The protocol is simple but not trivial and provides a good reference for studying performance and scalability issues of competing methods. The protocol was introduced in (Hogrefe, 1991) and the Inres Initiator model is depicted in Figure 10 as an EFSM. The model is deterministic and does not demonstrate the full potential of the method presented. The on-line phase of input data generation can be carried out also off-line for deterministic systems.

The model has 4 states, 14 transitions, 2 state variables *counter* and *number*, and 2 input variables *inp* and *num*. The integer variable *counter* has a range $0 \ldots 4$, *number* and *num* have a range $0 \ldots 1$ and the enumeration variable *inp* models the input messages *DR, CC, AK, ICONreq*, and *Timer.timeout*.

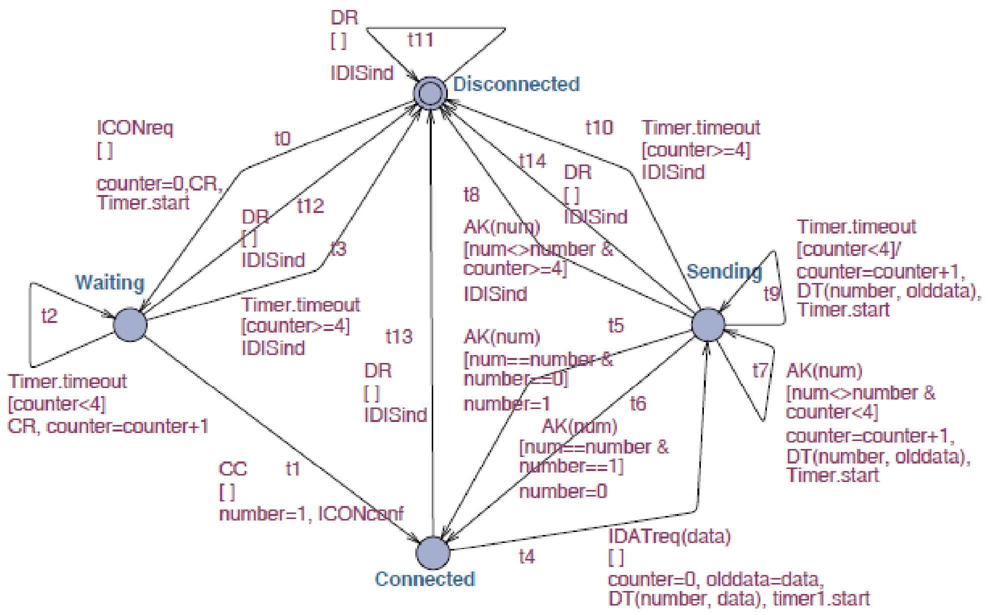


Fig 10: EFSM model of INRES protocol

An excerpt of the constraints and distance measures generated by the off-line tester synthesis is presented in Table 6. Traps are defined for transitions $t0 - t6$ with condition $true$ and shown in column *To*. The constraints and distance measures are given for a pair of transitions in columns *Via* and *To*, i.e. the constraint $inp = DR$ on the second line of the column *C* is $C_{t11,(t0,true)}$. Values C in the column *C** mean that the constraint is the same as in column *C*.

Table 7 explains the on-line tester behavior for guiding the IUT towards the trap on transition *t3* from the initial state *Disconnected*. The data in the table expresses the results of constraint solving done in the on-line process. Empty entries mean that there was no need to solve the corresponding constraints. The column *Next* expresses the decision of the transition to be taken next and it succeeds always because of the deterministic nature of the model. Two steps similar to step 2 are omitted.

Table 6: Excerpt of generated constraints for the Inres Initiator example

Via	To	C^g	C	L	C*	L*
t0	t0	true	inp = ICONreq	1	C	1
t11	t0	false	inp = DR	2	C	2
t1	t1	true	inp = CC	1	C	1
t2	t1	false	counter$\leq$3 $\wedge$ inp = Timer.timeout	2	C	2
t3	t1	false	counter = 4 $\wedge$ inp = Timer.timeout	3	C	3
t12	t1	false	inp = DR	3	C	3
t0	t3	true	inp = ICONreq	6	C	6
t1	t3	false	inp = CC	8	C	8
t2	t3	counter <= 3	counter = 3 $\wedge$ inp = Timer.timeout	2	counter$\leq$3 $\wedge$ inp = Timer.timeout	5
t11	t3	false	inp = DR	7	C	7
t3	t3	counter = 4	counter = 4 $\wedge$ inp = Timer.timeout	1	C	1
t12	t3	false	inp = DR	7	C	7
t4	t4	true	inp = 1	1	C	1
t13	t4	false	inp = DR	4	C	4
t5	t5	number = 0	inp = AK $\wedge$ num = number $\wedge$ number = 0	1	C	1
t6	t5	number = 1	inp = AK $\wedge$ num = number $\wedge$ number = 1	3	C C	3
t7	t5	false	inp = AK $\wedge$ num <> number $\wedge$ number = 0 $\wedge$ counter <=3	2	C	2
t8	t5	false	inp = AK $\wedge$ num$\neq$number $\wedge$ counter = 4	7	C	7
t9	t5	false	inp = Timer.timeout $\wedge$ number = 0 $\wedge$ counter <=3	2	C	2
t10	t5	false	inp = Timer.timeout $\wedge$ counter = 4	7	C	7
t14	t5	false	inp = DR	7		7
t5	t6	number = 0	inp = AK $\wedge$ num = number $\wedge$ number = 0	3	C	3
t6	t6	number = 1	inp = AK $\wedge$ num = number $\wedge$ number = 1	1	C	1
t7	t6	false	inp = AK $\wedge$ num$\neq$number $\wedge$ number = 1 $\wedge$ counter $\leq$3	2	C	2

t8	t6	false	inp = AK $\wedge$ num$\neq$number $\wedge$ counter = 4	5	C	5
t9	t6	false	inp = Timer.timeout $\wedge$ number = 1 $\wedge$ counter $\leq$3	2	C	2
t10	t6	false	inp = Timer.timeout $\wedge$ counter = 4	5	C	5
t14	t6	false	inp = DR	5	C	5

Table 7: Creating a path to reach the transition t3 from the state Disconnected

Step	(State, counter, number)	Via	To	C^g	C	C*	Next
1	(Disconnected,_,_)	t0 t11	t3 t3	$true$	ICONreq		t0
2	(Waiting,0,_)	t3 t2 t1 t12	t3 t3 t3 t3	$false$ $true$	UNSAT	Timer.timeout	t2
	...						
5	(Waiting,3,_)	t3 t2 t1 t12	t3 t3 t3 t3	$false$ $true$	Timer.timeout		t2
6	(Waiting,4,_)	t3 t2 t1 t12	t3 t3 t3 t3	$true$	Timer.timeout		t3

Table 8 demonstrates the use of the generated constraints for guiding the IUT along the path $< t0; t1; t4; t6; t4; t5 >$. This path is particularly difficult to achieve with random testing (Derterian, Hierons, Harman, & Guo, 2010), but it is straightforward using the proposed method.

The off-line calculation of the constraints for all-transitions test goal expressed by 14 traps took 54 seconds on 2 GHz computer and involved 1744 calls to underlying solver. The constraint solving for on-line data generation is fast, being 0.026 seconds in average and comprising mainly of input-output operations for such a simple constraints. The on-line solving time has been in the same order of magnitude in other case studies that include considerably larger constraints having thousands of subformulas.

5.6 Handling complexity

The complexity of on-line constraint solving is a critical factor in testing time critical systems. Strictly bounded reaction time in test execution is a restriction that forces to pay contribution to test run-time planning quality. Several heuristics can be used for guiding the selection of test paths at run-time, e.g., anti-ant search strategy (Nachmanson, Veanes, Schulte, Tillmann, & Grieskamp, 2004), in (Derderian, Hierons, Harman, & Guo, 2010) fitness function is computed for EFSM IUT models, in (Vain, Raiend, Kull, & Ernits, 2007) a control graph based gain function is proposed and its usability discussed. To address the scalability problem of the method proposed in Section 5.1, we propose to extend the offline test data constraint construction technique so that it takes explicitly into account its run-time execution time bound. The data constraint construction method describe in Section 5.3 is incremental in the sense that the algorithm extends the global path constraint step-wise by backward search starting from the traps. Thus, it is natural to assume that the size of path constraint increases monotonously along the model traversal process. One can calculate the constraint complexity by the length of the constraint formula but that gives only indirect characterization of its on-line solving time. Therefore, instead of formula based complexity estimation we evaluate the data constraint on-line solving time iteratively at each constraint construction step. The constraints are solved with arbitrary data values since due to the constraint shape used its valuation time is roughly independent of its variable valuation. As a result of iterative evaluation of the path constraint for a trap, its construction stops when the constraint solving time exceeds the test reaction time limit.

Table 8: Executing the transition path t0;t1;t4;t6;t4;t5

Step	(State, counter, number)	Via	To	C^g	C	C*	Next
1	(Disconnected,_,_,)	t0	t0	$true$	ICONReq		t0
		t11	t0				
2	(Waiting,0,_)	t1	t1	$true$	CC		t1
		t2	t1				
		t3	t1				
		t12	t1				
3	(Connected,0,1)	t4	t4	$true$	IDATReq		t4
		t13	t4				
4	(Sending,0,1)	t6	t6	$true$	AK(1)		t6
		t7	t6				
		t9	t6				
		t5	t6				
		t8	t6				
		t10	t6				
		t14	t6				
5	(Connected,0,0)	t4	t4	$true$	IDATReq		t4
		t13	t4				
6	(Sending,0,0)	t5	t5	$true$	AK(0)		t5
		t7	t5				
		t9	t5				
		t6	t5				
		t8	t5				
		t10	t5				
		t14	t5				

The given heuristics alone provide only partial information for on-line planning since every trap may not be “visible” (the data constraint grows over complexity limit) for some states of the IUT model. In order to avoid those “blind” states regarding traps with too complex data constraints, we use partial (and more concise) guiding information in the form of control graph based distance estimation from given state to the states where the full data constraint to “unseen” trap is present. Similarly to (Vain, Raiend, Kull, &

Ernits, 2007) the weaker knowledge about the trap reachability is encoded in the gain functions that allow evaluating the control graph distances and based on that, the best directions to the states where the full data constraint is defined for a targeted trap. Procedurally, both explicit and implicit knowledge for on-line planning are computed in two waves: (i) the full data constraint for each feasible path to given trap labeled transition is computed using backward constraint propagation algorithm and the propagation stops when the constraint solving time exceeds its given bound; (ii) for each trap constraint and for each IUT model state unlabelled with data constraint constructed in (i) the gain function is computed based on control graph to evaluate the most promising direction to data constraint labeled states. As shown in (Vain, Raiend, Kull, & Ernits, 2007) the gain function construction complexity is $O(|E|^3)$, ($|E|$ is the number of transitions in the tester model) and its practical usage feasible for testing embedded systems with bounded planning horizon (Kull, Raiend, Vain, & Kääramees, 2009).

6 CONCLUSIONS AND FUTURE WORK

In this chapter we proposed a model-based construction of an on-line planning tester for black-box testing of the IUT. The IUT is modelled in terms of an output observable non-deterministic EFSM. The tester synthesis was introduced at first using a restricted class of EFSM models of the IUT. That is motivated by the fact thatthere exists a transformation (although with very high complexity) from such EFSM to FSM (Henniger, Ulrich, & König, 1995). That are EFSM models were the variables have finite, countable domains and the data constraints are linear. The method comprises off-line static analysis phase and the on-line test planning and execution phase. As the result of the off-line analysis of the IUT model the data and control constraints for efficient on-line test planning are prepared. The generated constraints are used in on-line test guidance for generating IUT inputs to reach the test goal along sub-optimal paths. No costly model exploration and path finding operation is needed on-line.

As an extension of the on-line planning tester synthesis method the test data constraint construction and solving was introduced. The experiments have been made to prove the feasibility of the RPT synthesis method on a case-study where the IUT is the well-known Inres protocol. The case study showed that deriving the constraints without the need to restrict the planning horizon is feasible, solving the constraints for test data generation is very efficient and the results allow to drive the IUT along the optimal paths to fulfill the test requirements. Also a case study of the model of stopwatch having deep loop counters have been tried successfully up to 1000 steps of search depth. The current results have been obtained by a tool which builds on top of the state of the art SMT-solver Z3 (Moura & Bjørner, 2008) using it for quantifier elimination, simplification, checking satisfiability, and solving the complex constraints. The experiments provide a good indication that the proposed method has a potential for the case studies of reasonable size where the on-line testing of non-deterministic systems is needed.

Finally, the heuristics that improve the scalability of on-line planning for systems of industrial size and requiring tester’s bounded response time have been proposed. According to the heuristic, the planning horizon is determined strictly based on the time complexity of solving data constraints on-line.

6.1 Related Work

For MBT a model that represents the IUT specification is required. Finite state machine (FSM) and extended finite state machine (EFSM) are commonly used for the purpose of test case derivation (Petrenko, Boroday, & Groz, 2004). An FSM can model the control flow of a system. In order to model a system which has both control and data parts, e.g., communication protocols, an extension is needed. Such systems are represented using an EFSM model (Kalaji, Hierons, & Swift, 2009). The EFSM model has been widely studied and many methods are available which employ different test data generation approaches (Lai, 2002; Lee & Yannakakis, 1996). Nevertheless, automated test data generation from EFSM model is complicated by the presence of infeasible paths and is an open research problem (Offutt & Hayes, 1996).

In an EFSM model, a given path can be classified as either infeasible or feasible. The existence of some infeasible paths is due to the variable inter-dependencies among the actions and conditions. If a path is infeasible, there is no input test data that can cause this path to be traversed. Thus, if such a path is chosen in order to exercise certain transitions, these transitions are not exercised even if they can be exercised through other feasible paths (Kalaji, Hierons, & Swift, 2009). While the feasibility of paths is undecidable, there are several techniques that handle them in certain special cases (Offutt & Hayes, 1996; Chanson & Zhu, 1993; Hamon, Moura, & Rushby, 2004).

MBT can be applied for both off-line and on-line generation of test cases. In case of on-line testing, the test generation procedure derives only one test input at a time from the model and feeds it immediately to the IUT as opposed to deriving a complete test case in advance like in off-line testing. In on-line testing, it is not required to explore the whole state space of the model of the IUT every time the test stimulus is generated. Instead, the decisions about the next actions are made by observing the current output of the IUT (Tretmans & Brinksma, 2002). However, on-line test execution requires more run-time resources for interpreting the model and choosing the test stimulus. The on-line testing methods differ in how the test purpose is defined, how the test stimuli are selected on-the-fly, and what is the planning effort behind each choice.

The test purpose can be stated in very abstract way when applying conformance (IOCO or TIOCO) relation (Briones & Brinksma, 2004). Usually the conformance relation is tested using either a completely random or some heuristic driven state space exploration algorithm. A test stimulus at given state is selected randomly from the set of stimuli having uniform distribution of preference to trigger a next transitions of IUT model. Random choice has been used in early TorX tool (Belinfante, Feenstra, René, Vries, Tretmans, Goga, Feijs, Mauw, & Heerink, 1999), Uppaal-Tron (Larsen, Mikucionis, Nielsen, & Skou, 2005; Mikucionis, Larsen, & Nielsen, 2004) and also in the on-the-fly testing mode of SpecExplorer (Nachmanson, Veanes, Schulte, Tillmann, & Grieskamp, 2004). In (Feijs, Goga, & Mauw, 2000), also the transition probabilities directed input selection method is introduced to TorX. More restrictive are the test goal-directed exploration algorithms that reduce the total number of states to be explored in a model. The goal-directed approach is stronger than random exploration in the sense of providing guidance towards a certain set of IUT execution sequences that cover so called test goal items (e.g., states or transitions in the IUT model). The goal-directed approach was introduced in (Ferguson & Korel, 1996; Korel & Al-Yami, 1996), used in testing tool elaborated in (Vries, 2001) and later used in TorX (Tretmans & Brinksma, 2003) and TGV (Jard & Jeron, 2005).

Further advancement of test goal specification has been introduced in NModel (Veanes, Campbell, Schulte, 2007; Veanes, Campbell, Grieskamp, Schulte, Tillmann, & Nachmanson, 2008) where the IUT model presented as a model program can be composed with test scenario models to restrict the sets of test sequences. An “anti-ant” heuristic (Li & Lam, 2005) (“anti-ant” heuristic prefers least visited edges while making a graph search) based algorithm of reinforcement learning (Veanes, Roy, & Campbell, 2006) is used to cover specified test sequences in the model program.

While distinguishing the on-the-fly test input selection methods by their planning effort, e.g., by the depth of planning horizon, the simplest and fastest method is random choice. The planning horizon of the random choice is zero steps ahead. Non-zero, but still short range planning is applied in anti-ant methods that try to avoid already executed test sequences. For selecting the next stimuli from the set of possible ones the anti-ant method takes the less used transition of the model. The planning horizon of the anti-ant algorithm is just 1 step ahead. The anti-ant heuristic-based state space exploration method is used in (Larsen, Mikucionis, Nielsen, & Skou, 2005), and (Li & Lam, 2005) to cover all transitions of the IUT model.

The reactive planning testing approach introduced in this chapter is goal-directed like the methods used in TorX and TGV. The test goal in RPT can be stated also in the form of test scenario model like in NModel. For that, the tester model is constructed as synchronous parallel composition of the test scenario model and the IUT model. The test scenario model specifies the test coverage items (e.g., states, transitions), also the conditions and temporal order the coverage items need to be visited during the test run. The usage of test scenario models clearly increases the expressiveness of test goal specification language compared with simple trap set specification format. That allows stating dynamic resetting, partial order and repetitive visits of coverage items during the test run.

When comparing the on-line testing methods by their planning capability, the methods can be ordered by their online planning depth. The RPT (Vain, Raiend, Kull, & Ernits, 2007) involves a planner that looks ahead more than one step at a time to reach still unsatisfied parts of the test purpose, whereas the anti-ant approach looks only one step ahead when selecting the least visited outgoing transition from a current state. While the random walk has planning horizon 0 steps ahead and “anti-ant” just 1 step ahead the planning horizon of RPT can be parametrically tuned. That allows to “see” the existence and direction of unattended test coverage items from other states of the model though the item itself can be behind the exact planning horizon yet. Moreover, the reactive planning tester is able to guide the model on-the-fly exploration towards still unexplored areas even in cases when they are “shielded” by the parts of the model already traversed. Because of the longer planning horizon RPT can result in shorter test sequences compared to random choice and anti-ant methods.

The price to pay for shorter test cases is the complexity of the planning constraints to be solved on-line to guide the selection of test stimuli. The performance advantage of RPT over pure anti-ant and random choice methods depends on many factors, like model size and structural complexity, the degree of non-determinism, and the placement of coverage items in the model. Model size and complexity affect, in the first place, the size and complexity of the rules the RPT has to execute on-the-fly. Second, as the IUT specification model may be non-deterministic, it is impossible to predict exactly what paths appear to be more preferable in terms of the test length and how long time such test should take. Only under the fairness assumption all non-deterministic choices will be traversed eventually and the test purpose is potentially reachable. Thus, the degree of the non-determinism of the IUT model is a factor that can make the exhaustive or even deeper planning worthless. For instance, the planning may target to reach a coverage item behind some non-deterministic choice point in the IUT model but due to the non-determinism the reachability of the coverage item is not granted for the run of a given length. If there are many non-deterministic choices on the path to the next coverage item then the non-determinism can direct the test execution more likely to the paths other than those needed for reaching the test goal. Therefore, the distribution of coverage items in the model may affect the resulting test sequence length. Depending on the non-determinism handling strategy the test planning rules in RPT can prefer either potentially shorter paths to reach the test coverage items but including in the same time non-deterministic choices (“optimistic“ strategy), or alternatively, the deterministic paths that may be considerably longer but guarantee the reachability of test coverage items (“pessimistic” strategy).

Tightly related with on-line test planning is the topic of test data generation that implement the intended test stimuli. The problem has been studied initially within the context of programs and specific programming language data types. With the advent of MBT the problem has been addressed in terms of more abstract structures, e.g., EFSMs. Generally, the goal of test data generation is to find the input values to IUT that will guide the execution to reach the testing goals. This is achieved in two steps: (i) find the data constraint for some test goal related path, (ii) solve the path constraint in terms of input variables. The solution will then be a system of (in)equalities describing how input data should be formed in order to traverse the path (Edvardsson, 1999).

Depending on the test coverage criteria the test data generation method can be either random generation, generating test data for an unspecific path, or generating test data for a specific path (Ferguson & Korel, 1996). These approaches are called respectively random, goal-oriented, and path oriented test data generation. Each of these generation methods can be implemented statically or dynamically (on-the-fly). Random testing relies on probability and has quite low chances in finding faults that are revealed by a small percentage of the program input (Offutt & Hayes, 1996), and thus accomplish high coverage. Since random testing is considered to be of the lowest acceptance rate it is often used as test data generation benchmark (Chang, Carlisle, Cross II, & Brown, 1991). The goal-oriented (-directed) approach is stronger in the sense of providing guidance towards a certain set of IUT execution sequences. It generates input that traverses some of the sequences that satisfies the test goal. Since the sequences are selected arbitrarily this reduces the risk of encountering infeasible (non-traversible) sequences and provides a way to direct the search for input values as well. For instance, two goal-oriented methods using this technique have been implemented in the system (Ferguson & Korel, 1996; Korel & Al-Yami, 1996):

• Chaining approach uses data dependence to find solutions to branch predicates. For that a chain of nodes is identified that is necessary to the execution of the goal coverage item. The chain is built up iteratively during execution. Since it uses the “find-any-path” concept it is hard to predict the coverage given a set of goals.

• Assertion-oriented approach is an extension of the chaining approach. It utilizes the goal-oriented generation in the following way: certain conditions (assertions) are inserted in the code either manually or automatically. When an assertion is executed it is supposed to hold, otherwise there is an error either in the program or in the assertion. The goal of assertion-oriented generation is to find any execution sequence to violate an assertion. The advantage of this method is that the test oracle is given in the code and there is no need for calculating the test data from some other source than the code. So called path-oriented generation does not provide the test data generator with a possibility of selecting among a set of test sequences, but just one specific. It is the same as a goal-oriented test data generation, except for the use of specific sequences. This leads to a better prediction of coverage. On the other hand, due to the more strict path constraint it is harder to find the test data.

All test data generation methods listed above (except random testing) have to solve a path constraints (predicates). Due to the fact that symbolic constraint solving generally is undecidable, e.g., in case of programs with function calls, partial constraint satisfaction techniques are applied. Promising search methods are simulated annealing (Tracey, Clark, & Mander, 1998), and evolutionary algorithms (Kalaji, Hierons, & Swift, 2009) for their data type independence and iterative relaxation (Gupta, Mathur, & Soffa, 1998) for its predictability. As an alternative to static test data generation the dynamic approaches (Godefroid, Halleux, Nori, Rajamani, Schulte, Tillmann, & Levin, 2008; Derderian, Hierons, Harman, & Guo, 2010) do not suffer from undecidable constraints to the same extent as static methods. Dynamic test generation extends static test generation with additional run-time information, so it is more general and powerful. For instance, in a DART system (Godefroid, Halleux, Nori, Rajamani, Schulte, Tillmann, & Levin, 2008) directed search is applied. Each new input vector tries to force the program’s execution through some new path. By repeating this process, the directed search attempts to force the program to sweep through all its feasible execution paths, similarly to systematic testing and dynamic software model checking.

An example of combining dynamic and static test generation is white box fuzz testing used in SAGE system (Godefroid, Levin, & Molnar, 2008). SAGE extends previous dynamic testing approaches by using offline trace-based, rather than on-line, constraint generation. It handles also hard-to-control non-determinism in large target programs that makes debugging on-line constraint generation difficult. Thanks to offline analysis, constraint generation in SAGE is completely deterministic because it works with an execution trace that captures the outcome of all non-deterministic events encountered during the recorded run. As pointed out in cases of goal- and path oriented test data generation methods constructing the full path constraints, their simplification and run time solving is very complex task even in cases when they are decidable, e.g., in case of linear constraints on bounded finite data domains.

ACKNOWLEDGMENTS

This work was partially supported by the Estonian Science Foundation under grant No. 7667, and by the ELIKO Competence Centre project “System Validation and Testing”.

REFERENCES

TestCast Generator. On-line: http://motes.elvior.com/. (Accessed November 7, 2010)

DACS Gold Practice Website. MODEL-BASED TESTING. On-line: https://goldpractice.thedacs.com/practices/mbt/. (Accessed November 7, 2010)

ITEA2 project “Deplyment of Model-Based Technologies to Industrial Testing” Website. On-line: http://www.d-mint.org/. (Accessed November 4, 2010)

Belinfante, A., Feenstra, J., René, G., de Vries, R. G., Tretmans, J., Goga, N, Feijs, L. M. G., Mauw, S., & Heerink, L. (1999). Formal test automation: A simple experiment. In *IFIP TC6 12th International Workshop on Testing Communicating Systems* (pp. 179–196). Kluwer, B.V.

Brinksma, E., & Tretmans, J. (2001) Testing transition systems: an annotated bibliography. *Lecture Notes in Computer Science*, Springer-Verlag. *Vol.* 2067, 187–195.

Briones, L. B., & Brinksma, E. (2005) A test generation framework for quiescent real-time systems. In *FATES2004*, *Vol* 339. (pp. 64–78). Springer-Verlag.

Chang, K.-H., Carlisle, W. H., Cross II, J. H., & Brown, D. B. (1991) A heuristic approach for test case generation. In *CSC ’91: Proceedings of the 19th annual conference on Computer Science* (pp. 174–180). ACM.

Chanson, S. T., & Zhu, J. (1993) A unified approach to protocol test sequence generation. In *INFOCOM ’93. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future* (pp. 106–114). IEEE.

Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001) *Introduction to Algorithms*. McGraw-Hill Higher Education.

Derderian, K., Hierons, R. M., Harman, M., & Guo, Q. (2010) Estimating the feasibility of transition paths in extended finite state machines. *Automated Software Engg.*, 17(1), 33–56.

Duale, A. Y., & Uyar, M. Ü. (2004) A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Trans. Comput.*, 53(5), 614–627.

Edvardsson J. (1999) A survey on automatic test data generation. In *Second Conference on Computer Science and Engineering*, (pp. 21–28). ECSEL, Linköping.

Feijs, L. M. G., Goga, N., & Mauw, S. (2000) Probabilities in the Torx test derivation algorithm. In *2nd Workshop on SDL and MSC* (pp. 173– 188). *VERIMAG, IRISA, SDL Forum Society*.

Ferguson R., & Korel, B. (1996) Generating test data for distributed software using the chaining approach. *Inf. Software Technology*, 38(5), 343–353.

Godefroid, P., de Halleux, P., Nori, A. V., Rajamani, S. K., Schulte, W., Tillmann, N., & Levin, M. Y. (2008) Automating software testing using program analysis. *IEEE Software*, 25, 30–37.

Godefroid, P., Levin, M. Y., & Molnar, D. A. (2008) Automated whitebox fuzz testing. In *NDSS: 16th Annual Network & Distributed System Security Symposium: The Internet Society*. http://www.isoc.org/isoc/conferences/ndss/08/papers/10_automated_whitebox_fuzz.pdf (Accessed November 7, 2010).

Gupta, N., Mathur, A. P., & Soffa, M. L. (1998) Automated test data generation using an iterative relaxation method. In *In SIGSOFT 201998/FSE-6 6th ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 231–244). ACM Press.

Hamon, G., de Moura, L., & Rushby, J. (2004) Generating efficient test sets with a model checker. In *SEFM '04: Proceedings of the Software Engineering and Formal Methods, Second International Conference* (pp. 261–270). IEEE Computer Society.

Henniger, O., Ulrich, A., & König, H. (1995) Transformation of estelle modules aiming at test case generation. *8th IFIP International Workshop on Protocol Test Systems* (pp.45-60). IFIP.

Hierons, R. M., Kim, T.-H., & Ural, H. (2004) On the testability of sdl specifications. *Comput. Netw.*, 44(5), 681–700.

Hogrefe, D. (1991). OSI formal specification case study: The Inres protocol and service. *Technical Report: Vol. 91-012*, University of Bern, Switzerland.

Jard, C., & Jeron, T. (2005) Tgv: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Int. J. Software Tools Technology Transfer*, 7(4), 297–315.

Kalaji, A., Hierons, R. M., & Swift, S., (2009) A search-based approach for automatic test generation from extended finite state machine (efsm). In *TAIC-PART '09: Testing: Academic and Industrial Conference - Practice and Research Techniques* (pp. 131–132). IEEE Computer Society.

Korel, B., & Al-Yami, A. M. (1996) Assertion-oriented automated test data generation. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, (pp. 71–80). IEEE Computer Society.

Kull, A., Raiend, K., Vain, J., & Kääramees, M. (2009) Case study-based performance evaluation of reactive planning tester. In *Model-based Testing in Practice: 2nd Workshop on Model-based Testing in Practice (MoTiP 2009)*, CTIT Workshop Proceedings Series, *WP09-08*, 87–96.

Lai, R. (2002) A survey of communication protocol testing. *J. Syst. Softw.*, 62(1), 21–46.

Larsen, K. G., Mikucionis, M., Nielsen, B., & Skou, A. (2005) Testing real-time embedded software using uppaal-tron: an industrial case study. In *EMSOFT '05: 5th ACM International Conference on Embedded Software,* (pp. 299–306). ACM.

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

Li, H., & Lam, C. (2005) Using anti-ant-like agents to generate test threads from the uml diagrams. In *Testcom 2005, Lecture Notes in Computer Science, Vol 4262* (pp. 69-80). Springer-Verlag.

Luo, G., von Bochmann, G., & Petrenko, A. (1994) Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Trans. Softw. Eng.*, 20(2), 149–162.

Lyons, D. M., & Hendriks, A. J. (1992) Reactive planning. In S.C. Shaphiro (Ed.), *Encyclopedia of Artificial Intelligence, 2nd edition*, (pp. 1171–1181). John Wiley & Sons.

Mikucionis, M., Larsen, K.G., & Nielsen, B. (2004) T-uppaal: Online model-based testing of real-time systems. In *ASE '04: 19th IEEE international conference on Automated software engineering,* (pp. 396–397). IEEE Computer Society.

Moura, L., & Bjørner, N. (2008) Z3: An efficient SMT solver. In *Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, Vol*. 4963 (pp. 337–340). Springer-Verlag.

Nachmanson, L., Veanes, M., Schulte, W., Tillmann, N., & Grieskamp, W. (2004) Optimal strategies for testing nondeterministic systems. In *ISSTA04: Vol. 29. Software Engineering Notes* (pp. 55–64). ACM.

Offutt, A. J., & Hayes, J. H. (1996) A semantic model of program faults. *SIGSOFT Softw. Eng. Notes*, 21(3), 195–200.

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

Starke, P. H. (1972) *Abstract Automata*. Elsevier, North-Holland, Amsterdam.

Tracey, N., Clark, J., & Mander, K. (1998) Automated program flaw finding using simulated annealing. In *ISSTA '98: ACM SIGSOFT international symposium on Software testing and analysis*, (pp. 73–81). ACM.

Tretmans, G. J., & Brinksma, H. (2003) Torx: Automated model-based testing. In A. Hartman and K. Dussa-Ziegler, (Eds.), *First European Conference on Model-Driven Software Engineering, Nuremberg, Germany*, (pp. 31–43), University of Twente Publications.

Tretmans, J. (1999) Testing concurrent systems: A formal approach. In *10th International Conference on Concurrency Theory (CONCUR '99),* (pp. 46–65), Springer-Verlag.

Tretmans, J., & Brinksma, E. (2002) Côte de resyste: Automated model based testing. In M. Schweizer, (Ed.), *3rd PROGRESS Workshop on Embedded Systems*, (pp. 246–255), STW Technology Foundation.

Vain, J., Raiend, K., Kull, A., & Ernits, J. (2007) Synthesis of test purpose directed reactive planning tester for nondeterministic systems. In *22nd IEEE/ACM International Conference on Automated Software Engineering*, (pp. 363 – 372). ACM Press.

Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., & Nachmanson, L. (2008) Model-based testing of object-oriented reactive systems with spec explorer. In *Formal Methods and Testing, Lecture Notes in Computer Science, Vol. 4949* (pp. 39–76). Springer.

Veanes, M., Campbell, C., Schulte, W. (2007) Composition of model programs. In *FORTE 2007, Lecture Notes in Computer Science, Vol. 4574* (pp. 128–142). Springer.

Veanes, M., Roy, P., & Campbell, C. (2006) Online testing with reinforcement learning. In *Proceedings of FATES/RV*: *Lecture Notes in Computer Science, Vol.* 4262 (pp. 240–253). Springer-Verlag.

Vries, R. G. (2001) Towards formal test purposes. In G. J. Tretmans and H. Brinksma, (Eds.), *Formal Approaches to Testing of Software 2001 (FATES'01) BRICS Notes Series*: *Vol* NS-01-4, (pp. 61–76). BRICS Aarhus, Denkmark.

Williams, B. C., & Nayak, P. P. (1997) A reactive planner for a model-based executive. In *15th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1178–1185), American Association for Artificial Intelligence.