Model driven testing based on test history

Model Driven Testing Based on Test History

Isaac Corro Ramos, Alessandro Di Bucchianico, Lusine Hakobyan, and Kees van Hee

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{i.corro.ramos, a.d.bucchianico, l.hakobyan, k.m.v.hee}@tue.nl

Abstract. We consider software systems consisting of a single component running one sequential process only. We model such software systems as a special class of transition systems. We propose an exhaustive test procedure that uses the knowledge of system structure and the prior test history. Since exhaustive testing is not always feasible, we also present a statistical release procedure based on the characteristics of the system and the test history.
Keywords: transition systems; Petri nets; stopping criterion; software testing.

1 Introduction

In this paper we consider a simplified model of software systems consisting of a single component running one sequential process only, leaving the problem of considering a more realistic model with several components for future works. We model software systems as labelled transition systems (LTS). Atomic software operations are modelled as transitions that either have a correct or an erroneous behaviour. Therefore, an error is a symbolic labeling of a transition that can be discovered only when the transition is fired. Using labelled transition systems, in particular finite state machines, for testing has a long history. Some early papers in this direction are [4], [7] and [8]. A similar model, event sequence graphs, is used in [2]. For recent overviews of transition based testing, we refer to [1] and [3]. Since exhaustive testing is not always feasible in practice, our strategy can also be used as a statistical release procedure, based on a statistical stopping criterion. In spite of the extensive literature on statistical stopping criteria for black-box testing, there do not seem to be similar criteria for white-box testing (in which the structure of the software system is taken into account). Our term "statistical procedure" should not be confused with the common statistical testing techniques (see *e.g.*, [9]) used to generate test cases using a probability distribution. Our release procedure is based on the test history and focuses on the probability of having a certain number of remaining errors when we decide to stop testing. The difference with existing approaches (cf. [1]) is that we give an efficient strategy for exhaustive testing, *i.e.*, visiting all transitions, combined with a statistical procedure that allows to stop earlier. The selection of the next

transition to be tested is based on a randomized choice. So we provide a possibly exhaustive random testing strategy.

The rest of the paper is organized as follows. An example to illustrate a real application of our procedure is presented in Section 2. In Section 3 we introduce the test framework. A detailed description of the exhaustive test strategy is explained in Section 4. The statistical release procedure is described in Section 5. Finally, in Section 6 we discuss the results obtained so far and future work.

2 Example of modelling software as a workflow transition system

Before formalizing our test framework in Section 3, we illustrate with the following example how we map the abstract model to a real application. This example is taken from the web site `www.Petriweb.org`. *Petriweb* is a web application for maintaining repositories of Petri nets (a detailed description is given in [6]). We consider web applications as labelled transition systems (a formal definition will be given in Section 3). The web pages are the states and the links or buttons on the pages are the transitions. Pressing a button activates the code that performs an atomic operation (modelled as a transition) which results in moving to another page (modelled as a state). Note that the result can also be new data in a page. An error is considered as a wrong functioning link or button.

We consider a use case scenario from the user's perspective. There are several types of users such as administrator, registered user or unregistered user. Each of them can use Petriweb for different purposes. We consider the use case for a logged in user who wants to add a new net to the repository. Figure 1 shows the transition system of the "adding a new net" use case where the initial state is "logged in user screen" and the final state is "finish use case". Branching

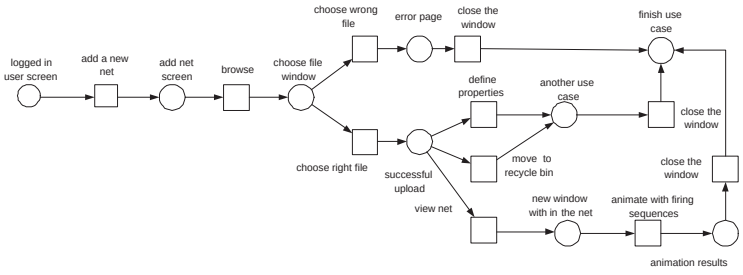

Fig. 1. Transition system of "adding a new net" use case.

points can be regarded as the options the user can choose. An execution of a path from the initial state is called run. If a run is successful, *i.e.*, we reach the final state, then one of the user's options can be skipped for the next run, as we shall explain in Section 4. There are several approaches to modelling and

testing web applications using finite state machines (see [1] for an overview). In [1], the model is similar to ours, however they consider more components (called clusters by them) for which they introduced a hierarchy of finite state machines. They consider some standard coverage strategies for exhaustive testing, but no statistically based stopping rule. We define a statistical release procedure based on the graphical structure of the model to stop testing when the probability of having a certain number of remaining errors reaches a desired bound (see Section 5 for details). For example, from the state "another use case" a new use case can start which can be seen as detailed elaboration of "define properties" actions the user can perform. In general we also have different use cases based on the user's type. Therefore, if we model all possible use cases, our transition system will be expanded. If the application is modelled using a large number of states and transitions, exhaustive testing (visit all the transitions at least once) may not be feasible anymore.

3 Modelling framework for testing

In this section we introduce the basic definitions and concepts to be used in our test procedure. As we mentioned in the introduction, we consider software systems consisting of one component running one sequential process. We use labelled transition systems, which can be seen as a subclass of Petri nets, to model such processes.

Definition 1 (Labelled transition system). *A* labelled transition system *(LTS) is a triple $L = (S, T, R)$, where*

1. *S is a non-empty finite set whose elements are called* states*,*
2. *T is a non-empty finite set whose elements are called* transitions*,*
3. *$R \subseteq S \times T \times S$ is a ternary relation such that for all $t \in T$ there exist $s, s' \in S$ such that $(s, t, s') \in R$,*
4. *there is exactly one state $i \in S$ (called the* initial state*) such that there is no triple $(s, t, i) \in R$,*
5. *there is at least one state f such that there is no triple $(f, t, s) \in R$ (such states f are called final states),*
6. *for any state $s \in S$ there is a sequence $(s_1, t_1, s_2, \ldots, s_n, t_n, s_{n+1})$ such that $(s_k, t_k, s_{k+1}) \in R$ for all $1 \leq k \leq n$, $s_1 = i$, and $s_{n+1} = s$,* i.e.*, any state is reachable from the initial one.*

When we do not want to specify the name of a transition we say that there is an *arc* from s to s'. Given an LTS, for all $s \in S$ we define $\bullet s = \{u \in S \mid \exists t \in T : (u, t, s) \in R\}$ and $s\bullet = \{v \in S \mid \exists t \in T : (s, t, v) \in R\}$ as the *preset* and the *postset* of s, respectively. If $(s, t, s') \in R$, we also write $s \xrightarrow{t} s'$. A *path* in an LTS is either the empty sequence, denoted by ε, or a sequence $p = (s_1, t_1, \ldots, s_n, t_n, s_{n+1})$ such that for all $1 \leq k \leq n$, $(s_k, t_k, s_{k+1}) \in R$. A *subpath* of a path p is a subsequence p' of p starting and ending with a state. A path is said to be *linear* if for all s_k, $1 < k < n + 1$, it follows that $|\bullet s_k| =$

$|s_k \bullet| = 1$. We say that a path $p = (s_1, t_1, \ldots, s_n, t_n, s_{n+1})$ is a *cycle* if $s_1 = s_{n+1}$. A *firing sequence* is the projection of a path p onto the set of transitions T, *i.e.*, $(t_1, \ldots, t_n)$ is a firing sequence if and only if there exist $s_1, \ldots, s_{n+1} \in S$ such that $s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots \xrightarrow{t_n} s_{n+1}$. An LTS is *acyclic* if it does not contain cycles. An LTS is said to be a *one-path* LTS if $|s \bullet| = 1$ for all non-final states s.

The next step is to present the new concepts needed for our test purposes. First we define what an error is. The main assumption is to model atomic software operations as transitions that either behave correctly or have an error. We define an error as a symbolic marking or labelling of a transition.

Definition 2 (Error). *A symbolic marking of transitions in an LTS is a function $M : T \to \{0, 1\}$ such that*

$$M(t) = \begin{cases} 1, \textit{ if } t \textit{ is error marked} \\ \\ 0, \textit{ if } t \textit{ is error free} \end{cases}$$

The set $\mathcal{D} = \{t \mid M(t) = 1\}$ is called error set*.*

This function is unknown to the tester and it is the result of a random process as it will be described in Section 5. When an error is repaired, the marking of the transition at hand is changed from 1 to 0.

In our model we discover an error if and only if we visit an error-marked transition. As soon as we discover an error, it is repaired before we continue testing. The error finding process consists of executing a path from the initial state to either a state with empty postset, to a repeated state or to an error marked transition. We refer to this path as run.

Definition 3 (Run). *Let $L = (S, T, R)$ be an LTS with a unique initial state i. A* run *σ in L is a path $(i, t_1, \ldots, s_n, t_n, s_{n+1})$. A run is said to be* successful *if $s_k \neq s_j$, for all $1 \leq k \leq n$, $1 \leq j \leq n$, $M(t_k) = 0$, for all $1 \leq k \leq n$, and either $|s_{n+1} \bullet| = \varnothing$ or there exists exactly one $1 \leq k \leq n$ such that $s_k = s_{n+1}$. A run is said to be* failure *if $s_k \neq s_j$, for all $1 \leq k \leq n$, $1 \leq j \leq n$, $M(t_k) = 0$, for all $1 \leq k < n$, and $M(t_n) = 1$. We denote by Σ the set of all runs of L.*

We now define a special test procedure called walking function.

Definition 4 (Walking function). *Let $L = (S, T, R)$ be an LTS. A* walking function *for L is a function $w : T \to [0, 1]$ such that $\sum_{t \in s\bullet} w(t) = 1$ for all non-final states s. We denote by $\mathcal{W}$ the set of all walking functions.*

Note that a walking function w can be regarded as a probability distribution on the branching points of the LTS. Initially, all the transitions are weighted with nonzero probabilities and therefore all the transitions are executable. When we are in a state, we choose the next transition to be executed by a weighted random drawing based on the walking function. After each successful run, the walking function may be updated in order to produce a new function. This new walking function assigns probability zero to some already executed transitions so that for

the next execution those transitions will not fire. Note that a zero probability transition can also be considered as a non-existing transition. The update of the walking function is done by the following procedure.

Definition 5 (Walking function update). *Let $L = (S, T, R)$ be an LTS. A function $U : \mathcal{W} \times \Sigma \to \mathcal{W}$ such that if $w(t) = 0$ for some $t \in T$, then $w'(t) = 0$, where $U(w, \sigma) = w'$, is called a* Walking Function Update *(WFU) function for w.*

So an update means that no transitions are added in most cases, but transitions may get blocked. A detailed description of both walking function and WFU function as well as a proof of exhaustiveness for the test procedure are given in Section 4.

As we already mentioned in the introduction, exhaustive testing is not always feasible in practice. Therefore we need to define a stopping criterion that allows us to stop before we have visited all transitions.

Definition 6 (Stopping set). *Let $\mathcal{H} = \{0, 1\}^*$ be the set of all finite sequences of 0 and 1. If $\prec$ denotes the proper prefix relation, a set $\mathcal{A} \subset \mathcal{H}$ is a* stopping set *if and only if*

1. *for all $a, b \in \mathcal{A}$ we have $\neg(a \prec b)$*
2. *for all $h \in \mathcal{H}$ there exists $a \in \mathcal{A}$ such that $h \prec a \vee a \prec h$*

A procedure determining a stopping set is called stopping rule.

Note that the first condition in the previous definition states that when a sequence that stops the procedure is found, that sequence is not continued. The second condition states that any sequence has a stopping moment either in the past or possibly in the future. In Section 5 we define concrete stopping sets for our testing procedure. Our test procedure consists basically of four steps: collecting visited transitions during runs, counting the error marked transitions, updating the LTS by using a walking function update and defining a stopping rule.

4 Walking strategy

In this section we first describe a general walking function update and then a more efficient update for a special subclass of LTS. After each successful run, we want to increase the probability of visiting new transitions. For that reason, for the next run we discard some already visited parts of the LTS, in such a way that the reduced system remains an LTS. We show that after a finite number of updates all the transitions are visited, so that the updating procedure is exhaustive. At the end of this section we compare the two algorithms for acyclic LTS.

4.1 Walking function update for labelled transition systems

First we give an informal description of a WFU function for LTS and successful runs. Let $L = (S, T, R)$ be an LTS with walking function w. If L is a one-path LTS, then we stop after the first successful run. Suppose that L is not a one-path LTS. Given a successful run $\sigma = (i, t_1, \ldots, s_n, t_n, s_{n+1})$ we look for the last state, say s_k, in the sequence σ with at least two outgoing arcs. Since L is not a one-path LTS such a state always exists. Note that s_{n+1} is either a state with no outgoing arcs or a state that we encounter twice in σ. We update w to a new walking function w' by setting $w'(t_k) = 0$. By setting $w'(t_k) = 0$ we avoid to run t_k the next time we reach s_k. We do the same for transitions after t_k until we reach a state with more than one incoming transition if any. The formal description of the WFU function is given in Algorithm 1. We assume that our system is not one-path. A test for this situation is trivial, and we should stop immediately. An example of the application of the WFU function is described

Algorithm 1: WFU function for an LTS and a successful run

input : $L = (S, T, R)$, $\sigma = (i, t_0, s_1, t_1, \ldots, t_n, s_{n+1})$
output: $w' = U(w, \sigma)$

```
1  Var w : T → [0, 1]
   Var s_0 := i, tail
   begin
2      tail := n; w' := w
       while (tail ≥ 0) do
3          if (|s_tail •| ≤ 1) then
4              tail := tail − 1
5          else
6              w'(t_tail) := 0, tail := −1
7          endif
8      end
9      tail := tail + 1
       while (tail ≤ n ∧ |• s_tail| = 0) do
10         w'(t_tail) := 0, tail := tail + 1
11     end
12 end
```

in Figure 2. Suppose that $(s_0, t_0, s_1, t_1, s_2, t_2, s_1)$ is a subpath of a successful run σ in L. We update w by setting $w'(t_2) = 0$ since s_2 is the last state in the run with more than one outgoing arcs. Note that this is equivalent to removing t_2 from L.

4.2 Validity of walking function update

We now study the validity of the walking function update procedure. The next result shows that the resulting system after updating w remains an LTS.

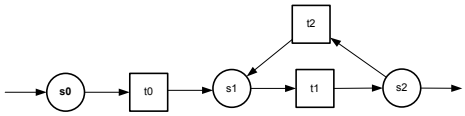

Fig. 2. LTS with a cycle. We remove t_2.

Definition 7 (Reduced system). *Let $L = (S, T, R)$ be an LTS with walking update function U. Let w be a walking function w for L and σ be a successful run. The reduced system L' w.r.t. $w' = U(w, \sigma)$ is the triple (S', T', R') such that $T' = \{t \in T \mid w'(t) > 0\}$, $S' = S \setminus \{s \in S \mid \bullet s \subset \tilde{T} \wedge s\bullet \subset \tilde{T}\}$, where $\tilde{T} = \{t \in T \mid w'(t) = 0\}$, and $R' = R \cap (S' \times T' \times S')$.*

Theorem 1. *Let $L = (S, T, R)$ be an LTS an let U be the WFU function for L as defined by Algorithm 1. If w is a walking function for L, then for any successful run σ in L there exists at least one transition t in σ such that $U(w, \sigma)(t) = 0$. Moreover, the reduced system L' w.r.t. U and w remains an LTS and after a finite number of updates we visit all transitions.*

Proof. If L is a one-path LTS, then there is nothing to prove since after a successful run we stop our procedure and we do not update w. Assume that L is not one-path and denote by $\sigma = (i, t_0, s_1, \ldots, t_n, s_{n+1})$ a successful run in L. Since L is not a one-path LTS, there exists a state s_k in σ, with $0 \leq k \leq n$, such that $|s_k \bullet| > 1$. According to Algorithm 1 we choose the last state in the sequence σ with more than one outgoing arc. We can assume without loss of generality that s_k is such a state. We consider the path $p = (s_k, t_k, s_{k+1}, \ldots, t_n, s_{n+1})$. Therefore, we update w by setting $w'(t_k) = 0$. We now prove that the reduced system, denoted by L', remains an LTS. It suffices to verify that every state x not in p is reachable from the initial one. Note that x is also a state in L' and there exists a path v from i to x in L. If p and v have no states in common, then v is also a path in L' and we are done. Now assume that p and v have at least one state in common. Suppose first that $|s_{n+1} \bullet| = 0$. Obviously x is not reachable from s_{n+1} and since s_k is the last state with two outgoing arcs in σ, s_k is the only common state of p and v. Therefore, v is a path from i to x via s_k but not via t_k. Thus, v is also a path in L'. Suppose now that $|s_{n+1} \bullet| > 0$, *i.e.*, s_{n+1} is observed twice in σ. If s_k is also in v, then two cases are possible. Either v is a path from i to x via s_k but not via t_k, in which case v is also a path in L', or v is a path from i to x via t_k which means (since s_k is the last state in σ with two outgoing arcs) that v passes through s_{n+1}. Therefore, there exists a path v' from i to x via s_{n+1} that is also a path in L'. Finally, if s_k is not in v, then there exists l with $k+1 \leq l \leq n+1$, such that $(s_l, t_l, \ldots, t_n, s_{n+1})$ is a subsequence of both p and v. In any case, s_{n+1} is also a state in v. Hence, there exists a path v' in L' from i to x via s_{n+1}. For the last statement recall that we discard at least one transition after a successful run. Failure runs may not reduce L, but since the number of error marked transitions is finite, after a finite number of runs we visit all the transitions and thus an exhaustive procedure is defined. □

Note that σ must be a successful run, otherwise Theorem 1 is not true. This condition is illustrated in Figure 3. Suppose that $(s_0, t_0, s_1, t_1, s_2)$ is a subpath of a failure run σ in L. According to Algorithm 1, we would update the walking function w by setting $w'(t_1) = 0$. However, the reduced system is not an LTS anymore because t_2 would be unreachable and, in case of being error marked, the corresponding error would never be discovered.

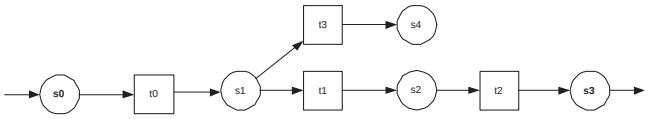

Fig. 3. Failure run in an LTS. The WFU function cannot be applied.

4.3 Walking function update for acyclic workflow transition systems

In this subsection we present a more efficient walking function update for a special subclass of LTS. Since this subclass is in fact a special class of workflow Petri nets, we call it Workflow Transitions Systems (WTS).

Definition 8 (Workflow Transition System). *A WTS is an LTS with the additional requirements that there is a unique final state f and that for every state $s \neq f$ there is a path from s to f.*

We first give an informal description of the alternative walking function update. Let $W = (S, T, R)$ be an acyclic WTS with walking function w. Given a successful run σ in W we look for the first state, say s, in σ with at least two outgoing arcs. Since W is not a one-path WTS such a state always exists. Setting s as a marker, called "head", we move forward through σ. If the next state has exactly one incoming and one outgoing arc, then we move to the following state. If we reach a state, say s', with at least two outgoing arcs but only one incoming, then we set s' as "head". We continue the same procedure until we find a state, say $\tilde{s}$, with at least two incoming arcs. Such a state always exists because there is exactly one final state, there are no cycles and the final state can be reached from any other state. We update w to a new walking function w' by setting $w'(t') = 0$, where t' is any transition in σ between s' and $\tilde{s}$. Therefore, we avoid to run again the sequence comprised between t' and $\tilde{t}$ the next time we reach s'. The formal description of this WFU function is given in Algorithm 2. An example of the application of the WFU function is shown in Figure 4. Suppose that $(b, t_0, s_1, t_1, s_2, t_2, e)$ is a subpath of a successful run σ in W. We update w by setting $w'(t_0) = 0$. Note that this is equivalent to removing t_0 from W. Similarly, in the situation illustrated in Figure 5, we update w by setting $w'(t_0) = 0$ and $w'(t_2) = 0$. If in both cases (b, t_3, s_3, t_4, e) was a subpath of σ, then we would update w by setting $w'(t_3) = 0$ and $w'(t_4) = 0$. Note that the update procedure

Algorithm 2: WFU function for an acyclic WTS and a successful run

input : $W = (S, T, R)$, $\sigma = (i, t_0, s_1, t_1, \ldots, s_n, t_n, f)$
output: $w' = U(w, \sigma)$

Var $w : T \to [0, 1]$
Var $s_0 := i, s_{n+1} := f, head, current$
begin
  $head := 0, current := 0$
  while $(current \leq (n + 1))$ **do**
    if $(|s_{current} \bullet| > 1)$ **then**
      $head := current$
    endif
    if $(| \bullet s_{current}| > 1)$ **then**
      $w(t_{head}) := 0$
    endif
    $current := current + 1$
  end
  if $(head = 0)$ **then**
    $w(t_{head}) := 0$
  endif
  for $(x = head + 1$ **to** $current)$
  $w(t_x) := 0$
end

is valid only for an acyclic WTS. Suppose that $(b, t_0, s_1, \ldots, s_4, t_4, b)$ is a cycle as it is shown in Figure 6, and subpath of σ. According to Algorithm 2 we would update w by setting $w'(t_0) = 0$. However, the reduced system is not a WTS anymore because t_5 would be unreachable.

4.4 Validity of the walking function update for acyclic WTS

Similarly to the general case, we now study the validity of the procedure for an acyclic WTS. Reduced WTS are defined in a similar way as in Definition 8.

Theorem 2. *Let $W = (S, T, R)$ be an acyclic WTS with WFU function U as defined by Algorithm 2. If w is a walking function for W, then for a successful*

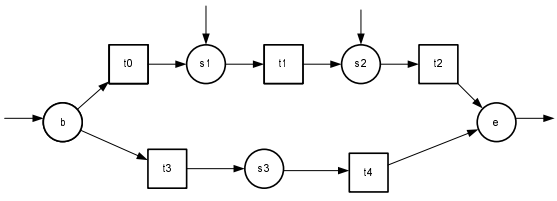

Fig. 4. Acyclic WTS where s_1 and s_2 have only one outgoing arc. We remove t_0.

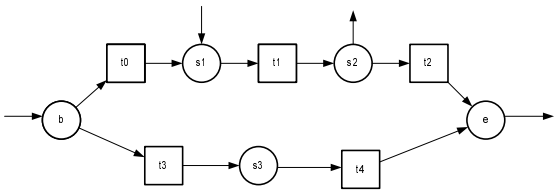

Fig. 5. Acyclic WTS where s_1 has only one outgoing arc and s_2 has two outgoing arcs. We remove t_0 and t_2.

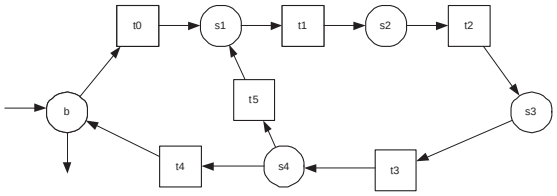

Fig. 6. Cyclic WTS. The WFU function cannot be applied.

run σ in W there exists at least one transition t in σ such that $U(w,\sigma)(t) = 0$. Moreover, the reduced system W' w.r.t. U and w remains a WTS and after a finite number of updates we visit all transitions.

Proof. If W is a one-path WTS, then there is nothing to prove since after a successful run we stop our procedure and we do not update w. Assume that W is not a one-path WTS and denote by $\sigma = (i, t_0, s_1, \ldots, t_n, f)$ a successful run in W. Since W is not a one-path WTS, there exists a state s_k in σ, with $0 \leq k \leq n$, such that $|s_k \bullet| > 1$. We can assume without loss of generality that s_k is the "head" marker in Algorithm 2 and s_l, with $k < l \leq n$, be the first state in σ after s_k with more than one incoming arc. We consider the path $p = (s_k, t_k, s_{k+1}, \ldots, t_{l-1}, s_l)$. Note that p is a linear subpath of σ. Therefore, we update w by setting $w'(t_k) = \ldots = w'(t_{l-1}) = 0$. We now prove that the reduced system, denoted by W', remains a WTS. Since we do not introduce new transitions, it is enough to verify the existence of paths to f. Consider an arbitrary state x in W that is not in p. Therefore, x is also a state in W' and there exists a path v from i to x and from x to f in W. If p is not a subpath of v, then v is also a path in W' and we are done. Suppose now that p is a subpath of v. Either p is a subpath from i to x or from x to f. Suppose that it is a subpath from i to x. Since p is a linear path and passes via s_l to x and $|\bullet s_l| > 1$, there exists at least another path from i to s_l such that p is not a subpath of it. Assume now that p is a subpath from x to f. Since $|s_k \bullet| > 1$ there exists at least another path from s_k to f that p is not a subpath of it. Therefore, W' is a WTS. The final statement follows as in the proof of Theorem 1. □

4.5 Algorithm comparison

We now illustrate with a simple example the advantage of using Algorithm 2 for acyclic WTS. We have shown in Section 4.2 that Algorithm 1 is valid for general LTS. Therefore, if we do not have any information about whether the system is acyclic or not we apply Algorithm 1. However, if we know that the system is an acyclic WTS it is more efficient to use Algorithm 2 since after a successful run it reduces at least the same number of transitions as Algorithm 1. This is depicted in Figure 7. Suppose the path $\sigma = (i, t_0, s_1, t_1, s_2, t_2, f)$ is a

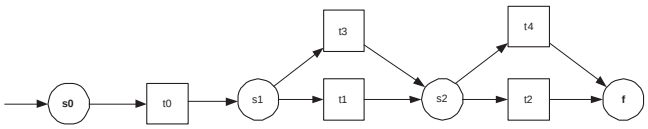

Fig. 7. Acyclic WTS. The transitions t_1 and t_2 are removed by Algorithm 2. Only t_2 is removed by Algorithm 1.

successful run. According to Algorithm 1 we update the walking function w by setting $w'(t_2) = 0$, *i.e.*, the system is reduced by one transition. Nevertheless, if we apply Algorithm 2, then we update w by setting $w'(t_1) = 0$ and $w'(t_2) = 0$, reducing thus the system by two transitions.

5 Statistical release procedure

In Section 4 we presented a procedure based on the structure of the net that allows for exhaustive testing. When exhaustive testing is not feasible in practice, statistical procedures must be considered. We present in this section a statistical release procedure that only makes use of the transitions visited during the walking phase.

Let $L = (S, T, R)$ be an LTS and write $N = |T|$. Denote by t_j, $1 \leq j \leq N$, the elements of T. The error marking process can be modelled as a Bernoulli process consisting of repeatedly performing independent identical Bernoulli trials. Our Bernoulli trial consists of selecting without replacement a transition t_j from T and labeling it as an error with unknown probability θ. Denote by ε_j, $1 \leq j \leq N$, the independent and identically distributed Bernoulli random variables representing the output of the error marking process and let e_j, $1 \leq j \leq N$, be their realizations. Then, $M(t_j) = e_j$ for all $t_j \in T$, $1 \leq j \leq N$, where M is the error marking function in Definition 2. Define $D = \sum_{j=1}^{N} \varepsilon_j$ as the random variable representing the unknown number of error marked transitions in L. D follows a binomial distribution with parameters N and θ. Suppose that the set of visited transitions $K \subset T$ is such that $|K| = n \leq N$. This is equivalent to sample (without replacement) n transitions from the set T. Formally, let $Y_1, \ldots, Y_n$ be a random sample drawn without replacement from the set of labels $\{1, \ldots, n\}$.

Therefore, t_{Y_l}, $1 \leq l \leq n$, is a new visited transition. Let X_l be the output of t_{Y_l}, *i.e.*, we can define $X_l = \varepsilon_{Y_l}$ and thus, $x_l = e_{Y_l} = M(t_{Y_l})$. Note that $X_1, \ldots, X_n$ are not independent, because $Y_1, \ldots, Y_n$ are sampled without replacement but the distribution of the errors is independent of the sampling procedure. As soon as we sample a new transition t_{Y_l}, the Bernoulli error marking experiment can be executed for this transition and the output of this experiment is the same as the one obtained executing the error marking experiment directly from T.

The next step is to develop a statistical release procedure that uses the information collected during the test phase. We determine the probability of having at most k remaining errors when we decide to stop testing. Thus, if we fix k and a confidence level α (normally 0.05), our problem consists of visiting the minimal number of transitions such that the probability of having at most k remaining errors is greater than or equal to $1 - \alpha$. We present two approaches to this problem, one based on classical statistics and one based on Bayesian statistics.

5.1 Classical approach

Denote by $s = \sum_{l=1}^{n} x_l$ the number of error marked transitions in the sample. Since D is binomially distributed with parameters N and θ, the maximum likelihood estimate of θ is given by $\hat{\theta} = s/n$. Together with the point estimate of θ we must also give a confidence interval where the true value of θ is likely to be with 95% of probability. In the worst case scenario the estimated value of θ is the upper bound of the confidence interval and it is given by $\theta_u \approx \hat{\theta} + 1.96\sqrt{\hat{\theta}(1-\hat{\theta})/n}$. Therefore, we can define the stopping set $\mathcal{A}_{\theta_u} = \{\mathbf{x} \subset \mathcal{H} \mid \mathrm{P}_{\theta_u}[s \leq D \leq s+k] \geq 1-\alpha\}$, where

$$\mathrm{P}_{\theta_u}[s \leq D \leq s+k] = \sum_{d=s}^{s+k} \binom{N}{d} (\theta_u)^d (1-\theta_u)^{N-d} \geq 1-\alpha \tag{1}$$

Note that $\mathcal{A}_{\theta_u}$ satisfies Definition 6 since as n increases to N (and thus s increases to d) the probability $\mathrm{P}_{\theta_u}[s \leq D \leq s+k]$ will always tend to 1. Thus, the stopping criterion will always be met. We will refer to the condition in $\mathcal{A}_{\theta_u}$ as the binomial rule. Note also that the value of k obtained in (1) can be considerably bigger than the one obtained using $\hat{\theta}$ instead. This and the fact that nothing can be said when $s = 0$, lead us to consider Bayesian procedures as a natural alternative.

5.2 Bayesian approach

In order to exploit the information contained in X_l, $1 \leq l \leq n$, we develop a release procedure for white-box testing considering the error probability θ as a random variable Θ with prior distribution function $F_\Theta(\theta)$ (see [5] for a similar black-box test procedure). Denote by $\mathbf{X} = (X_1, \ldots, X_n)$ the output of the n

sampled transitions and let $\mathbf{x} = (x_1, \ldots, x_n)$ be their realizations. In this case we want to calculate the minimal value of n such that

$$\mathrm{P}\left[s \leq D \leq s+k | \mathbf{X} = \mathbf{x}\right] = \sum_{d=s}^{s+k} \mathrm{P}\left[D = d | \mathbf{X} = \mathbf{x}\right] \geq 1 - \alpha \tag{2}$$

Application of the Bayes rule and the law of total probability yields

$$\mathrm{P}\left[D = d | \mathbf{X} = \mathbf{x}\right] = \frac{\mathrm{P}\left[\mathbf{X} = \mathbf{x} \mid D = d\right] \int_0^1 \mathrm{P}\left[D = d | \Theta = \theta\right] \; f_\Theta\left(\theta\right) \; d\theta}{\int_0^1 \mathrm{P}\left[\mathbf{X} = \mathbf{x} | \Theta = \theta\right] \; f_\Theta\left(\theta\right) \; d\theta} \tag{3}$$

To calculate the probabilities in (3) note that $\mathrm{P}\left[\mathbf{X} = \mathbf{x} | D = d\right]$ is the result of a hypergeometric experiment where the order is taken into account, the random variable $D|_{\Theta=\theta}$ is binomially distributed with parameters N and θ and finally $\mathrm{P}\left[\mathbf{X} = \mathbf{x} | \Theta = \theta\right]$ is the result of a binomial experiment where the order is taken into account. Substitution in (3) yields

$$\mathrm{P}\left[D = d | \mathbf{X} = \mathbf{x}\right] = \binom{N-n}{d-s} \frac{\int_0^1 \theta^d (1-\theta)^{N-d} f_\Theta(\theta) \; d\theta}{\int_0^1 \theta^s (1-\theta)^{n-s} f_\Theta(\theta) \; d\theta}$$

Since all the probabilities in (2) can be computed, we can define the stopping set $\mathcal{A} = \{\mathbf{x} \subset \mathcal{H} | \mathrm{P}\left[s \leq D \leq s+k | \mathbf{X} = \mathbf{x}\right] \geq 1 - \alpha\}$. Note that again $\mathcal{A}$ satisfies Definition 6 and hence the stopping criterion will always be met. We will refer to the condition in $\mathcal{A}$ as the prior Bayesian rule. If we observe $\mathbf{x} = (x_1, \ldots, x_n)$, then both n and s are known and therefore, the posterior distribution of Θ can be written as $f_{\Theta|_{\mathbf{X}=\mathbf{x}}}\left(\theta\right) = C\theta^s(1-\theta)^{n-s} \; f_\Theta\left(\theta\right)$, where C is a constant given by $\int_0^1 \theta^s(1-\theta)^{n-s} \; f_\Theta\left(\theta\right) \; d\theta$. We can use the posterior distribution of Θ to update the prior. We will refer to the condition in $\mathcal{A}$ using the updated posterior distribution as the posterior Bayesian rule. This procedure of collecting data and updating the distribution of Θ can be done in several stages, defining in that way a sequential procedure. Note that using a Bayesian approach we can compute the probabilities of interest also when $s = 0$.

The next section illustrates with a simple example the ideas described before.

5.3 Example

We consider for this example an LTS consisting of $N = 3000$ transitions. We assume as prior distributions for the error probability Θ the Uniform$(0, 1)$ and the Beta$(5, 495)$ distribution. Beta and Uniform distributions are typical choices for prior distributions in Bayesian statistics. Note that assuming the Beta$(5, 495)$ as prior distribution we force the error probability to be small, in this case it is concentrated near its mean value 0.01.

First we show to what extent the choice of the prior distribution influences the results of the calculations in Section 5.2. Denote by k_U and k_B the minimal value of k such that (2) holds for $\alpha = 0.05$, for uniform and beta prior distributions, respectively. Table 1 shows that k_B is smaller than k_U. This is intuitively unsurprising, since the expected posterior value of Θ (last two columns of Table 1) is smaller in case of prior beta distribution (and therefore, fewer errors are expected). We observe that the difference between k_B and k_U is larger for small values of n. Nevertheless, as n approaches to N both k_U and k_B converge to the same value. In both cases, to certify error freeness approaches exhaustive testing. If for example our quality criterion consists of accepting at most $k_U = 20$ remaining errors, with probability 0.95, we know that testing 2073 (69.1% of the total) transitions the stopping criterion met (which differs in almost 30% from exhaustive testing). Since the cost of testing is usually high, we can decide whether to stop testing at this point or not. Of special interest is the case where

Transitions (n)	Errors (s)	k_U	k_B	$\mathrm{E}[\Theta_U \mid \mathbf{X} = \mathbf{x}]$	$\mathrm{E}[\Theta_B \mid \mathbf{X} = \mathbf{x}]$
500	6	61	45	0.0139	0.011
1200	12	32	29	0.0108	0.01
2073	27	20	18	0.0134	0.0124
2400	35	15	14	0.0149	0.0137

Table 1. Minimal value of k such that (2) holds, and expected posterior value of Θ for prior Uniform$(0, 1)$ and Beta$(5, 495)$ distributions, for an LTS with $N = 3000$ transitions, given n and s.

we do not observe any errors after several runs ($s = 0$). Denote by n_U and n_B the number of error free visited transitions needed to ensure that the probability of having at most k remaining errors is at least 0.95, for uniform and beta prior distributions, respectively. Table 2 shows that in this case n_U is smaller than n_B. This can be intuitively surprising considering what we said in the previous case. However, this result can be explained looking at the posterior density function of Θ, assuming prior uniform distribution. The posterior density in this case is proportional to $(1 - \theta)^n$. Clearly, this is a strictly decreasing function that converges to its maximum when θ approaches to 0. Figure 8 shows the posterior density function of Θ assuming uniform prior distribution (left) and beta prior distribution (right) for the first three values of n_U in Table 2. Note that in both cases when n increases the density function becomes higher and its mean value is shifted to the left, *i.e.*, the expected posterior value of Θ decreases when n increases. For the uniform prior distribution θ approaches to 0 rapidly and therefore, fewer errors are expected. As in the previous case, to certify an error free system leads to exhaustive testing. In case of uniform prior distribution, exactly 95% of the transitions have to be consecutively visited without finding any error to declare that the system is error free (with probability 0.95). In case of beta prior 98.8% is required. According to this example we can conclude that in

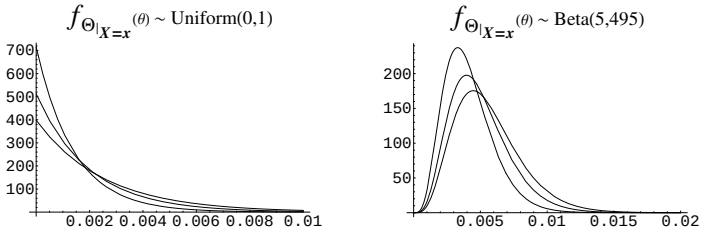

Fig. 8. Posterior density functions of Θ assuming uniform and beta prior distributions, when $s = 0$ and $n = 397$, 511 and 714.

Errors (k)	n_U	n_B
20	397	653
15	511	903
10	714	1287
5	1178	1937
1	2329	2780
0	2850	2965

Table 2. Error free transitions needed to have at most k remaining errors with probability at least 0.95, for an LTS with $N = 3000$ transitions, $s = 0$ and prior distributions Uniform$(0, 1)$ and Beta$(5, 495)$.

case of finding errors during testing the results assuming beta prior distribution outperforms the ones assuming uniform prior while the opposite holds when no error is found during testing.

We now compare the results obtained using the binomial rule in (1) and the Bayesian rules in (2). In the following example we only assume prior uniform distribution for Θ. Figure 9 shows the probability of having at most k remaining errors using the binomial rule (1), represented by dots, and the Bayesian posterior rule (2), represented by squares, respectively, when we observe $s = 16$ and $n = 1500$. Clearly, the value of k needed to meet the stopping condition is smaller in case of using the Bayesian rule. Table 3 shows the value of k obtained applying the prior, the posterior and the binomial stopping rules, respectively. Note that

Data	Prior	Posterior	Binomial
$n = 1500,\ s = 16$	$k = 27$	$k = 25$	$k_u = 43$
$n = 1500, s = 0$	$k = 4$	$k = 3$	–

Table 3. Values of k obtained using the prior, the posterior and the binomial stopping rules.

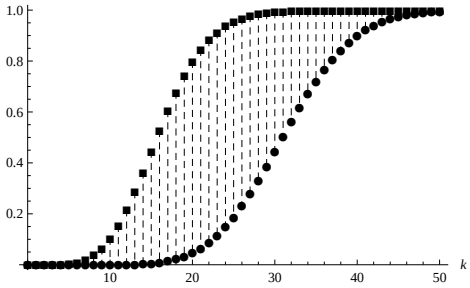

Fig. 9. Probability of having at most k remaining errors using the Bayesian posterior rule (squares) and the binomial rule (dots) using Uniform$(0, 1)$ prior distribution when $s = 16$ and $n = 1500$.

in case we do not observe any errors during testing, only the Bayesian stopping rules are suitable since the binomial rule does not provide any information. The difference between the values obtained using the prior and the posterior rules are not very significant in this example. However, this is due to the fact that the posterior rule updates the value of the posterior distribution of Θ just once. If the rule is applied sequentially using the information collected during the test, the results of the posterior rule will also be improved.

6 Conclusion and future work

We have presented a test procedure for simplified software systems consisting of a single component running one sequential process only. Such software systems are modelled as transition systems. We have defined an exhaustive procedure that uses the knowledge of the structure of the system and the results of prior testing. However, since exhaustive testing is not always feasible in practice, we have presented a statistical stopping criterion consisting of accepting with certain confidence a maximum number of remaining errors in the system.

There are many natural extensions to this work. We intend to study a walking function update procedure for general nets instead of sequential ones. We will consider errors located in the direction of the arcs or in the states, or that they are input dependent, meaning that for one input the transition can show an error but for another input the transition functions correctly. We can also introduce different kind of errors or correlations between them. Restarting the run not from the initial state but from the error marked transition is also a possible extension. These extensions must be also statistically modelled in order to define new statistical procedures.

References

1. A. Andrews, J. Offutt, and R. Alexander. Testing web applications by modeling with FSMs. *Software Syst. Model.*, 4(3):326–345, 2005.

2. F. Belli, C. Budnik, and L. White. Event-based modelling, analysis and testing of user interactions: approach and case study. *Software Testing, Verification and Reliability*, 16(1):3–32, 2006.
3. E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In F. Cassez et al., editor, *Modelling and Verification of Parallel Processes: 4th Summer School, MOVEP 2000*, volume 2067 of *Lecture Notes in Computer Science*, pages 187–195. Springer, Berlin, 2001.
4. T. Chow. Testing software designs modeled by finite-state machines. *IEEE Trans. Softw. Eng.*, 4(3):178–187, 1987.
5. A. Di Bucchianico, J. F. Groote, K. v. Hee, and R. Kruidhof. Statistical Certification of Software Systems. *Comm. Stat. C*, 37(2):To appear, 2008.
6. R. Goud, K. v. Hee, R. D. J. Post, and J. M. E. M. v. d. Werf. Petriweb: A repository for Petri nets. In S. Donatelli and P. S. Thiagarajan, editors, *27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, volume 4026 of *Lecture Notes in Computer Science*, pages 411–420, Berlin, 2006. Springer.
7. W. Howden. Methodology for the generation of progarm test data. *IEEE Trans. Softw. Eng.*, 24:208–215, 1975.
8. J. Huang. An approach to program testing. *ACM Comp. Surveys*, 7(3):113–128, 1975.
9. P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet. Software statistical testing. In B. Randell, J. C. Laprie, H. Kopetz, and B. Littlewood, editors, *Predictably Dependable Computing Systems*, pages 253–272. Springer, 1995.