Tri-Modal Under-Approximation for Test Generation

Hadrien Bride, Jacques Julliand, Pierre-Alain Masson

Tri-Modal Under-Approximation for Test Generation

Hadrien Bride Jacques Julliand
Pierre-Alain Masson
FEMTO-ST/DISC, Université de Franche-Comté
16, route de Gray F-25030 Besançon Cedex France
{bride, julliand, masson}@femto-st.fr

SAC 2015, 30th ACM/SIGAPP Symposium On Applied Computing, Salamanca, Spain, April 2015, doi: 10.1145/2695664.2695731

Abstract

This paper presents a method for under-approximating behavioural models with the guarantee that the abstract paths can be instantiated as executions of models. This allows a model-based testing approach to operate on an abstraction of infinite or very large behavioural model. We characterize the abstract transitions as may, must+ or must-. This allows us to benefit from Thomas Ball's result that any abstract sequence in the shape of must-*.may.must+* (a Ball chain) can be instantiated as a sequence of connected concrete transitions. We adapt Ball's work aiming at abstracting C programs to the case of event systems, where the instantiated Ball chains might not be reachable from a model's initial state. We propose as a solution to this problem to symbolically explore the set of states reachable after a finite number of steps, to identify the Ball chains that start in any of these states. By keeping track of the paths that lead to these starting states, we are able to instantiate with certainty the sequences made of the reached Ball chains with their prefix. This method improves the usual methods that often look for instantiations even though they don't exist for some sequences. Finally we show by means of preliminary experimental results that, despite the complexity of symbolic exploration, the method is able to reach many Ball chains within a small number of exploration steps.

Keywords: Model-Based Testing, Abstraction, Symbolic exploration, Over and under approximations.

1 Motivations

Infinite systems are practically out of reach for exhaustive exploration, or for test generation with selection criteria such as all states, all transitions, etc. An abstraction allows for having a finite and size limited view of an infinite or very large system. This paper presents a model-based testing [7, 28] approach that generates tests from an abstraction of a system's behavioural model. Generating tests from an abstraction requires to select some paths of the abstraction and to concretize them so as to obtain executions of the model.

Predicate abstraction [15] is usually used for verifying programs as in [12, 5]. Its principle is to map the (potentially infinite) set of concrete states onto a finite number of abstract ones, by means of a set of predicates that characterizes each abstract state. This leads to over-approximations when an abstract transition between two abstract states expresses that there are concrete instances of the transition that go from the source state to the other. Such transitions are called *may*. A path of *may* transitions cannot always be concretized as a path (thus connected) of concrete transitions, because the concrete instances might be disconnected. For test generation, under-approximations are preferable to over-approximations since only feasible paths are considered, though maybe not all of them. This is adequate with the testing paradigm: check some judicious paths though not all of them. This paper focuses on computing paths of abstract transitions that are guaranteed to be instantiable as connected concrete paths. This requires more knowledge on the abstract transitions than their *may* modality.

We consider the tri-modal systems of Thomas Ball [2], which consider two additional $must+$ and $must-$ modalities for the abstract transitions. It is proved in [2] that a sequence in the shape of $(must-)^* \cdot may \cdot (must+)^*$ (let us call it a *Ball chain*) is guaranteed to be concretizable as a connected chain of concrete transitions.

We adapt [2] to perform model-based testing from event systems. Ball aims in [2] at performing control flow coverage of C programs. In event systems, contrarily to programs, the control structure is implicit and becomes abstracted by the predicate abstraction process. In this context w.r.t. [2], transforming a Ball chain into a model execution requires additional guarantee that it is reachable, by a prefix that links it to an initial concrete state of the model. In [8], we have adapted Ball's work to the case of event systems. In this paper, our main additional contribution is to compute Ball chains whose reachability is guaranteed. We propose to perform a few steps of symbolic exploration (through $must-$ transitions) of the reachable states, as a prefix to the Ball chains eventually reached. This gives a symbolic state that is added and used as the initial state of our abstraction. We use SMT solvers to concretize the prefixed Ball chains. The result is a set of instantiated executions of the model, that can be used as model based tests. The behaviours exercised depend on the adequation of abstraction predicates to test purposes. [6] proposes a method to automatically compute the abstraction predicates from a given test purpose.

The background regarding event systems, their semantics as concrete transition systems, predicate abstraction, tri-modal systems and Ball chains is given in Sec. 2. An illustrative example is specified in Sec. 3. Our main contribution is described in Sec. 4, where we present the method to compute by symbolic exploration reachable Ball chains and their prefix, and to concretize them as model executions. We also discuss the soundness of the method, give example application results and show the practical feasibility of our approach by presenting preliminary experimental results. Section 6 positions our approach w.r.t. related work. Section 7 concludes the paper and indicates further research directions.

2 Background

Our behavioural models are described as Event Systems (ES) whose semantics is defined by means of Concrete labelled Transition Systems (CTS). In this paper we describe the ES in B syntax [1] but our results are generic since they are based on transition systems. We first briefly present the syntax and the semantics of the B event systems. Then we present the concept of predicate abstraction and formalize the abstraction of event systems by means of Tri-modal Transition Systems (3MTS).

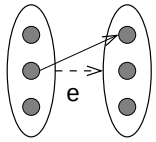
2.1 Model Syntax and Semantics

We define B event systems in Def. 1. The events are defined by means of an equation $e \mathrel{\widehat{=}} a$ where e is the name of the event and a is a generalized substitution that defines a guarded action [11] using five primitive substitutions: *skip* that is the substitution with no effect, $x, y := E, F$ that is a multiple assignment, $P \Rightarrow a$ that is a guarded substitution, $a_1 [] a_2$ that is a bounded non-deterministic choice and $@z.P \Rightarrow a$ that is an unbounded non-deterministic choice $a_{z_1} [] a_{z_2} [] \ldots$ for all the values of z satisfying the condition P. Figure 3 is an example of an event system illustrating Def. 1.

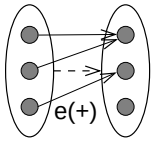
Definition 1 (Event System) *Let Ev be a set of event names. A B event system is a tuple $\langle X, I, Init, EvDef \rangle$ where:*

- *X is a set of state variables; each variable x of X has a domain defined in the invariant predicate I,*
- *$Init$ is a substitution called* initialization*, such that the invariant holds in any initial state,*
- *$EvDef$ is a set of event definitions, each in the shape of $e \mathrel{\widehat{=}} a$ for any $e \in Ev$, and such that every event preserves the invariant,*

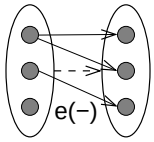
The semantics of an event system is defined in [4] as a concrete labelled transition system (CTS).



(a) A *may* Transition



(b) A *must+* Transition



(c) A *must−* Transition

Figure 1: Tri-Modal Abstract Transitions

2.2 Predicate Abstraction

Predicate abstraction [15] is a special instance of the framework of abstract interpretation [10] that maps the potentially infinite state space C of a transition system onto the finite state space A of a symbolic transition system *via* a set of n predicates $\mathcal{P} \stackrel{\text{def}}{=} \{p_1, p_2, \ldots, p_n\}$ over the state variables. The set of abstract states A contains 2^n states. Each state is a tuple $q \stackrel{\text{def}}{=} (q_1, q_2, \ldots, q_n)$ with q_i being equal either to p_i or to $\neg p_i$, and we also consider q as the predicate $\bigwedge_{i=1}^{n} q_i$. We define a total abstraction function $\alpha_{\mathcal{P}} : C \rightarrow \mathsf{A}$ such that $\alpha_{\mathcal{P}}(c)$ is an abstract state q where c satisfies q_i for all $i \in 1..n$. By a misuse of language, we say that c is in q, or that c is a state of q.

Let us now define the abstract transitions as *may*-ones. Consider two abstract states q and q' and an event e. There exists a *may* transition from q to q' by e, denoted by $q \xrightarrow{e} q'$, if and only if there exists at least one concrete transition $c \xrightarrow{e} c'$ where c and c' are concrete states with $\alpha_{\mathcal{P}}(c) = q$ and $\alpha_{\mathcal{P}}(c') = q'$ (see Fig. 1(a)).

As in [2], we define *must+* and *must−* transitions in addition to *may* ones. The *must+* transitions are *may* transitions that are triggerable from all the concrete states of the abstract source state (see Fig. 1(b)). The *must−* transitions are *may* transitions that reach all the concrete states of the abstract target state (see Fig. 1(c)). We need not enumerate all the concrete states (there might be an infinity of them) to decide if a transition is of the *must+* and/or the *must−* type: we characterize these modalities by means of SAT formulas, as is done in [8].

2.3 Tri-modal Transition Systems

We define a Tri-modal Transition System (3MTS) in Def. 2. It is a transition system with abstract states, and abstract transitions characterized as *may*, *must+* or *must−*.

Definition 2 (Tri-modal Transition System) *Let Ev be a finite set of event names and $\mathcal{P} \stackrel{def}{=} \{p_1, p_2, \ldots, p_n\}$ be a set of predicates. Let A be a finite set of abstract states defined by $\{p_1, \neg p_1\} \times \{p_2, \neg p_2\} \times \ldots \times \{p_n, \neg p_n\}$. A tuple $\langle Q, Q_0, \Delta, \Delta^+, \Delta^- \rangle$ is a 3MTS if it satisfies the following conditions:*

- *$Q (\subseteq \mathsf{A})$ is a finite set of states,*
- *$Q_0 (\subseteq Q)$ is a set of abstract initial states,*
- *$\Delta (\subseteq Q \times Ev \times Q)$ is a may labelled transition relation,*
- *$\Delta^+ (\subseteq \Delta)$ is a must+ labelled transition relation,*
- *$\Delta^- (\subseteq \Delta)$ is a must− labelled transition relation.*

The 3MTSs that we define are defined by Ball in [2]. They come from the Modal Systems defined in [20, 13]. Now, Def. 3 associates an abstraction defined by a 3MTS to an event system.

Definition 3 (3MTS associated to an ES) *Let ES be an event system and $\mathcal{P} \stackrel{def}{=} \{p_1, p_2, ..., p_n\}$ be a set of n predicates defining a set of 2^n abstract states $\mathsf{A} \stackrel{def}{=} \{p_1, \neg p_1\} \times \{p_2, \neg p_2\} \times ... \times \{p_n, \neg p_n\}$. A tuple $\langle Q,$*

$Q_0, \Delta, \Delta^+, \Delta^-\rangle$ *is a 3MTS associated to ES and* $\mathcal{P}$ *where* $Q \subseteq \mathsf{A}$, Q_0 *is the set of abstract states in* A *that contain at least an initial concrete state,* Δ, Δ^+ *and* Δ^- *are respectively the sets of* may, *must+ and must− transitions.*

An example of a 3MTS, whose ES is described in Fig. 3, can be seen in Fig. 4. The four abstract states named q_0 to q_3 appear as rounded rectangular boxes. The predicates p_0 and p_1 from which they are defined are given explicitly in Sec. 3. The abstract transitions of Δ are represented as dashed arrows labelled by an event name, with the possible mentions + and/or − indicating respectively when they are in Δ^+ or Δ^-.

2.4 Over and Under Approximations Based on 3MTS

An *execution* of a CTS or of a 3MTS is a finite or infinite sequence of transitions that begins in an initial state. We denote by $q_0 \xrightarrow{e_0} q_1 \xrightarrow{e_1} \dots$ where $q_i \xrightarrow{e_i} q_{i+1} \in \Delta$ for $i \geq 0$ an abstract execution, and by $c_0 \xrightarrow{e_0} c_1 \xrightarrow{e_1} \dots$ a concrete execution. We say that $q_0 \xrightarrow{e_0} q_1 \xrightarrow{e_1} \dots$ and $c_0 \xrightarrow{e_0} c_1 \xrightarrow{e_1} \dots$ are similar when for all i, c_i is a state of q_i.

An abstraction is an *over-approximation* of a model when, for every execution of the model, there is a similar execution of the abstraction. In other words, the abstraction may define more and/or longer executions than the model but not less. Any safety property that holds on such an abstraction also holds on the model, which allows for verifying some safety properties on the abstraction rather than on the model. But for testing, since an over-approximation may define more executions than the model, a test extracted as an execution path of the abstraction may be impossible to instantiate as a model execution.

So testing can take advantage of considering under-approxi-mations rather than over-approximations. An abstraction is an *under-approximation* of a model when for every execution of the abstraction, there is a similar execution of the model. In other words, the abstraction may define less and/or smaller executions than the model but not more. Thus every test extracted from such an abstraction is guaranteed to be instantiable on the model, to give a concrete test.

The Δ transition relation of Def. 3 defines an over-approxi-mation. Indeed, the existence of a concrete transition gives birth to a *may* abstract transition. But an execution of two consecutive *may* transitions $q \xrightarrow{e} q'$ and $q' \xrightarrow{e'} q''$ may not always have a similar connected concrete counterpart. Think for example of the case where no concrete target state of e is a concrete source state of e'.

T. Ball defines in [2] a method to compute an under-approximation by means of the Δ, Δ^+ and Δ^- abstract transition relations of a 3MTS.

2.5 Ball's Universal Under-Approximation

Thomas Ball proves in [2] that a sequence of *must−* transitions, followed by at most one *may* transition that is not a *must* one, followed by a sequence of *must+* transitions, is guaranteed to be instantiable as a connected sequence of concrete transitions. Indeed, as illustrated by the bold sequence in Fig. 2, any concrete state of a *must−* target is reached from some concrete source state, while whatever concrete state is reached by a *must+* transition is possible to leave from. A *may* transition in between joins a necessarily reached state to a necessarily left one. We call such a sequence a *Ball chain* and we write it as a regular expression[1] by means of $(must-)^* \cdot may \cdot (must+)^*$. In [2], Ball defines its under-approximation as the set of abstract states reachable from an abstract initial state by a Ball chain.

3 Illustrative Example

We introduce in this section a simple computational model in an illustrative purpose. Its small size allows an exhaustive graphical representation of the application of our method to it in Sec. 5.1. Its state space is infinite.

[1] Since a *must* transition is also *may*, we see the central *may* transition as mandatory.

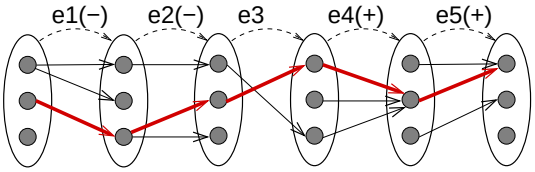


Figure 2: A Concretization of a $(must-)^* \cdot may \cdot (must+)^*$ Sequence of Abstract Transitions

The specification is given in Fig. 3. It models a conditional computation over three variables $x,\ y,\ z$. Its semantics is an infinite CTS for unbounded integers. Our abstraction method computes the finite 3MTS of Fig. 4 from the set of predicates $\mathcal{P}_0 = \{p_0, p_1\}$ where $p_0 \stackrel{\text{def}}{=} z = 1$ and $p_1 \stackrel{\text{def}}{=} x > y$.

$$
\begin{array}{lcl}
X & \widehat{=} & \{x, y, z\} \\
I & \widehat{=} & x \in \mathbb{N} \wedge y \in \mathbb{N} \wedge z \in 0..1 \\
Init & \widehat{=} & x, y, z := 0, 0, 0 \\
e_1 & \widehat{=} & z = 1 \wedge x > y \Rightarrow \\
 & & @a.(@b.(a \in \mathbb{N} \wedge b \in \mathbb{N} \wedge b \geq a \Rightarrow x, y := a, b)) \\
e_2 & \widehat{=} & z = 1 \wedge y \geq x \Rightarrow x := y + 1 \\
e_3 & \widehat{=} & z = 1 \wedge x = 7 \wedge y = 11 \Rightarrow x := 17 \\
e_4 & \widehat{=} & z = 0 \Rightarrow \\
 & & @a.(@b.(a \in \mathbb{N} \wedge b \in \mathbb{N} \wedge b < a \Rightarrow x, y, z := a, b + 5, 1)) \\
e_5 & \widehat{=} & z = 0 \Rightarrow \\
 & & @a.(@b.(a \in \mathbb{N} \wedge b \in \mathbb{N} \wedge b < a \wedge b \leq 5 \Rightarrow x, y, z\ := a, b, 1))
\end{array}
$$

Figure 3: A Small Illustrative Model

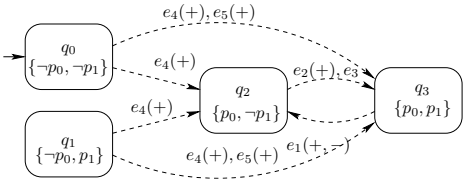


Figure 4: An Abstraction for the Small Model of Fig. 3

4 Instantiation Method for the Tri-Modal Under-Approximations

In [2], Ball performs control flow coverage of C programs, and his abstraction predicates do not abstract the program counter, that is kept explicit in the abstraction. By contrast in our case, as the control structure of an event system is implicitly defined by the guards of the events, it depends on any state variable. Since the tester defines its abstraction predicates from test purposes, intended at exercising targeted functionalities, he can abstract the control flow. In this framework the Ball chains, though concretizable, may not be reachable: the initial abstract state may include other concrete states than the initial ones. This does not occur with programs where the unabstracted program counter guarantees all executions to begin in an initial abstract state. We propose in this section to symbolically explore the reachable abstract states, in order to detect those that start a Ball chain. The states reachable from a set of states Q after a number n of event applications are characterized as an initial abstract state $R_Q(n)$. We add it to the 3MTS with Q being the states reached by initialization. The Ball chains that start in it are guaranteed to be concretizable as tests. We also describe in this section how we instantiate these prefixed Ball chains as model executions, and discuss how they can be played as tests. Then we discuss the soundness of the method and illustrate its application to the example of Sec. 3.

4.1 Symbolic Execution from the Initial States

In order to expand the set of initial states, we use static symbolic execution [19]. For that we define $R_Q(n)$ in Def. 4, where $sp(a, q)$ refers to the *strongest postcondition* [11] of an action a from a source state defined by a predicate q. It is the smallest set of states reached by the execution of a from a state that satisfies q.

Definition 4 *Let $R_Q(n)$ be the set of states of an Event System reachable from the set of states Q after applying a maximum of $n \geq 0$ event(s). This set is characterized by the following predicate:*
$R_Q(0) = Q$,
$R_Q(i+1) = R_Q(i) \vee \bigvee_{\{a|e \widehat{=} a \in EvDef\}} \mathrm{sp}(a, R_Q(i))$.

$R_{Q_0}(n)$ is the set of all the abstract states reachable from the initial states of an event system after applying at most n events. We define in Def. 5 a data structure called *reachable state tree*, denoted as RST_n, to store $R_{Q_0}(n)$ with the detail of the symbolic execution paths that lead to any abstract state of $R_{Q_0}(n)$.

Fig. 5 shows the RST_1 of the ES of Fig. 3. Each node appears as a rounded rectangular box featuring the node label and its characteristic predicate.

Definition 5 (Reachable State Tree of an ES) *A Reachable State Tree (RST_n) of an $ES \stackrel{def}{=} \langle X, I, Init, \{e \widehat{=} a | e \in Ev\}\rangle$ is a directed acyclic graph $\langle L, l_0, R, C\rangle$ where:*

- *L is a non-empty set of nodes,*
- *$l_0 \in L$ is the root node,*
- *$R \in L \times Ev \times L$ is a labelled transition relation that links a father to its children,*
- *$C \in L \times F(X)$ (where $F(X)$ is the set of first order logic formulas over the set of variables X) is a one-to-one relation that associates any node of L with a predicate characterizing a set of concrete states of the ES.*

Computing RST_n can simply be obtained by a depth-first application of each event until depth n. The cost of this computation is exponential in the depth n. With this structure, we define the set $R_{Q_0}(n)$ by $\cup\{p \mid \exists l \in L \text{ s.t. } (l, p) \in C\}$. We can recover the symbolic execution path of any state in $R_{Q_0}(n)$ by traversing the tree from the root node l_0 to the abstract state(s) of L containing it.

4.2 Test Computation and Instantiation Method

We now describe how to compute symbolic test sequences (i.e sequences of abstract transitions starting by the initialization of an ES), and instantiate them.

We characterize $R_{Q_0}(n)$ as a predicate p_{d_0}. It characterizes an abstract state q_{d_0}, that we add to the 3MTS of the ES, and make it its unique initial state. We link it to the other states by computing the *must*− and *may* transitions that come out of it and join them. Notice that we need not compute the *must*+ transitions since they will be considered as *may* ones when they start a Ball chain. This modified 3MTS is called an n-Derived 3MTS (n-D3MTS). Its definition is given by Def. 6. Figure 6 shows the 1-D3MTS of the 3MTS of Fig. 4, with the RST_1 of Fig. 5.

Definition 6 (n-Derived 3MTS) *Let $\langle Q, Q_0, \Delta, \Delta^+, \Delta^-\rangle$ be the 3MTS and $\langle L, l_0, R, C\rangle$ be the RST_n associated with an ES provided with a set of events $EvDef \stackrel{def}{=} \{e \widehat{=} a | e \in Ev\}$. An n-Derived Tri-modal Transition System (n-D3MTS) is a 3MTS $\langle Q_d, q_{d_0}, \Delta_d, \Delta^+, \Delta_d^-\rangle$ where:*

- *q_{d_0} is a new abstract state characterized by*

$$p_{d_0} \stackrel{def}{=} \bigvee_{q \in \{p|(l,p) \in C\}} q,$$

- $Q_d \stackrel{def}{=} Q \cup \{q_{d_0}\}$,
- $\Delta_d \stackrel{def}{=} \Delta \cup \{q_{d_0} \xrightarrow{e} q' | q_{d_0} \xrightarrow{e} q' \textit{ is a may transition}\}$,
- $\Delta_d^- \stackrel{def}{=} \Delta^- \cup \{q_{d_0} \xrightarrow{e} q' | q_{d_0} \xrightarrow{e} q' \textit{ is a must}- \textit{ transition}\}$.

We then compute, using the n-D3MTS, the set of abstract transition sequences in the shape of $(must-)^* \cdot may \cdot (must+)^*$ that start in q_{d_0}. This is done via a simple Depth-first search algorithm. It first traverses the $must-$ transitions (possibly none), then traverses if possible a may transition, and finally traverses the $must+$ transitions (possibly none). As this set can be infinite (e.g. when there is a $must-$ or a $must+$ loop), we use a parameter m that restricts the number of times an abstract transition appears in the sequence and denote this set T_m. Every abstract transition sequence of T_m can then be instantiated, which means that there exists at least one corresponding concrete transition sequence starting in an initial concrete state of the Event System. This concrete sequence is obtained by instantiating first the may abstract transition by SMT valuation. The $must-$ (resp. $must+$) transitions are then instantiated by recursive backward (resp. forward) SMT valuation. Finishing the backward instantiation of the $must-$ transitions gives a concrete state of $q_{d_0}(R_{Q_0}(n))$ that starts this instantiated sequence. Using the RST_n allows for recovering the abstract transition sequence that led to it. These transitions are also $must-$ ones (see Sec. 4.4), and are in their turn concretized by recursive backward SMT valuation. This ends in a concrete initial state of the event system. Finally, this now prefixed instantiated sequence is an execution of the event system, that exercises the behaviours isolated by the predicate abstraction.

4.3 Test Execution

These executions can be seen as scenarios that exercise the system according to the purpose emphasized by the abstraction predicates. The way an execution can be turned into a test depends on the controllability and observability of the system under test (SUT). To control the SUT, we assume that it can be instrumented so as to react to the commands as appearing in the execution of the model. We also assume that the non-deterministic choices that the system could make can be controlled, so that we can reproduce the choices made by the SMT-solver. For the system's observability, we assume that we can interpret the outputs produced by the SUT in terms of model variables values.

Let us make the strong assumption that: (i) the SUT is instrumented so that the state variables are totally observable, (ii) the events fireability as well as the non-deterministic choices of the events are totally controllable. Under these assumptions, the model executions can be directly used as a set of off-line tests. Consider a test of the illustrative example where the event e_1 is activated, and the values $a = 10$ and $b = 11$ have been chosen by the solver. In the real system, there is no guarantee that these values of a and b will actually be chosen. Playing the test thus requires that these values will be chosen by the instrumented SUT when this occurrence of e_1 is activated. The conformance of the SUT to the model can be evaluated by comparing the outputs produced by the system with the variable values predicted by the model for this execution. The test passes if they conform, otherwise it fails.

If the system cannot be completely controlled, then the executions have to be played as on-line tests: once a non-deterministic choice has been operated by the SUT, the tests are adapted dynamically. At first, by checking that the choice conforms to a choice allowed by the model. If not, the test fails. Otherwise, the solver is forced to operate the same choice, and the end of the test sequence is re-calculated accordingly. In the end, the test passes if the outputs of the system have conformed to some variable values predicted by the model, otherwise it fails.

4.4 Soundness of the Method

The soundness of our method is based on the three following properties, used for proving Prop. 1:

(i) Every abstract transition sequence in the shape of $(must-)^* \cdot may \cdot (must+)^*$ is instantiable (proved in [2] and explained in Sec. 2.5).

(ii) Every transition ($l \xrightarrow{e} l'$) in an RST_n is a $must-$ transition by definition (proved in [2]).

(iii) The node l_0 of an RST_n is the set of initial states of an ES by definition.

Property 1 *Let $\langle Q, q_{d_0}, \Delta, \Delta^+, \Delta^- \rangle$ be an n-D3MTS. Any abstract sequence $q_{d_0} \xrightarrow{e_0} q_1 \ldots \xrightarrow{e_{n-1}} q_n$ that is a Ball chain is instantiable from an initial state of the CTS that is the semantics of the ES whose the n-D3MTS is derived.*

Proof Any abstract sequence $q_{d_0} \xrightarrow{e_0} q_1 \ldots \xrightarrow{e_{n-1}} q_n$ that is a Ball chain can be instantiated as a concrete sequence $c_{d_0} \xrightarrow{e_0} c_1 \ldots \xrightarrow{e_{n-1}} c_n$ according to (i), but c_{d_0} is not necessarily an initial concrete state.

The initial abstract state q_{d_0} is the union of all the nodes $l_i \in L$ of the RST_n used to construct the n-D3MTS. So there exists $l_i \in L$ such that $c_{d_0} \in l_i$.

By definition of an RST_n, for all $l_i \in L$, there exists a sequence $l_0 \xrightarrow{e_{l_1}} l_1 \ldots \xrightarrow{e_{l_i}} l_i$ that is a sequence in the shape of $(must-)^*$ according to (ii). According to (i), this sequence can also be instantiated as a concrete sequence $c_{l_0} \xrightarrow{e_{l_1}} c_{l_1} \ldots \xrightarrow{e_{l_i}} c_{d_0}$ where c_{l_0} is an initial concrete state. By concatenating both these concrete sequences, we obtain a concrete sequence starting from an initial concrete state of the ES according to (iii).

5 Illustrative Application and Experimentation of the Method

We illustrate in this section our method by applying it to the small computational model of Sec. 3. We also give preliminary experimental results that were obtained by means of an experimental research prototype, that we have implemented in a proof-of-concept perspective.

5.1 Application to the Example

Suppose we want to test the events e_1 and e_2 of the ES of Fig. 3. We can use the atomic predicates that appear in their guards as the set of abstraction predicates. It is the set $\mathcal{P}_0$.

We first construct the RST_n for $n = 1$ that is graphically shown in Fig. 5. Then the 3MTS of Fig. 4 is computed. Due to the exponential complexity of the construction of RST_n, we have limited n to 1 for the sake of our illustration's readability. However, the instantiation step presented hereunder already shows the benefits of our method with such a small value of n.

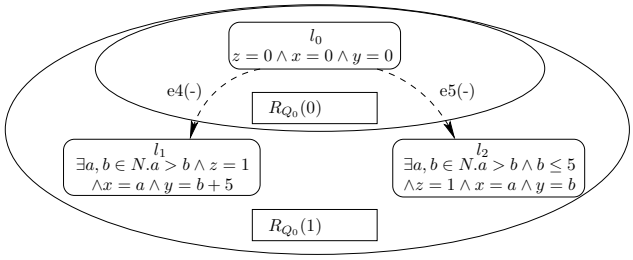


Figure 5: Graphical Representation of RST_1

Finally we construct the 1-D3MTS (see Fig. 6) from the 3MTS of Fig. 4 and the RST_1 of Fig. 5.

Let us now compute the set of abstract transition sequences in the shape of $(must-)^* \cdot may \cdot (must+)^*$ that start by q_{d_0}. With at most one cycle (i.e. $m = 2$) these sequences are the following:

$$q_{d_0} \xrightarrow{e_1-} q_2 \xrightarrow{e_3} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3,$$
$$q_{d_0} \xrightarrow{e_1-} q_2 \xrightarrow{e_2} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3,$$
$$q_{d_0} \xrightarrow{e_2} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3,$$
$$q_{d_0} \xrightarrow{e_3} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3,$$

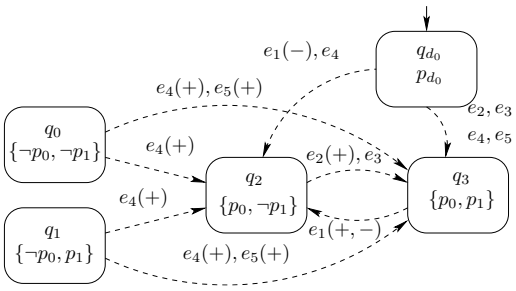


Figure 6: Graphical Representation of the 1-D3MTS Obtained from Fig. 4

$$q_{d_0} \xrightarrow{e_4} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2,$$
$$q_{d_0} \xrightarrow{e_4} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3,$$
$$q_{d_0} \xrightarrow{e_5} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3.$$

We observe, as expected by the predicates chosen (from the guards of e_1 and e_2), that each sequence applies mainly the events e_1 and e_2 in various contexts. The sequences are finite, and of the maximal length allowed by the limitation of the number of times a cycle is used.

Any of these sequences can be instantiated as described in Sec. 4.2. We illustrate it by instantiating (with the models as returned by the SMT solver) the following sequence, that is a reduction of the first sequence of the previous suite, obtained with $m = 1$: $q_{d_0} \xrightarrow{e_3} q_3 \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3$.

We first instantiate the *may* abstract transition (i.e. $q_{d_0} \xrightarrow{e_3} q_3$): $\{z{=}1,x{=}7,y{=}11\} \xrightarrow{e_3} \{z{=}1,x{=}17,y{=}11\} \xrightarrow{e_1+} q_2 \xrightarrow{e_2+} q_3$.

We then use the last instance previously obtained (i.e. $\{z = 1, x = 17, y = 11\}$) to instantiate forwardly the following *must*+ transitions (i.e. e_1 and e_2): $\{z{=}1,x{=}7,y{=}11\} \xrightarrow{e_3} \{z{=}1,x{=}17,y{=}11\} \xrightarrow{e_1+} \{z{=}1,x{=}28,y{=}52\} \xrightarrow{e_2+} \{z{=}1,x{=}53,y{=}52\}$.

N.B. The values given here in the application of the event e_1 result from the choice of the a and b values ($a = 28$ and $b = 52$) as performed by the solver during our experience. These are the values with which a completely controllable SUT should be piloted to replay this choice.

Then the first instance in the sequence is used to instantiate backwardly the preceding *must*− transitions (none here).

Finally we need to instantiate the concrete transitions leading from one of the initial states of the ES to the beginning of our previously obtained transition sequence. Thanks to the RST_1 showed in Fig. 5, we find that the concrete state $\{z = 1, x = 7, y = 11\}$ belongs to l_1 and can be reached through the following sequence: $\xrightarrow{init} l_0 \xrightarrow{e_4-} l_1$.

This sequence is instantiated backwardly as (with the values $a = 7$ and $b = 6$ chosen by the solver):

$$\xrightarrow{init} \{z = 0, x = 0, y = 0\} \xrightarrow{e_4-} \{z = 1, x = 7, y = 11\}.$$

By concatenating the last sequence with the previous one, we have a concrete sequence that begins with the initial state of the ES.

5.2 Preliminary Experimental Results

We have performed experiments with our method by means of proof-of-concept prototype software applied to two event systems more realistic than our illustrative computational model. The goal was to generate abstract tests intended to be used as off-line tests on a totally controllable and observable SUT. We have used an Intel® Core i5-2410M CPU @ 2.3 GHz in our experiments.

RST depth n	0	1	2	3
# *may*	2	6	9	9 (10)
# *must*+	2	0	0	0
# *must*−	0	0	0 (1)	2 (4)
t (sec)	0.1	0.3	0.7 (0.8)	11.5 (18.6)

Table 1: Number of Modal Transitions Issued from q_{d_0} for $\mathcal{P}_1$ (and $\mathcal{P}_2$)

RST Depth n	Iteration Number m	# Tests		# Test Steps	
		$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_1$	$\mathcal{P}_2$
0	1	2	4	6	12
0	2	2	6	10	24
1	1	7	12	24	36
1	2	7	18	38	72
2	1	10	17	35	50
2	2	10	25	55	98
3	1	19	28	87	95
3	2	26	40	195	173

Table 2: Number of Tests and of Test Steps

The first model considered is that of an Electrical System [8]. We have computed two distinct abstractions of this model, respectively w.r.t. two sets of predicates $\mathcal{P}_1$ and $\mathcal{P}_2$.

Using the set of predicates $\mathcal{P}_1$, a 3MTS with 4 abstract states, 11 *may* transitions, 5 *must*+ transitions and 5 *must*− transitions is computed in 0.6 seconds. With the set of predicates $\mathcal{P}_2$, we obtain another 3MTS with 4 abstract states, 12 *may* transitions, 5 *must*+ transitions and 5 *must*− transitions in 0.5 seconds.

Table 1 gives, according to the number n of symbolic exploration steps, the numbers of *may*, *must*+, and *must*− transitions added to the 3MTS obtained *via* the set of predicates $\mathcal{P}_1$. Table 1 also indicates the times in seconds needed to calculate them. When these numbers differ for the 3MTS obtained *via* $\mathcal{P}_2$, we indicate them between parentheses. We see that the more n grows, the more *may* and *must*− transitions are added. In contrast, the number of *must*+ transitions decreases, because the set of concrete reachable states increases with the depth n of the RST_n. Thus there are less and less chances that the transitions are fireable on all the states of this set.

Table 2 shows the number of (maximal length) abstract tests generated, and the corresponding number of test steps, w.r.t. to n, m, $\mathcal{P}_1$ and $\mathcal{P}_2$. We can see that increasing n augments the number of tests generated, because more Ball chains are reached. Consequently, the number of test steps also increases with n. Increasing m provides longer tests and can be seen as another way of obtaining more tests and test steps. We note that for both 3MTS, 100% of the abstract state as well as 100% of the abstract transitions where covered by the abstract tests generated with $n = 3$.

The second model considered is the specification of a car front wiper issued from an industrial case study. The event-B system that specifies it is composed of 15 variables and 25 events, which is significant. Also, the events are of a much greater size than in the previous examples, as they involve multiple conditional and parallel substitutions.

We obtained *via* a set of four predicates a 3MTS with 12 abstract states, 136 *may* transitions, 44 *must*+ transitions and 26 *must*− transitions in 6 minutes and 34.6 seconds. For this model we calculated $R_{Q_0}(3)$ in 17 minutes and 43 seconds and obtained 92 tests (642 test steps) with $m = 1$, which covered 66.6% of the abstract states and 42% of the abstract transitions. Notice that this coverage is as good as possible since all of the four abstract states not covered are in fact unreachable.

We observe that our method remains applicable on a real size example, and offers a good coverage of the abstract states and transitions.

6 Related Works

In [22] as well as in [23], the set of abstraction predicates is iteratively refined in order to compute a bisimulation of the initial model when it exists. None of these two methods is guaranteed to terminate,

because of the refinement step that sometimes needs to be repeated endlessly. SYNERGY [17] and DASH [3] combine under-approximation and over-approximation computations to check safety properties on programs. As we aim at proposing an efficient MBT [28] method, our algorithm always terminates because it does not refine the approximation.

The method presented in [16] applies the same two steps: computation of a predicate abstraction and generation of tests by means of a Chinese postman algorithm. The under-approximation computation is different because it does not compute a tri-modal abstraction.

Other works are about generating tests from abstraction. The tools Agatha [25], DART [14], CUTE [26], EXE [9] and PEX [27] also compute abstractions from models or from programs, but only by means of symbolic execution [24]. This data abstraction approach computes an execution graph. Its set of abstract states is possibly infinite whereas it is finite with the predicate abstraction method. The method implemented in STG [18] uses abstractions defined by the user and modelled by IOSTS (Input Output Symbolic Transition System). They use test purposes synchronized with abstractions, both defined as IOSTS. Then, the synchronized product allows generating tests after an optimization step, which consists of pruning the unreachable states by abstract interpretation. Our approach is very similar in that we also use test purposes and abstractions. But there are three differences. First, any abstraction is computed from a set of predicates defined from a test purpose and a behavioural model, whereas STG uses user-defined abstractions. Second, an optimization is performed by the abstraction computation by using the invariant properties (that do not exist in an IOSTS) specified in the B models used implicitly in our method. It allows, for the weakest precondition computation, to minimize the symbolic state space and the feasible transitions. Third, we use SMT-solvers, that combine constraint solving and theories for proof, instead of pure constraint solving to instantiate the symbolic tests.

Similarly to the concolic execution of [26], we use symbolic execution. But concolic tools do not use predicate abstraction. Concolic execution performs a concrete execution and at the same time collects the symbolic path constraints. Moreover, hybrid concolic execution [21] combines random generation of input values.

Finally, we have also used Ball's chain in [8] for generating tests from a tri-modal abstraction of an event system. Although Ball's under-approximation was combined with an existential one to try to instantiate the abstract tests generated, there was no guarantee that these tests were possible to instantiate. On the contrary in this paper, all the tests generated are guaranteed to be instantiable.

7 Conclusion and Further Works

We have presented a method for generating model-based tests from abstractions of infinite or very large behavioural models. A test is an execution of the model in this context. The abstraction is a tri-modal transition system for which the Ball chain are guaranteed to be concretizable as connected sequences of concrete (i.e. of the model) transitions. Our proposition is to select only those Ball chains that are reachable from concrete initial states. We turn them as model executions by concretizing them and computing a prefix that links them to a concrete initial state, adapting Ball's work for programs to event systems. This is performed by a symbolic exploration of the set of reachable abstract states.

We have defined a predicate that characterizes the set of states reachable in n steps, i.e. after n events have been applied. We have given a procedure to compute a symbolic execution tree, which records the successions of abstract states reached by the successive event applications. The predicate that characterizes the union of all such abstract states defines an abstract initial state that we add to the tri-modal transition system of the model. This allows us to select the Ball chains that originate from it, and prefix them with paths computed from the symbolic execution tree. We concretize each prefixed chain by SMT solving, which results in a set of model executions. They can be played as off-line or on-line model-based tests, according to the controllability and observability of the system under test. Our preliminary experimental results confirm that even small values of n quickly enhance the number of Ball chains reached and concretized as tests.

The complexity of symbolic exploration limits our computation of the reachable state space to that reachable in a few steps. Using heuristics rather than exhaustive enumeration for applying successive events

could lead us much faster towards targeted Ball chains starts. This could improve the coverage of the model by tests by discovering more reachable Ball chains for such targeted behaviours. Also, we intend to compute and combine several tri-modal systems rather than one as considered in this paper. This would allow more behaviours of the system to be tested, with one set of predicates per expected behaviour. The symbolic exploration of the reachable states would only have to be performed once, and could be added to all the tri-modal transition systems computed.

References

[1] J.-R. Abrial. *The B Book*. Cambridge Univ. Press, 1996.

[2] T. Ball. A theory of predicate-complete test coverage and generation. In *FMCO*, volume 3657 of *LNCS*, pages 1–22, 2004.

[3] N. E. Beckman, A. V. Nori, S. K. Rajamani, R. J. Simmons, S. Tetali, and A. V. Thakur. Proofs from tests. *IEEE Trans. Software Eng.*, 36(4):495–508, 2010.

[4] D. Bert and F. Cave. Construction of finite labelled transition systems from B abstract systems. In *IFM*, pages 235–254, 2000.

[5] D. Beyer, T. A. Henzinger, and G. Théoduloz. Program analysis with dynamic precision adjustment. In *ASE*, pages 29–38, 2008.

[6] F. Bouquet, P.-C. Bué, J. Julliand, and P.-A. Masson. Test generation based on abstraction and test purposes to complement structural tests. In *A-MOST*, pages 54–61, Paris, 2010.

[7] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*. Springer, 2005.

[8] P.-C. Bué, J. Julliand, and P.-A. Masson. Association of under-approximation techniques for generating tests from models. In *TAP*, volume 6706 of *LNCS*, pages 51–68. Springer, 2011.

[9] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: automatically generating inputs of death. In *ACM Conference on Computer and Communications Security*, pages 322–335, 2006.

[10] P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Log. Comput.*, 2(4):511–547, 1992.

[11] E. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. *Com. of the ACM*, 18(8):453–457, 1975.

[12] C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *POPL*, pages 191–202, 2002.

[13] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *CONCUR*, pages 426–440, 2001.

[14] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. In *PLDI*, pages 213–223, 2005.

[15] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[16] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *ISSTA*, pages 112–122, 2002.

[17] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani. Synergy: a new algorithm for property checking. In *SIGSOFT FSE*, pages 117–127, 2006.

[18] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *TACAS*, volume 3440 of *LNCS*, pages 349–364, 2005.

[19] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[20] K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.

[21] R. Majumdar and K. Sen. Hybrid concolic testing. In *ICSE*, pages 416–426, 2007.

[22] K. S. Namjoshi and R. P. Kurshan. Syntactic program transformations for automatic abstraction. In *CAV*, volume 1855 of *LNCS*, pages 435–449, 2000.

[23] C. S. Păsăreanu, R. Pelánek, and W. Visser. Predicate abstraction with under-approximation refinement. *LMCS*, 3(1), 2007.

[24] C. S. Păsăreanu and W. Visser. A survey of new trends in symbolic execution for software testing and analysis. *STTT*, 11(4):339–353, 2009.

[25] N. Rapin, C. Gaston, A. Lapitre, and J.-P. Gallois. Behavioral unfolding of formal specifications based on communicating extended automata. In *ATVA*, 2003.

[26] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *ESEC/SIGSOFT FSE*, pages 263–272, 2005.

[27] N. Tillmann and J. de Halleux. Pex-white box test generation for .net. In *TAP*, volume 4966 of *LNCS*, pages 134–153, 2008.

[28] M. Utting and B. Legeard. *Practical Model-Based Testing*. Morgan Kaufmann, 2006.