Symbolic verification and test generation for a network of communicating FSMs*

Xiaoqing Jin[1], Gianfranco Ciardo[1], Tae-Hyong Kim[2], and Yang Zhao[1]

[1] University of California, Riverside, USA
{jinx,ciardo,zhaoy}@cs.ucr.edu
[2] Kumoh National Institute of Technology, Gumi, Korea
taehyong@kumoh.ac.kr

Abstract. A network of communicating FSMs (NCFSMs) is a useful formalism to model complex concurrent systems, but its use demands efficient analysis algorithms. We propose a new symbolic framework for NCFMS verification and test generation. We explore the use of the breadth-first search (BFS) and saturation algorithms to compute the "unstable transitive closure" of transitions for the observable product machine of an NCFSM. Our framework can verify properties such as livelock freeness and includes a fully automatic test generation based on mutation analysis. Being symbolic, our framework can efficiently manage a large number of mutants with moderate resource consumption and derive a test suite to distinguish all non-equivalent first-order mutants.

1 Introduction

Concurrent systems, such as communication and multiprocessor systems, consist of several components connected via FIFO queues and can be naturally modeled as a network of communicating finite state machines (NCFSMs) where each component is a communicating finite state machine (CFSM). While the state space of an NCFSM with unbounded queues is infinite, the *slow environment assumption* [8] satisfied by most systems avoids the need to manage infinite state spaces. Our slow environment NCFSMs require a single global queue of size one.

Both structural and fault-based testing can be employed on NCFSMs. Approaches to structural testing either transform an NCFSM into a behaviorally equivalent FSM, the *observable product machine*, or try to restrict the model to allow only local transition tests, but suffer from state-space explosion [7] or require an exhaustive search to generate executable test cases [6, 9]. Fault-based testing adopts mutation analysis, which scales well in web applications and other collaborative systems, but requires dealing with a large number of mutants and must generate a distinguishing sequence for each non-equivalent mutant [13].

The lack of an efficient and fully automated verification and test derivation framework for NCFSMs limits their applicability and results in large manual testing efforts. As symbolic methods such as binary decision diagrams (BDDs) [3]

* This work was supported in part by the National Science Foundation under Grant CCF-1018057 and by a UC MEXUS-CONACYT Collaborative Research Grant.

have had great success in verification, we propose a framework for critical property verification and test generation using a symbolic strategy and mutation analysis. We use multiway decision diagrams (MDDs) [10] to compute the unstable transitive closure of the transition relation for the observable product machine, then employ verification techniques to check critical properties and provide counter-examples. Finally, we generate first-order mutants through specification mutant operators and use edge-value decision diagrams [5] to symbolically obtain distinguishing sequences (a test suite) that "kill" all non-equivalent mutants.

The remainder of this paper is organized as follows. Sect. 2 provides some background. Sect. 3 elaborates our symbolic framework and preprocessing algorithms. Sect. 4 focuses on verification algorithms and Sect. 5 on automatic test derivation algorithms. Sect. 6 gives experimental results. We conclude in Sect. 7.

2 Preliminaries

A CFSM M_k is a tuple $(\mathcal{S}_k, \mathcal{X}_k, \mathcal{Y}_k, \delta_k, \lambda_k, s_k)$ where $\mathcal{S}_k$ is a finite set of local states, $\mathcal{X}_k$ is a finite set of input symbols generated from the environment or other CFSMs, $\mathcal{Y}_k$ is a finite set of output symbols absorbed by the environment or other CFSMs, $\delta_k : \mathcal{S}_k \times \mathcal{X}_k \rightarrow \mathcal{S}_k$ is the local state transition function, $\lambda_k : \mathcal{S}_k \times \mathcal{X}_k \rightarrow \mathcal{Y}_k$ is the output function, and $s_k \in \mathcal{S}_k$ is the initial state. If $\delta_k(i,a) = j$ and $\lambda_k(i,a) = b$, we let $i \,[M_k, a/b\rangle\, j$ denote this *local* transition from state i to j caused by input a and output b in M_k.

An NCFSM M consists of K CFSMs $M_1, M_2, ..., M_K$ with pairwise disjoint sets of input symbols and a FIFO buffer β containing symbols in transit between CFSMs. The semantics of an NCFSM is defined by the *product* FSM $(\mathcal{S}, \mathcal{X}, \mathcal{Y}, \delta, \lambda, \mathbf{s}_{init})$ where $\mathcal{X} = \bigcup_{1 \le k \le K} \mathcal{X}_k$, $\mathcal{Y} = \bigcup_{1 \le k \le K} \mathcal{Y}_k$, $\mathcal{S} = (\{\epsilon\} \cup \mathcal{Y} \cup \mathcal{X}) \times \mathcal{S}_1 \times ... \times \mathcal{S}_K$ is the set of global states, $\delta : \mathcal{S} \times (\{\epsilon\} \cup \mathcal{X}) \rightarrow \mathcal{S}$ and $\lambda : \mathcal{S} \times (\{\epsilon\} \cup \mathcal{X}) \rightarrow \mathcal{Y}$ are the global state transition and output functions, respectively, which will be defined later, and $\mathbf{s}_{init} = (\epsilon, s_1, ..., s_K) \in \mathcal{S}$ is the initial global state.

Let $\mathcal{Z}_{int} = \mathcal{X} \cap \mathcal{Y}$ be the set of *internal* symbols that can appear in buffer β (we underline these symbols, e.g., $\underline{a}$). Let $\mathcal{X} \setminus \mathcal{Y} \subseteq \mathcal{X}_{ext} \subseteq \mathcal{X}$ and $\mathcal{Y}_{ext} = \mathcal{Y} \setminus \mathcal{X}$ be the set of *external input* and *external output* symbols. $\mathcal{Y}_{ext}$ contains the output symbols observable outside the system. $\mathcal{X}_{ext}$ must contain all the symbols that only the environment can place into the buffer β, thus $\mathcal{X} \setminus \mathcal{Y}$, but it may include symbols in $\mathcal{Z}_{int}$. Given $\mathbf{i} = (i_1, ..., i_K)$, define $\mathbf{i}|_{k:j_k}$ to be the vector $(i_1, ..., j_k, ..., i_K)$ obtained by setting the k^{th} component of $\mathbf{i}$ to j_k. A (global) state $(i_\beta, i_1, ..., i_K)$ is *stable* if $i_\beta = \epsilon$, we write it as $\mathbf{i}$, otherwise it is *unstable*, we write as $a.\mathbf{i}$, with $a \in \mathcal{Z}_{int}$. Let $\mathcal{S}_{st}$ and $\mathcal{S}_{unst}$ be the set of reachable stable and unstable states, respectively, and $\mathcal{S}_{rch} = \mathcal{S}_{st} \cup \mathcal{S}_{unst}$. If $i_k \,[M_k, a/x\rangle\, j_k$, λ and δ satisfy:

- $\delta(\mathbf{i},a) = \mathbf{i}|_{k:j_k}$ and $\lambda(\mathbf{i},a) = x$, if $a \in \mathcal{X}_{ext}$, $x \in \mathcal{Y}_{ext}$, written as $\mathbf{i} \,[M, a/x\rangle\, \mathbf{i}|_{k:j_k}$, or simply $\mathbf{i} \,[a/x\rangle\, \mathbf{i}|_{k:j_k}$ if M is clear from the context.
- $\delta(\mathbf{i},a) = x.\mathbf{i}|_{k:j_k}$ and $\lambda(\mathbf{i},a) = \epsilon$, if $a \in \mathcal{X}_{ext}$, $x \in \mathcal{Z}_{int}$, written as $\mathbf{i} \,[a/\underline{x}\rangle\, x.\mathbf{i}|_{k:j_k}$.
- $\delta(a.\mathbf{i},\epsilon) = \mathbf{i}|_{k:j_k}$ and $\lambda(a.\mathbf{i},\epsilon) = x$, if $a \in \mathcal{Z}_{int}$, $x \in \mathcal{Y}_{ext}$, written as $a.\mathbf{i} \,[\underline{a}/x\rangle\, \mathbf{i}|_{k:j_k}$.
- $\delta(a.\mathbf{i},\epsilon) = x.\mathbf{i}|_{k:j_k}$ and $\lambda(a.\mathbf{i},\epsilon) = \epsilon$, if $a \in \mathcal{Z}_{int}$, $x \in \mathcal{Z}_{int}$, written as $a.\mathbf{i} \,[\underline{a}/\underline{x}\rangle\, x.\mathbf{i}|_{k:j_k}$.

The NCFSMs we study conform to the slow environment assumption [8]: if the output symbol a of a CFSM can be absorbed by another CFSM as an

input symbol, then the system does not accept any other input symbol from the environment until a has been consumed. A buffer of size one is sufficient under this assumption, as β can only be empty or contain one symbol from $\mathcal{Z}_{int}$.

As neither unstable states nor the symbols in β are observable, we focus on stable state transitions: if $\mathbf{i}\,{}_{[a/\underline{a^{(1)}}\rangle}\,a^{(1)}.\mathbf{i}^{(1)}{}_{[\underline{a^{(1)}}/\underline{a^{(2)}}\rangle}\cdots{}_{[\underline{a^{(n-1)}}/\underline{a^{(n)}}\rangle}\,a^{(n)}.\mathbf{i}^{(n)}{}_{[\underline{a^{(n)}}/b\rangle}\,\mathbf{j}$, where $n \geq 1$, $\mathbf{i},\mathbf{j} \in \mathcal{S}_{st}$, $a \in \mathcal{X}_{ext}$, $a^{(1)}\cdots a^{(n)} \in \mathcal{Z}_{int}$ and $b \in \mathcal{Y}_{ext}$, we merge this sequence into a *stable transition* $\mathbf{i}\,{}_{[\![a/b]\!]}\,\mathbf{j}$. We define the observable transition function δ_{obs} and output function λ_{obs}, $\delta_{obs}(\mathbf{i},a)=\mathbf{j}$ and $\lambda_{obs}(\mathbf{i},a)=b$ if $\mathbf{i}\,{}_{[\![a/b]\!]}\,\mathbf{j}$. Then, we define the *observable product machine* M_{obs} of an NCFSM as a six-tuple ($\mathcal{S}_{st}$,$\mathcal{X}_{ext}$,$\mathcal{Y}_{ext}$,δ_{obs},λ_{obs},$\mathbf{s}_{init}$). Sect. 3 presents our symbolic algorithm to generate M_{obs}, needed to verify NCFSM equivalence and used in test derivation and test selection [12]. We let $a_1/b_1,...,a_n/b_n \in (\mathcal{X}_{ext}\times\mathcal{Y}_{ext})^*$ be a *sequence* from state $\mathbf{i} \in \mathcal{S}_{st}$ if $\lambda_{obs}(\mathbf{i},a_1a_2\cdots a_n)=\lambda_{obs}(\mathbf{i},a_1)\lambda_{obs}(\delta_{obs}(\mathbf{i},a_1),a_2\cdots a_n)=b_1b_2\cdots b_n$.

2.1 Decision diagrams

Symbolic encodings such as BDDs [3] and MDDs [10] work well for formal verification. We use MDDs to encode boolean functions for sets and EV$^+$MDDs [5] to encode partial integer functions, where ∞ means "undefined".

Given L *domain* variables v_l $(1 \leq l \leq L)$ having finite domain $\mathcal{V}_{v_l}$ and a boolean *range* variable v_0, ordered $v_L \succ \cdots \succ v_1 \succ v_0$, a (*quasi-reduced*) MDD is a directed acyclic edge-labeled graph where:

- Each node p is associated with a domain variable v_l. We write $p.v=v_l$.
- The *terminal* nodes are $\mathbf{0}$ and $\mathbf{1}$, and are the only nodes with $\mathbf{0}.v=\mathbf{1}.v=v_0$.
- A *nonterminal* node p with $p.v=v_l$ has, for each $i \in \mathcal{V}_{v_l}$, an edge pointing to node q, with $q.v=v_{l-1}$ or $q=\mathbf{0}$. We write $p[i]=q$. We must have at least one $p[i] \neq \mathbf{0}$.
- For canonicity, there are no *duplicates*: given two nonterminal nodes p and q with $p.v = q.v$, there must be at least one $i \in \mathcal{V}_{p.v}$ such that $p[i] \neq q[i]$.

A nonterminal MDD node p with $p.v=v_l$ encodes the set of tuples recursively defined by $\mathcal{B}_p=\bigcup_{i\in\mathcal{V}_{v_l}}\{i\}\cdot\mathcal{B}_{p[i]}$, with terminal cases $\mathcal{B}_{\mathbf{0}}=\emptyset$, the empty set, and $\mathcal{B}_{\mathbf{1}}=\{\epsilon\}$, the empty tuple, where "$\cdot$" indicates tuple concatenation.

To encode partial integer functions, we use a variant of the above. A normalized EV$^+$MDD [5] is a directed acyclic edge-labeled graph where:

- Ω is the only *terminal* node, with $\Omega.v=v_0$.
- A nonterminal node a with $a.v = v_l$ has, for each $i \in \mathcal{V}_{v_l}$, an edge labeled with $\rho \in \mathbb{N}\cup\{\infty\}$ pointing to node b. We write $a[i] = \langle\rho,b\rangle$, $b = a[i].node$, and $\rho = a[i].val$. We must have $b = \Omega$ if $\rho = \infty$, $b.v = v_{l-1}$ otherwise, and at least one $a[i].val = 0$.
- For canonicity, there are no *duplicates*: given two nonterminal nodes a and b with $a.v = b.v$, there must be at least one $i \in \mathcal{V}_{p.v}$ such that $a[i] \neq b[i]$, i.e., $a[i].node \neq b[i].node$, or $a[i].val \neq b[i].val$, or both.

Given EV$^+$MDD node a with $a.v=v_l$ and $\rho\in\mathbb{N}$, $\langle\rho,a\rangle$ encodes the function $f_{\langle\rho,a\rangle}:\mathcal{V}_{v_l}\times\cdots\times\mathcal{V}_{v_1}\rightarrow\mathbb{N}\cup\{\infty\}$ recursively defined by $f_{\langle\rho,a\rangle}=\rho+f_{a[v_l]}$, with base case $f_{\langle\rho,\Omega\rangle}=\rho$.

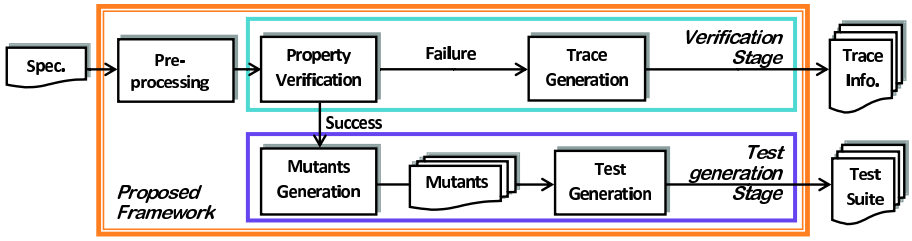

Fig. 1. System framework.

To make MDDs and EV$^+$MDDs more compact and their manipulation more efficient, edges can skip variables under various reduction rules [3, 17]. These rules still ensure canonicity and implicitly define the meaning of these "long" edges, but we do not discuss them further in the interest of clarity and brevity.

$Or(a, b)$ and $And(a, b)$ are two operators used to compute the MDD encoding $\mathcal{B}_a \cup \mathcal{B}_b$ and $\mathcal{B}_a \cap \mathcal{B}_b$. Analogously, $Min(\langle \rho, a \rangle, \langle \sigma, b \rangle)$, returns the EV$^+$MDD encoding $\min(f_{\langle \rho,a \rangle}, f_{\langle \sigma,b \rangle})$, and *Normalize* puts an EV$^+$MDD in canonical form [5].

2.2 Previous work

Structural test generation approaches for NCFSMs mostly fall into two classes. One transforms an NCFSM into its M_{obs} [12] which may encounter state-space explosion problem, then applies standard FSM test derivation techniques, such as the W-method, Wp-method, and UIO-method. However, a high complexity limits the applicability of these well-known approaches. More importantly, even if all CFSMs are deterministic, minimal, completely specified, and strongly connected, the resulting M_{obs} might not be. M_{obs} is deterministic and completely specified iff $\mathcal{M}$ is livelock-free while, if $\mathcal{Y}_{ext} \subset \mathcal{Y}$, minimal and strongly connected properties may be lost. In these cases, standard structural FSM-based test derivation algorithms are not directly applicable.

The other category of approaches [6, 9, 11] avoids building M_{obs} and uses instead branching coverage [11] or heuristic techniques [6]. These methods check local transitions instead of global transitions, and reduce testing efforts under the assumption that the system only has one fault. However, for some complex models, exhaustive searches or heuristic algorithms must be employed.

Our work falls into the first class. Our fully symbolic techniques copes with the large computational cost to generate M_{obs} for verification and test generation. Moreover, we adopt mutation analysis for test derivation, thus we do not require M_{obs} to be minimal, completely specified, or strongly connected.

3 System framework and symbolic encoding

Both specification or implementation errors can cause system failures. Our symbolic framework takes in an NCFSM model described in XML as the system specification and aims at detecting both types of errors. It has two stages: verification and test derivation, as in Fig. 1. The verification stage checks three important properties of the specification: livelock freeness, strong connectedness,

and absence of dead transitions. If a check fails, counter-examples are generated to help fixing the error. The test derivation stage generates a test suite using mutation analysis, to test the consistency between the *implementation under test* (IUT) and the specification.

Given an NCFSM with K component CFSMs and a system buffer, we use an MDD with variables $(w_K, ..., w_1, w_b)$ to encode sets of global states. The first K variables correspond to each CFSM local state, w_b corresponds to the current buffer content. A next-state function $\mathcal{T}: \mathcal{S} \times (\mathcal{X} \cup \{\epsilon\}) \rightarrow \mathcal{S} \times (\mathcal{Y} \cup \{\epsilon\})$ encoded using MDDs on $2(K+1)$ variables $(w_K, w'_K, ..., w_1, w'_1, w_b, w'_b)$, captures the global state transition function δ and output function λ, so that $\mathcal{T}(x,y) = (x',y')$ iff $\delta(x,y) = x'$ and $\lambda(x,y) = y'$, where x and x' are global states and $y, y' \in \mathcal{X} \cup \mathcal{Y} \cup \{\epsilon\}$. We define $\mathcal{T}_s = \bigcup_{1 \leq k \leq K} \mathcal{T}_k$, where $\mathcal{T}_k$ encodes the next-state function of M_k. Thus, $\mathcal{T} = \mathcal{T}_s \cup \mathcal{T}_\beta$, where $\mathcal{T}_\beta$ encodes the interaction with the environment.

Generation of the state-space $\mathcal{S}_{rch} = \{\mathbf{s}_{init}\} \cup \mathcal{T}(\mathbf{s}_{init}) \cup \mathcal{T}^2(\mathbf{s}_{init}) \cup \cdots$ is often the first step in formal verification. $\mathcal{S}_{rch}$ can be built by standard symbolic state-space generation algorithms [4] and we can split it into $\mathcal{S}_{st}$ and $\mathcal{S}_{unst}$ based on the status of the system buffer w_b: if $w_b = \epsilon$, the state is stable, otherwise it is unstable.

To compute the stable next-state function $\mathcal{T}_{obs}$ encoding δ_{obs} and λ_{obs}, we first define the *unstable transitive closure* (UTC): given $\mathcal{T}_s$, UTC is the smallest relation containing $\mathcal{T}_s$ and satisfying

$$\begin{aligned}
(c.\mathbf{j}, \epsilon) \in \mathcal{T}_s(b.\mathbf{i}, \epsilon) \wedge (b.\mathbf{i}, \epsilon) \in UTC(a.\mathbf{h}, \epsilon) &\Rightarrow (c.\mathbf{j}, \epsilon) \in UTC(a.\mathbf{h}, \epsilon), \\
(c.\mathbf{j}, \epsilon) \in \mathcal{T}_s(b.\mathbf{i}, \epsilon) \wedge (b.\mathbf{i}, \epsilon) \in UTC(\mathbf{h}, a) &\Rightarrow (c.\mathbf{j}, \epsilon) \in UTC(\mathbf{h}, a), \\
(\mathbf{j}, c) \in \mathcal{T}_s(b.\mathbf{i}, \epsilon) \wedge (b.\mathbf{i}, \epsilon) \in UTC(a.\mathbf{h}, \epsilon) &\Rightarrow (\mathbf{j}, c) \in UTC(a.\mathbf{h}, \epsilon), \\
(\mathbf{j}, c) \in \mathcal{T}_s(b.\mathbf{i}, \epsilon) \wedge (b.\mathbf{i}, \epsilon) \in UTC(\mathbf{h}, a) &\Rightarrow (\mathbf{j}, c) \in UTC(\mathbf{h}, a).
\end{aligned}$$

UTC captures all transition sequences in M that do not pass through stable states. We use UTC to build $\mathcal{T}_{obs}$, by applying the *And* operator (Sect. 2) to select the elements with $w_b \in \mathcal{X}_{ext}$ and $w'_b \in \mathcal{Y}_{ext}$, corresponding to input and output symbols leading from stable to stable states. UTC is the most time and memory consuming step in our framework. First, we define a *ComRP* operator to calculate this composition effect of next-state functions, taking two $2(K+1)$-variable MDDs and returning the result composition $2(K+1)$-variable MDD. UTC can be obtained by repeatedly applying *ComRP* to $\mathcal{T}_s$: $UTC = \mathcal{T}_s \cup ComRP(\mathcal{T}_s) \cup ComRP^2(\mathcal{T}_s) \cup \cdots$. Thus, *UtcBfs* performs a global fixpoint in BFS style at Line 2-5 and uses *ComRP* with Line 26b in Fig. 2.

However, for asynchronous systems, saturation [4] is often orders of magnitude more efficient in memory and runtime than BFS algorithms, due to its effective utilization of *locality* (transitions in $\mathcal{T}_k$ only affecting i_k of M_k and β) through a series of light-weight recursions. Our saturation algorithm *UtcSat* chooses a different iteration strategy to approach the fixpoint with exhaustive utilization of locality. Thus, instead of taking $\mathcal{T}_s$ as one MDD, *UtcSat* uses its *disjunctive* form as K MDDs and divides the whole procedure into K phases. The k^{th} phase starts when the lower $(k-1)^{th}$ phases end at Line 4-6 and extends the fixpoint using $\mathcal{T}_k$ until node p is "saturated" (no more new states can be found) at Line 7-19. If it finds new states during this phase, only these need to be resaturated by all previous $k-1$ phases by using *ComRP* with Line 26s. Saturation works bottom-

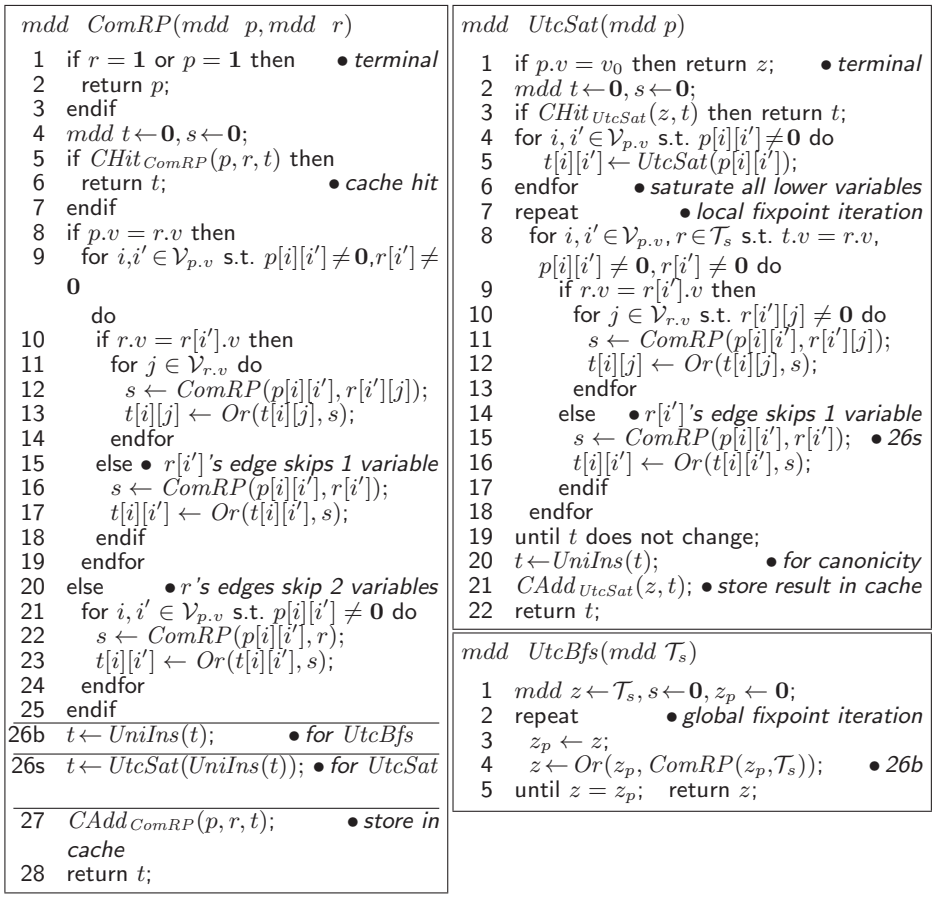

Fig. 2. The *ComRP* operator, the *UtcBfs*, and the *UtcSat* algorithms.

up and the result of local fixpoint for the K^{th} phase converges to the same global fixpoint as BFS. An *operation cache* avoids wasteful recomputations (Procedures *CAdd* and *CHit* are used to insert and retrieve computed results). Newly created nodes are inserted in a *unique table* (Procedure *UniIns*) to ensure MDD canonicity by avoiding duplicates. Our experience shows that the larger K is, the greater improvement saturation achieves. After building UTC, we obtain its restriction to stable transitions as $\mathcal{T}_{obs} = \{(a.\mathbf{i}, b.\mathbf{j}) \in UTC : \mathbf{i}, \mathbf{j} \in \mathcal{S}_{st}, a \in \mathcal{X}_{ext}, b \in \mathcal{Y}_{ext}\}$.

4 Symbolic NCFSM verification

Symbolic livelock check. An NCFSM does not terminate if it reaches a livelock (a cycle of unstable transitions): $a^{(1)}.\mathbf{i}^{(1)}{}_{[\underline{a^{(1)}}/\underline{a^{(2)}}\rangle} \cdots a^{(n)}.\mathbf{i}^{(n)}{}_{[\underline{a^{(n)}}/\underline{a^{(1)}}\rangle} a^{(1)}.\mathbf{i}^{(1)}$. Since livelock is a fatal design error, we need to guarantee livelock-freeness before test generation. If an NCFSM contains livelocks, the MDD encoding UTC has transitions where the "from" global state is the same as the "to" global state, and the input symbol equals the output symbol and belongs to $\mathcal{Z}_{int}$. Thanks

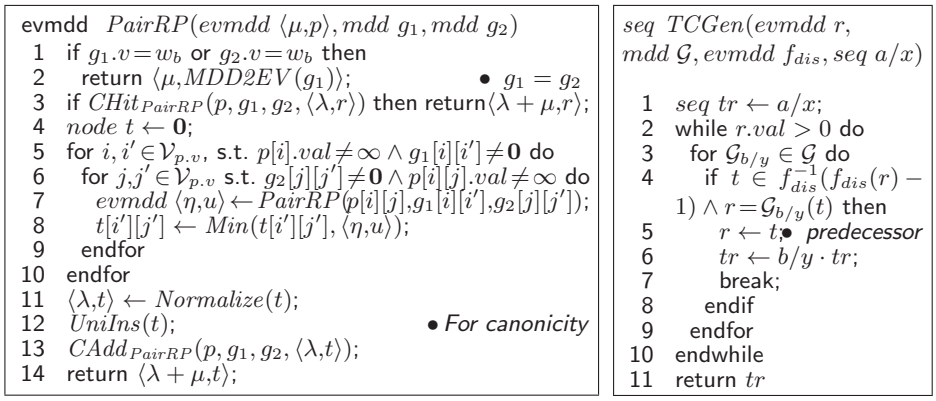

Fig. 3. Algorithms for the *PairRP* operator and test case generation.

to our MDD encoding, we can find all reachable states originating a livelock by *And*-ing $\mathcal{S}_{rch}$ and the "from" global states of the UTC. We not only verify livelock freeness, but also generate sequences from $\mathbf{s}_{init}$ to all livelocks, which is similar to distinguishing sequence generation, discussed in the next section.

Symbolic strong connectedness check. Many traditional FSM-based test derivation algorithms require the FSM to be strongly connected. While our approach does not require this property, strong connectedness can be checked in our framework. An NCFSM is *strongly connected* iff the initial state $\mathbf{s}_{init}$ is reachable from every reachable state $\mathbf{i} \in \mathcal{S}_{st}$. To check this property, we build the MDD for $\mathcal{T}^{-1}$, defined by $\mathbf{i}_{[a/x\rangle}\mathbf{j} \in \mathcal{T} \Leftrightarrow \mathbf{j}_{[x/a\rangle}\mathbf{i} \in \mathcal{T}^{-1}$, by switching the "from" and "to" variables. Then, we perform a backward state-space search from $\mathbf{s}_{init}$ along $\mathcal{T}^{-1}$ and build the set of reachable states $\mathcal{S}_{st}^{-1}$ using BFS or saturation. The global state space is strongly connected iff $\mathcal{S}_{st} = \mathcal{S}_{st}^{-1} \cap \mathcal{S}_{rch}$. Note that $\mathcal{T}^{-1}$ might be non-deterministic even if $\mathcal{T}$ is deterministic, but this does not hinder the applicability of symbolic state-space exploration.

Symbolic dead transition check. Transition $i_k\,[M_k, a/b\rangle\, j_k$ is *dead* if it does not contribute to building M_{obs}. Dead transitions reflect wasteful designs or useless functions, which should be reported to the designer. As $\mathcal{S}_{rch} = \mathcal{S}_{st} \cup \mathcal{S}_{unst}$ is available, dead transitions can be detected symbolically. For each $i_k\,[M_k, a/b\rangle\, j_k$, we can first check if $\mathcal{S}_{unst}$ contains an unstable state with $w_k = i_k$ and $w_b = a$; if it does, $i_k\,[M_k, a/b\rangle\, j_k$ is not dead. Otherwise, if $a \in \mathcal{X}_{ext}$, we check if $\mathcal{S}_{rch}$ contains a stable state with $w_k = i_k$; if it does, $i_k\,[M_k, a/b\rangle\, j_k$ is not dead, since a can be received from the environment in that state. Otherwise, $i_k\,[M_k, a/b\rangle\, j_k$ is dead.

5 Symbolic NCFSM test derivation

Given specification NCFSM M, we apply the following mutant operators, corresponding to possible error classes, to generate a set of first-order mutants $\mathcal{U}$.

– **Alter the initial state**: create a mutant by changing one of the local states in the initial state $\mathbf{s}_{init}$. This generates $\sum_{1 \le k \le K}(|\mathcal{S}_k| - 1)$ mutants.

– **Alter the output of a local transition**: create a mutant by changing local transition $i\,[M_k,\,a/b\rangle\,j$ to $i\,[M_k,\,a/b'\rangle\,j$, for $b' \in \mathcal{Y}_k \setminus \{b\}$. This generates $\sum_{1\leq k\leq K} |\delta_k|(|\mathcal{Y}_k|-1)$ mutants, where $|\delta_k|$ is the number of local transitions in M_k, thus $|\delta_k| = |\mathcal{X}_k|\cdot|\mathcal{S}_k|$ if the model is completely specified.
– **Alter the destination state of a local transition**: create a mutant by changing local transition $i\,[M_k,\,a/b\rangle\,j$ to $i\,[M_k,\,a/b\rangle\,j'$, where $j' \in \mathcal{S}_k \setminus \{j\}$. This generates $\sum_{1\leq k\leq K} |\delta_k|(|\mathcal{S}_k|-1)$ mutants.

Given a mutant $\overline{M}$ of specification M ("$\bar{}$" indicates quantities related to the mutant), we seek a sequence $a_1/b_1,\ldots,a_n/b_n$ that kills this mutant, if not equivalent to M, where each a_i/b_i pair corresponds to an input symbol and the corresponding expected output in M. Let $\alpha = a_1a_2\cdots a_{n-1}$ and $\beta = b_1b_2\cdots b_{n-1}$, then $\lambda_{obs}(s,\alpha)=\beta=\overline{\lambda_{obs}}(s,\alpha)$ and $\lambda_{obs}(\delta_{obs}(s,\alpha),a_n) = b_n \neq \overline{\lambda_{obs}}(\overline{\delta_{obs}}(s,\alpha),a_n)$.

If the *state pair* set is $\mathcal{P} = \mathcal{S}_{st} \times \overline{\mathcal{S}}_{st}$, define the *next-state-pair function* $\mathcal{G} = \{\mathcal{G}_{a/b} : a\in\mathcal{X}, b\in\mathcal{Y}\}$ and the *distinguishable-state-pairs* $\mathcal{D}=\{\mathcal{D}_{a/b} : a\in\mathcal{X}, b\in\mathcal{Y}\}$:

$$\mathcal{G}_{a/b} = \{\big((a.\mathbf{i}, b.\mathbf{j}), (a.\bar{\mathbf{i}}, b.\bar{\mathbf{j}})\big) : (a.\mathbf{i}, b.\mathbf{j}) \in \mathcal{T}_{obs} \wedge (a.\bar{\mathbf{i}}, b.\bar{\mathbf{j}}) \in \overline{\mathcal{T}_{obs}}\},$$
$$\mathcal{D}_{a/b} = \{(a.\mathbf{i}, b.\bar{\mathbf{i}}) : \exists(a.\mathbf{i}, b.\mathbf{j}) \in \mathcal{T}_{obs} \wedge \exists(a.\bar{\mathbf{i}}, \bar{b}.\bar{\mathbf{j}}) \in \overline{\mathcal{T}_{obs}} \wedge b \neq \bar{b}\},$$

which can be built through symbolic operations on $\mathcal{S}_{st}$, $\mathcal{T}_{obs}$, and $\overline{\mathcal{T}_{obs}}$.

Our test derivation algorithm takes in input the set $\mathcal{U}$ of mutants, the stable next-state function $\mathcal{T}_{obs}$, and $\mathbf{p}_{init} = (\mathbf{i}_{init}, \bar{\mathbf{i}}_{init})$. For each mutant $\overline{M}$, we first run the current test suite to check whether an existing test kills $\overline{M}$. If not, we build $\overline{\mathcal{T}_{obs}}$ and encode the next-state-pair function $\mathcal{G}$ and the distinguishable-state-pairs $\mathcal{D}$. Fig. 3 shows the *PairRP* operator that is analogous to state-space exploration except that we explore pairs of states (one from M, one from $\overline{M}$), and keep track of the distance of each such pair from $\mathbf{p}_{init}$ by using a $2(K{+}1)$-variable EV$^+$MDD to encode the *distance function* $f_{dis} : \mathcal{P} \to \mathbb{N}\cup\{\infty\}$ s.t. $f_{dis}(\mathbf{p}) = \min\{d : \mathbf{p}\in\mathcal{G}^d(\mathbf{p}_{init})\}$. Thus, $f_{dis}(\mathbf{p})=\infty$ iff $\mathbf{p}$ has not yet been reached in the exploration, initialized with $f_{dis}(\mathbf{p}_{init}) = 0$ and $f_{dis}(\mathbf{p}) = \infty$ for $\mathbf{p} \neq \mathbf{p}_{init}$. We also define the reverse function $f_{dis}^{-1}(d) = \{\mathbf{p} : f_{dis}(\mathbf{p}) = d\}$, where $d \in \mathbb{N}$.

The algorithm uses a BFS algorithm to generate the distance function for reachable state pairs until the search reaches a distinguishing state pair $\mathbf{p}_{err}$ in $\mathcal{D}$. $\mathbf{p}_{err}$ is used to generate a sequence as a new test case which is added to the test suite $\mathcal{C}$. Then, the algorithm *TCGen* in Fig. 3 use $f_{dis}^{-1}(d)$ to generate a sequence leading M and $\overline{M}$ from $\mathbf{p}_{init}$ to $\mathbf{p}_{err}$. This is the same approach proposed in [5] to generate the shortest path to a target state, except that now we target a *pair* of states $\mathbf{p}_{err}$. Starting from $\mathbf{p}$ at distance n, there must exist a predecessor $\mathbf{q}$, i.e., satisfying $\mathbf{p} = \mathcal{G}(\mathbf{q})$, at distance $n-1$, as Line 4. Thus, we keep reducing the distance value until reaching $\mathbf{p}_{init}$, at distance 0. If no such pair $\mathbf{p}_{err}$ is instead reachable, $\overline{M}$ is equivalent to M, and the algorithm builds a fixpoint $\mathcal{P}_{rch}$ containing all the pairs of states that can be reached from $\mathbf{p}_{init}$ by providing the same input sequence to both M and $\overline{M}$. Finally, the algorithm eliminates test cases subsumed by other test cases, to form a minimal test suite.

6 Experimental results

We implemented the proposed framework using our MDD library [17], and report experimental results on an Intel Xeon 2.53GHz workstation with 36GB

Model		Mutants		Test Suite		UTC_{bfs}	UTC_{sat}	Tot_{bfs}		Tot_{sat}	
M	K	Tot	NE	Num	Avg	time	time	mem	time	mem	time
Ideal models											
M_{se}	3	96	96	27	3.19	0.053 s	0.021 s	3.63 M	0.427 s	3.42 M	0.235 s
M_{hs}	3	81	81	16	2.81	0.044 s	0.018 s	3.60 M	0.301 s	3.30 M	0.158 s
Not ideal and real models											
M'_{se}	3	90	90	23	3.30	0.039 s	0.014 s	3.28 M	0.245 s	3.10 M	0.192 s
M'_{hs}	3	77	77	13	3.08	0.041 s	0.012 s	3.24 M	0.252 s	3.00 M	0.126 s
M_{hcs}	4	179	157	12	11.17	0.17 s	0.03 s	7.01 M	1.00 s	6.54 M	0.48 s
M_{tr}	4	177	150	12	2.5	0.17 s	0.09 s	6.72 M	1.00 s	5.33 M	0.20 s
M_{tr3}	5	1024	849	43	3.14	4.69 s	3.35 s	69.75 M	9.25 s	50.72 M	7.74 s
ABP	2	96	81	12	3.83	0.005 s	0.004 s	2.90 M	0.49 s	2.90 M	0.32 s
BGP	4	4898	1613	79	5.16	344.26 s	24.1 s	96.48 M	1230.0 s	82.41 M	438.6 s
EGP	3	69066	27883	3501	9.24	11.14 h	5.28 h	6.78 G	21.58 h	4.51 G	14.03 h

Table 1. Test derivation results (time in seconds or hours, memory in MB or GB).

RAM running Linux. The main metrics of our comparison are runtime and peak memory. For BFS and saturation, we compare the cumulative time to compute the UTC on all mutants (UTC_{bfs} and UTC_{sat}), and the total runtime and peak memory (Tot_{bfs} and Tot_{sat}) using BFS and saturation respectively. For each model, we list the number of components (K), of mutants (Tot), of non-equivalent mutants (NE), of test cases (Num), and the average length of the tests in the suite (Avg). The total time includes preprocessing, livelock checking, and test suite generation.

Table 1 presents results for two sets of models. The first set, shown under "Ideal models", consists of M_{se} [8] and M_{hs} [9]. All components are completely specified, minimized, strongly connected, and deterministic. The second set contains control systems, communication protocols, and two corresponding incompletely specified models M'_{se} and M'_{hs} by eliminating some self-loops. Three control systems include a heating controller system [2] and a train gate controller [1] with two trains, M_{tr}, or three trains, M_{tr3}. Three communication protocols contain the alternating bit protocol (ABP) [16], the border gateway protocol (BGP) [15], and the exterior gateway protocol (EGP) [14].

Saturation works better in both time and memory, although only minor improvements are observable for some models. For communication protocols, BGP and EGP are two important TCP/IP exterior routing protocols. BGP is currently used on the Internet and other larger autonomous systems. For these two models, we only consider mandatory events. Thus, including the component for the environment, we encode the model with 5 variables. Saturation is clearly superior, 14 times faster than BFS when computing the UTC for all mutants. Similar trends can be observed for EGP with three peers: saturation saves almost 8 hours and over 2GB over BFS (there are about 1.2×10^5 global states, 6.7×10^5 local transitions, and 1.5×10^6 global transitions).

The benefit of symbolic encodings can be clearly seen in our results, as the memory consumption remains stable even if the number of generated mutants

increases by an order of magnitude when growing the number of components. Also, we observe that the number of generated test cases and the average length of the test suite are stable even if the number of mutants increases dramatically. This is important for complex models in practice, as it reduces testing efforts.

7 Conclusion

We presented a new symbolic framework for NCFSM verification and test generation. We encode an NCFSM with MDDs and use the BFS and saturation algorithms to generate the unstable transitive closure of transitions. We symbolically check for livelocks, dead transitions, and strong connectedness. Then, we propose a symbolic mutation-based test generation algorithm. The experimental results demonstrate the effectiveness of this framework. A further advantage of our symbolic framework is that no constraints are required of IUTs. Some of those requirements by other test generation methods might not be met by many real models. Moreover, our framework guarantees a test suite with minimal-length tests to kill all non-equivalent mutants and it could be extended to non-deterministic NCFSMs by returning, instead of distinguishing sequences, pairs consisting of an input string and a set of all correct output strings.

References

1. BEEM models. `http://anna.fi.muni.cz/models/cgi/models.cgi`.
2. Event-B examples. `http://wiki.event-b.org/index.php/Event-B_Examples`.
3. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, Aug. 1986.
4. G. Ciardo, R. Marmorstein, and R. Siminiceanu. The saturation algorithm for symbolic state space exploration. *STTT*, 8(1):4–25, 2006.
5. G. Ciardo and R. Siminiceanu. Using edge-valued decision diagrams for symbolic generation of shortest paths. *Proc. FMCAD*, LNCS 2517:256–273. 2002.
6. Q. Guo et al. Computing unique input/output sequences using genetic algorithms. *Proc. FATES*, pp. 169–184, 2004.
7. O. Henniger. On test case generation from asynchronously communicating state machines. *Proc. TCS*, pp. 255–271, 1997.
8. R. Hierons. Testing from semi-independent communicating finite state machines with a slow environment. *IEE Proc., Software Engineering*, 144(5):291–295, 1997.
9. R. Hierons. Checking states and transitions of a set of communicating finite state machines. *Microprocessors and Microsystems*, 24:443–452, 2000.
10. T. Kam et al. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
11. J. Li and W. Wong. Automatic test generation from communicating extended finite state machine (CEFSM)-based models. *Proc. ISORC*, pp. 181–185, 2002.
12. G. Luo, G. von Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized WP-method. *IEEE Trans. Softw. Eng.*, 20:149–162, Feb. 1994.
13. A. P. Mathur. *Foundations of Software Testing*. Pearson Education, 2008.
14. D. L. Mills. Exterior gateway protocol formal specification, 1984.
15. Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4), 2006. RFC 4271.

16. A. Tanenbaum. *Computer Networks*. Prentice Hall, 2003.
17. M. Wan and G. Ciardo. Symbolic state-space generation of asynchronous systems using extensible decision diagrams. *Proc. SOFSEM*, LNCS 5404:582–594. 2009.