MUPPAAL: Reducing and Removing Equivalent and Duplicate Mutants in UPPAAL

Jaime Cuartas
Universidad del Valle
Cali, Colombia
jaime.cuartas@correounivalle.edu.co

Jesús Aranda
Universidad del Valle
Cali, Colombia
jesus.aranda@correounivalle.edu.co

Maxime Cordy
SnT, University of Luxembourg
Luxembourg, Luxembourg
maxime.cordy@uni.lu

James Ortiz
FOCUS/NaDI, University of Namur
Namur, Belgium
james.ortizvega@unamur.be

Gilles Perrouin
PReCISE/NaDI, University of Namur
Namur, Belgium
gilles.perrouin@unamur.be

Pierre-Yves Schobbens
PReCISE/NaDI, University of Namur
Namur, Belgium
pierre-yves.schobbens@unamur.be

***Abstract*—Mutation Testing (MT) is a test quality assessment technique that creates mutants by injecting artificial faults into the system and evaluating the ability of tests to distinguish these mutants. We focus on MT for safety-critical Timed Automata (TA). MT is prone to equivalent and duplicate mutants, the former having the same behaviour as the original system and the latter other mutants. Such mutants bring no value and induce useless test case executions. We propose MUPPAAL, a tool that: (1) offers a new operator reducing the occurrence of mutant duplicates; (2) an efficient bisimulation algorithm removing remaining duplicates; (3) leverages existing equivalence-avoiding mutation operators. Our experiments on four UPPAAL case studies indicate that duplicates represent up to 32% of all mutants and that the MUPPAAL bisimulation algorithm can identify them more than 99% of the time.**

***Index Terms*—Model-Based Testing, Timed Automata, Mutation Testing, UPPAAL**

I. Introduction

Timed systems (TS) are systems in which strict time constraints are essential for reliability and correctness. They appear in planes, trains, and a variety of safety-critical systems. Ensuring Quality Assurance (QA) of TS is essential. Model-Based Testing (MBT) [46], [49] exploits (timed) specifications to generate test cases assessing the system's behaviour and avoids scalability issues induced by exhaustive verification. Yet, one must ensure its ability to find bugs. Mutation Testing (MT) [28] creates *mutants* of the system by injecting artificial defects via predefined *mutation operators*. Tests can then *distinguish* (or *kill*) mutants if they behave differently on the mutant than on the original system. The *mutation score* is the ratio of killed mutants to the total number of mutants. Though MT has long focused on code [43], [20], [39], Model-Based Mutation Testing (MBMT) helps in the automatic identification of defects related to missing functionality and misinterpreted specifications [11] that are difficult to identify via code-based testing [26], [48]. Yet, not all mutants are relevant. Some may be *equivalent*, i.e., they exhibit the same behaviour as the original system despite their syntactic difference [42]. Therefore, no test case can distinguish such mutants. Similarly, *duplicate* mutants exhibit the same behaviour as other mutants [41], [42]. Preventing and removing such *useless* mutants reduces the computation costs of (MB)MT and builds more trust in mutation scores. Recently, Basile *et al.* tackled the equivalent mutant problem for Timed Automata with Input and Output (TAIO): they defined mutation operators preventing mutants from refining the original system [8], [7]. However, this technique does not address duplicate mutants: in our experiments, up to 32% of all generated mutants were duplicates. We propose MUPPAAL, a mutation approach that addresses this challenge for Timed Automata (TA) specified in UPPAAL [9]. The contributions of this paper are the following:

1) We introduce a *novel timed mutation operator*, SMI-NR, that we *proved to prevent duplicate mutants* by design;
2) We *detect duplicate mutants* using a timed bisimulation algorithm [40] to assess behavioural equivalence between two mutants. When duplicate mutants are detected, we keep only one of them;
3) We provide a random simulation baseline to compare to timed bisimulation: If, for a mutant, we can find a trace that the other mutant cannot execute, then we can conclude that the two mutants are not duplicates. The heuristic suggests a pair of mutants as duplicates otherwise;
4) We implemented MUPPAAL using the UPPAAL execution engine for TAs and UPPAAL-TRON [35] for timed trace generation and checking. In addition to the above contributions, MUPPAAL supports mutation operators for TA from [1], [8], [40] avoiding equivalent mutants using refinement prevention [7], [8];
5) We assessed MUPPAAL[1]. Our results on four cases indicate that timed bisimulation offers the *best trade-off* between performance and accuracy of detection. In contrast, the random baseline suggests many false duplicates (up to 10 times compared to bisimulation). Our

[1]MUPPAAL implementation and full results of its evaluation are available: https://anonymous.4open.science/r/Muppaal-91A7

novel mutation operator effectively reduces the number of mutants while perfectly capturing the initial mutation operator behaviour.

The remainder of this paper is as follows. Section II introduces the formalisms we use and the equivalent and duplicate mutant problem. Section III presents our new mutation operator and duplicate removal algorithms. Section IV describes MUPPAAL and reports on our experiments. Section V presents related work, and Section VI wraps up with concluding remarks and future work.

II. Background

A. Clocks and Timed Automata

To model the continuous time domain, we use non-negative real-valued variables: *clocks*. Clocks are variables that increase at the same rate (i.e., synchronously). TA are one of the most studied formalisms for modelling TS [4]. Several model checkers such as UPPAAL [9], KRONOS [10], and HYTECH [23] rely on TA. TA are an extension of Finite State Automata (FSA) with a set of clocks increasing at the same rate. Resetting TA clock means updating the clock value to zero. TA allows enabling or disabling transitions using *clock constraints*, and we take transition actions if all other conditions are satisfied. We use an extension of TA called Timed Automata with Inputs and Outputs (TAIO) [2]. TAIO partition the actions into two disjoint sets for inputs and outputs [15][2]. Here, we use the extension of TAIO proposed by Aichernig *et al.* [2]

B. Timed Automata with Inputs and Outputs

A TAIO is a refined TA where we model the interaction between a system and its environment by using output and input actions [2]. The *clock constraints* are defined below.

Definition 1 (Clock constraints)**.** *Let X be a finite set of clock variables ranging over $\mathbb{R}_{\geq 0}$ (non-negative real numbers). Let $\Phi(X)$ be a set of clock constraints over X. A clock constraint $\phi \in \Phi(X)$ can be defined by the following grammar:*

$$\phi ::= true \mid x \sim c \mid \phi_1 \ \wedge \ \phi_2$$

where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, >, \leq, \geq, =\}$.

Definition 2 (Clock Invariants)**.** *Let X be a finite set of clock variables ranging over $\mathbb{R}_{\geq 0}$. Let $\Delta(X)$ be a set of clock invariants over X. Clocks invariants are clock constraints of the following form:*

$$\delta ::= true \mid x < c \mid x \leq c \mid \phi_1 \ \wedge \ \phi_2$$

where $x \in X$, $c \in \mathbb{N}$.

Definition 3 (Clock valuations)**.** *Given a finite set of clocks X, a clock valuation function, $\nu : X \to \mathbb{R}_{\geq 0}$ assigning to each clock $x \in X$ a non-negative value $\nu(x)$. We denote $\mathbb{R}^X_{\geq 0}$ the set of all valuations. For a clock valuation $\nu \in \mathbb{R}^X_{\geq 0}$ and a time value $d \in \mathbb{R}_{\geq 0}$, $\nu + d$ is the valuation satisfied by $(\nu + d)(x) = \nu(x) + d$ for each $x \in X$. Given a clock subset $Y \subseteq X$, we denote $\nu[Y \leftarrow 0]$ the valuation defined as follows: $\nu[Y \leftarrow 0](x) = 0$ if $x \in Y$ and $\nu[Y \leftarrow 0](x) = \nu(x)$ otherwise.*

In TAIO, the transitions can have a *guard* that will allow the transitions to be taken or not, performing actions and resetting clocks. In TAIO, one classifies actions (or alphabet) into two disjoint subsets: input actions (suffixed with ?) and output actions (suffixed with !) [2]. The output actions of a TAIO $\mathcal{A}$ can be input actions of a TAIO $\mathcal{B}$. We adapt the definition of [30], where the initial location is unique, and discrete variables and internal actions are not allowed. We formally define TAIO as:

Definition 4 (TAIO)**.** *A TAIO is a tuple $(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$, where:*

- *L is a finite set of locations,*
- *$l_0 \in L$ is an initial location,*
- *X is a finite set of clocks,*
- *Σ_I is a finite set of input actions (?),*
- *Σ_O is a finite set of output actions (!),*
- *$\Sigma = \Sigma_I \cup \Sigma_O$, is a finite set of input and output actions, such that $\Sigma_I \cap \Sigma_O = \emptyset$,*
- *$T \subseteq L \times \Sigma \times \Phi(X) \times 2^X \times L$ is a finite set of transitions,*
- *$I : L \to \Delta(X)$ is a function that associates to each location a clock invariant.*

For a transition $(l, a, \phi, Y, l') \in T$, we classically write $l \xrightarrow{a,\phi,Y} l'$ and call l and l' the source and target location, ϕ is the guard, a the action (or alphabet), Y the set of clocks to reset. The semantics of a TAIO is a Timed Input/Output Transition System (TIOTS) where a *state* is a pair $(l, \nu) \in L \times \mathbb{R}^X_{\geq 0}$, where l denotes the current location with its accompanying clock valuation ν, starting at (l_0, ν_0) where ν_0 maps each clock to 0. The transitions can be of types: Delay transitions only let time pass without changing location. We only consider *legal* states, i.e. states satisfying the current state invariant $\nu \models$ Discrete transitions occur instead between a source and a target location. The transition can only occur if the current clock values satisfy both the guard of the transition and the invariant of the target location.

Definition 5 (Semantics of TAIO)**.** *Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ be a TAIO. The semantics of TAIO $\mathcal{A}$ is given by a TIOTS$(\mathcal{A}) = (S, s_0, \Sigma_I, \Sigma_O, \rightarrow)$ where:*

- *$S \subseteq L \times \mathbb{R}^X_{\geq 0}$ is a set of states,*
- *$s_0 = (l_0, \nu_0)$ with $\nu_0(x) = 0$ for all $x \in X$ and $\nu_0 \models I(l_0)$,*
- *$\Sigma_\Delta = \Sigma \uplus \mathbb{R}_{\geq 0}$,*
- *$\rightarrow \subseteq S \times \Sigma_\Delta \times S$ is a transition relation defined by the following two rules:*
  - ***Discrete transition:*** *$(l, \nu) \xrightarrow{a} (l', \nu')$, for $a \in \Sigma$ iff $l \xrightarrow{a,\phi,Y} l'$, $\nu \models \phi$, $\nu' = \nu[Y \leftarrow 0]$ and $\nu' \models I(l')$ and,*
  - ***Delay transition:*** *$(l, \nu) \xrightarrow{d} (l, \nu + d)$, for some $d \in \mathbb{R}_{\geq 0}$ iff $\nu + d \models I(l)$.*

A *path* in TIOTS$(\mathcal{A})$ is a finite sequence of consecutive delays and discrete transitions. A finite execution frag-

ment of $\mathcal{A}$ is a path in TIOTS($\mathcal{A}$) starting from the initial state $s_0 = (l_0, \nu_0)$, with delay and discrete transitions alternating along the path: $\rho = (l_0, \nu_0) \xrightarrow{d_0} (l_0, \nu'_0) \xrightarrow{a_0} (l_1, \nu_1) \dots (l_{n-2}, \nu'_{n-2}) \xrightarrow{a_{n-1}} (l_{n-1}, \nu_{n-1} \xrightarrow{d_n} (l_n, \nu_n)$ where $\nu_0(x) = 0$ for every $x \in X$. A *path* of TIOTS($\mathcal{A}$) is *initial* if $s_0 = (l_0, \nu_0) \in S$, where $l_0 \in L$, ν_0 assign 0 to each clock, and *maximal* if it ends in a location without outgoing edges.

A *timed trace* [4] over Σ is a finite sequence $\theta = ((\sigma_1, t_1), (\sigma_2, t_2) \dots (\sigma_n, t_n))$ of actions paired with non-negative real numbers (i.e., $(\sigma_i, t_i) \in \Sigma \times \mathbb{R}_{\geq 0}$) such that the timestamped sequence $t = t_1 \cdot t_2 \cdots t_n$ is non-decreasing (i.e., $t_i \leq t_{i+1}$).

Example 1. *Let $\mathcal{A}$ be the TAIO depicted in Fig 1. $\mathcal{A}$ contains two locations: l_0 (initial) and l_1. We denote input actions ($?$) and output actions ($!$). In particular, l_0 is the only location to define an invariant not trivially true: $I(l_0) = (x < 7)$, forcing the TAIO to exit l_0 before x becomes 7. Location l_1 has a true invariant (thus not drawn), allowing it to stay in l_1 forever. Suppose the current location is l_1. The transition $l_1 \xrightarrow{b?,(y=9),\{x:=0;y:=0\}} l_0$ specifies that when the input action $b?$ occurs and the guard $y = 9$ holds, this enables the transition, leading to a new current location l_0, while resetting clock variables x and y. Note that using a location invariant (which specifies the time limit to stay in a given location) differs from using a guard (specifying when the transition is enabled). The automaton in Fig 1 is nondeterministic because location l_1 has two outgoing transitions on the same input action ($b?$).*

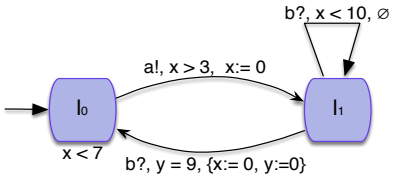

Fig. 1: A nondeterministic TAIO with two clocks x and y.

Definition 6 (Deterministic TAIO)**.** *A deterministic TAIO is a tuple $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ such that: for every $l \in L$, for all actions $a \in \Sigma$, for every pair of different edges of the form $(l, a, \phi_1, Y_1, l'_1) \in T$ and $(l, a, \phi_2, Y_2, l'_2) \in T$, imply $\phi_1 \cap \phi_2 = \emptyset$ and $l'_1 = l'_2$. (2) For every $l \in L$, for all actions $a \in \Sigma$, and every valuation ν there is an edge (l, a, ϕ, Y_1, l') such that $\nu \models \phi$.*

C. Timed Bisimulation and Trace Simulation

1) Traces: Simulation is a widely used technique to test software systems. However, *simulation* is insufficient to prove the absence of errors in safety-critical systems because of its non-exhaustiveness.

A *simulation trace* is a *timed trace* collected during the simulation execution.

2) Timed Bisimulation: To reason about the behavioural equivalence of mutants, we use the classical notion of timed bisimulation [12].

Definition 7 (**Timed Bisimulation** [12])**.** *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two TIOTS over the set of actions $\Sigma = (\Sigma_I \cup \Sigma_O)$. Let $S_{\mathcal{D}_1}$ (resp., $S_{\mathcal{D}_2}$) be the set of states of $\mathcal{D}_1$ (resp., $\mathcal{D}_2$). A timed bisimulation over TIOTS $\mathcal{D}_1$, $\mathcal{D}_2$ is a binary relation $\mathcal{R} \subseteq S_{\mathcal{D}_1} \times S_{\mathcal{D}_2}$ such that, for all $s_{\mathcal{D}_1} \mathcal{R}\ s_{\mathcal{D}_2}$, the following holds:*

1) *For every discrete transition $s_{\mathcal{D}_1} \xrightarrow{a}_{\mathcal{D}_1} s'_{\mathcal{D}_1}$ with $a \in \Sigma$, there exists a matching transition $s_{\mathcal{D}_2} \xrightarrow{a}_{\mathcal{D}_2} s'_{\mathcal{D}_2}$ such that $s'_{\mathcal{D}_1} \mathcal{R} s'_{\mathcal{D}_2}$ and symmetrically.*
2) *For every delay transition $s_{\mathcal{D}_1} \xrightarrow{d}_{\mathcal{D}_1} s'_{\mathcal{D}_1}$ with $d \in \mathbb{R}_{\geq 0}$, there exists a matching transition $s_{\mathcal{D}_2} \xrightarrow{d}_{\mathcal{D}_2} s'_{\mathcal{D}_2}$ such that $s'_{\mathcal{D}_1} \mathcal{R} s'_{\mathcal{D}_2}$ and symmetrically.*

$\mathcal{D}_1$ and $\mathcal{D}_2$ are timed bisimilar, written $\mathcal{D}_1 \sim \mathcal{D}_2$, if there exists a timed bisimulation relation $\mathcal{R}$ over $\mathcal{D}_1$ and $\mathcal{D}_2$ containing the pair of initial states.

D. Mutation Operators and Equivalence Problem

1) TA and Mutation Operators.: Nilsson *et al.* [38] were among the first to extend TA with a task model. A task model consists of a set of n (real-time) tasks, and they give each task a period T_i, a worst-case execution time C_i, and a relative deadline D_i and mutation operators. Nilsson *et al.* proposed six mutation operators: *execution time* (ET) affects the execution time of a task; *hold time shift* (HTS) and *lock/unlock time* (LUT) operators either shift the whole lock/unlock time interval for a resource or only one of its bounds; *precedence constraints* (PC) operators change precedence relations between pairs of tasks. The authors also define automata operators that affect both invariant and guard constraints either for a given location (*inter-arrival time* (IAT)) or for the initial location (*pattern offset* (PO)). Aboutrab *et al.* [25] and Aichernig *et al.* [2] also proposed some mutation operators for UPPAAL to test the behaviour of TS. Three of them are not time-related: *change action* (CA), *change source* (CS)/*target* (CT), and *sink location* (SL). The time-related operators are: *change guard* (CG) alters the inequality within the guard constraint, *negate guard* (NG) operator replaces a transition's Boolean guard by its logical negation, *invert reset* (IR) selects one clock variable and either adds it to the list of clocks to be reset during the transition if it is absent or removes it from the list if it is present, *change invariant* (CI) adds one time-unit to the invariant constraint in an automaton location. Basile *et al.* [8] proposed six mutation operators on TAIO, designed to avoid the generation of mutants subsumed by construction. Three of six mutation operators in [8] are time-independent: *Transition MIssing* (TMI) removes a transition; *Transition Add* (TAD) adds a transition between two locations; *State MIssing* (SMI) removes an arbitrary location (also called *state*) other than the initial location and all its incoming/outgoing transitions. The other three time-related operators are: *Constant eXchange Larger* (CXL) increases the constant of a clock constraint, *Constant eXchange Smaller* (CXS) decreases the constant of a clock constraint and *Clock Constraint Negation* (CCN) negates a clock constraint. The main idea in [8] is to perform a refinement check between

Nilsson *et al.* [38]		Aichernig *et al.* [2]		Basile *et al.* [8]	
Op	**Description**	**Op**	**Description**	**Op**	**Description**
ET	Execution time	**CA**	Change action	**TMI**	Transition missing
IAT	Inter-arrival time	**CT**	Change target	**TAD**	Transition ADd
PO	Pattern offset	**CS**	Change source	**SMI**	State missing
LT	Lock time	**CG**	Change guard	**CXL**	Constant exchange L
UT	Unlock time	**NG**	Negate guard	**CXS**	Constant exchange S
HTS	Hold time shift	**CI**	Change invariant	**CCN**	Constraint negation
PC	Precedence constraints	**SL**	Sink location	-	-
-	-	**IR**	Invert reset	-	-

TABLE I: Mutation operators for TA.

the mutant and the system model, using ECDAR [33]. Table I shows the mutation operators retrieved from the considered contributions.

2) Equivalent/Duplicate mutation problem.: MT is one of the most effective coverage criteria to evaluate test suite quality [28], [42]. In addition, several recent empirical studies have evaluated the effectiveness and efficiency of MT [41], [29], [42]. However, MT has a high cost. Equivalent and duplicate mutants (i.e., useless mutants [41]) contribute to increasing costs [42]: between 30% - 40% of equivalent mutants [37] and between 20% - 30% of duplicate mutants [42]. MUPPAAL decrease MBMT costs by eliminating useless mutants from the analysis.

E. UPPAAL and UPPAAL-TRON

UPPAAL is a tool for the modelling, simulation, and verification of networks of TA extended with data types, user functions, clocks, and synchronous communication channels [9]. UPPAAL-TRON [21] is a testing tool, based on UPPAAL, suited for online black-box conformance testing of TS. UPPAAL-TRON is used for testing the Implementation Under Test (IUT). UPPAAL-TRON can use a randomized online testing algorithm, an extension of the UPPAAL model checker [9]. UPPAAL-TRON can generate and execute tests event by event in real-time by stimulating and monitoring the IUT. UPPAAL-TRON performs these two operations, computing the possible set of symbolic states based on the *timed trace* observed so far. A *timed trace* in UPPAAL-TRON consists of a sequence of input or output actions and time delays [35].

III. Overcoming Equivalent and Duplicate Mutants Problem

Three strategies target the equivalent (and duplicate) mutant problem [37] : (1) avoid (2) detect, and (3) suggest equivalent (and duplicate) mutants. We describe in this section how MUPPAAL implements them.

A. Avoiding Equivalent and Duplicate Mutants

Mutation operator design is essential for an effective MBMT tool. It must generate as few mutants as possible without losing efficiency, *i.e.*, avoiding useless mutants. However, most MBMT tools [38], [25], [1] do not avoid the generation of useless mutants. Basile *et al.* [8] avoid equivalent mutants by proposing operators guaranteeing mutants do not refine the original system's behaviour. We have implemented them for UPPAAL. In addition, we offer a new duplicate-avoiding mutation operator.

1) Mutation Operators and Non-Equivalent Mutants.: Here, we use the guidelines and the six mutation operators of [8] (see Table I). We rely on them to avoid equivalent mutants.

2) A new duplicate-avoiding Mutation Operator.: A duplicate mutant has the same behaviour as another mutant and is thus useless. Basile *et al.* refinement technique ensures non-equivalence [8], [7] but does not avoid mutant duplicates. Hence, we introduce a new mutation operator (SMI-NR), avoiding first-order duplicate mutants between SMI and TMI.

Example 2. *Fig. 2 illustrates a base system modelled as a non-duplicate TAIO. Applying TMI, i.e. removing the second transition* $(l_1, a?, true, \emptyset, l_2)$ *gives Fig. 3, while applying SMI, i.e. removing location* l_2*, gives Fig. 4. Both have the same behaviour: they are thus mutant duplicates.*

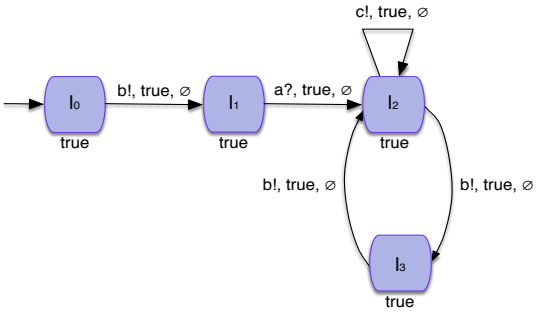

Fig. 2: An original model (TAIO).

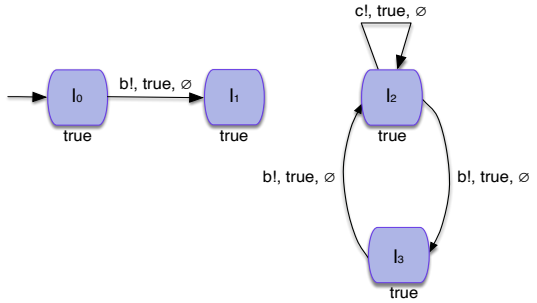

Fig. 3: A mutant generated by TMI operator from Figure 2.

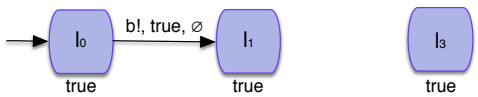

Fig. 4: A mutant generated by SMI operator from Figure 2.

To formally specify SMI-NR, we first note that a mutation operator is a function $\mathcal{M}_\mu$ that generates a set of mutants from a TAIO. We use μ to refer to each specific operator presented in [8]. The following theorem and proposition consider the case of a TMI mutant and a SMI mutant being timed bisimilar.

Theorem 1 (TMI and SMI duplicate mutants)**.** *Let $\mathcal{A}$ be a TAIO and the mutants $\mathcal{A}_{tmi}$ and $\mathcal{A}_{smi}$ where:*

$$\mathcal{A}_{tmi} \in \mathcal{M}_{tmi}(\mathcal{A}) \text{ such that } \mathcal{A}_{tmi} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{t_{tmi}\}, I),\ t_{tmi} = (l_1, a_{tmi}, \phi, Y, l_2) \in T \text{ and } a_{tmi} \in \Sigma_I, \text{ and } \mathcal{A}_{smi} \in \mathcal{M}_{smi}(\mathcal{A}) \text{ such that } \mathcal{A}_{smi} = (L \setminus \{l_{smi}\}, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T_{smi}, I),\ l_{smi} \in L, l_{smi} \neq l_0, T_{smi} = \{(l_1, a, \phi, Y, l_2) \in T \mid l_{smi} \neq l_1 \text{ and } l_{smi} \neq l_2\}$$

Then, $\mathcal{A}_{tmi} \sim \mathcal{A}_{smi}$ iff every initial and finite execution fragment of $\mathcal{A}$ ending in the location $l_{smi} \in L$, takes the same discrete transition, with the same transition $t_{tmi} = (l, a, \phi, Y, l_{smi}) \in T$ for some occurrence of the location l_{smi}.

Verifying the condition in Theorem 1 to prevent that TMI and SMI induce duplicates is costly. Therefore, we define a relaxed condition in Proposition 1 (see proofs in companion website) permitting to avoid some duplicate mutants using a breadth-first search algorithm in polynomial time.

Proposition 1 (Duplicate mutants)**.** *Let $\mathcal{A}$ be a TAIO and the mutants $\mathcal{A}_{tmi}$ and $\mathcal{A}_{smi}$ where:*

$$\mathcal{A}_{tmi} \in \mathcal{M}_{tmi}(\mathcal{A}) \text{ such that } \mathcal{A}_{tmi} = (L, l_0, X, \Sigma_I, \Sigma_O, T \setminus \{t_{tmi}\}, I),\ t_{tmi} = (l_1, a_{tmi}, \phi, Y, l_2) \in T \text{ and } a_{tmi} \in \Sigma_I, \text{ and } \mathcal{A}_{smi} \in \mathcal{M}_{smi}(\mathcal{A}) \text{ such that } \mathcal{A}_{smi} = (L \setminus \{l_{smi}\}, l_0, X, \Sigma_I, \Sigma_O, T_{smi}, I),\ l_{smi} \in L, l_{smi} \neq l_0, T_{smi} = \{(l_1, a, \phi, Y, l_2) \in T \mid l_{smi} \neq l_1 \text{ and } l_{smi} \neq l_2\}$$

If every possible initial finite execution fragment of $\mathcal{A}$ ending in location l_{smi} has the same previous location $l'_{smi} \neq l_{smi}$ for some l_{smi} occurrence and l'_{smi} only has one edge $t_{tmi} = (l'_{smi}, a, \phi, Y, l_{smi})$ to l_{smi}, then $\mathcal{A}_{tmi}$ and $\mathcal{A}_{smi}$ are duplicates.

Since Proposition 1 is not a sufficient condition, we cannot prevent some duplicates with this condition. We depict in Fig 5 and 6 examples where Proposition 1 cannot prevent duplicates.

B. Detecting Duplicate Mutants

Mutant duplicates are a well-known issue in mutation testing: empirical studies report that between 20% and 30% of all generated mutants are duplicates [41], [37], which affects mutation testing effectiveness [32]. In addition, to be computationally tractable, the SMI-NR is incomplete. Therefore, we present an approach to detect and remove duplicate mutants after mutant generation by using a *timed bisimulation algorithm* [12], [40]. MUPPAAL uses Ortiz *et al.*' timed bisimulation algorithm [40]. Because timed bisimulation's complexity is

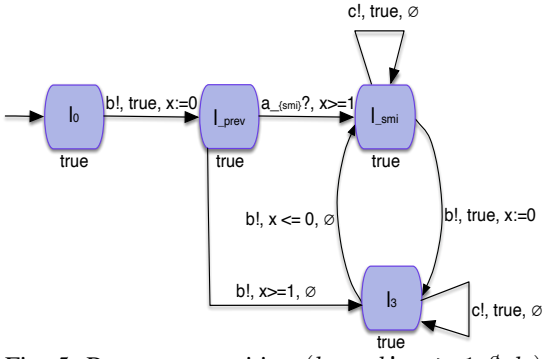

Fig. 5: Removes transition $(l_{prev}, b!, x \geq 1, \emptyset, l_3)$.

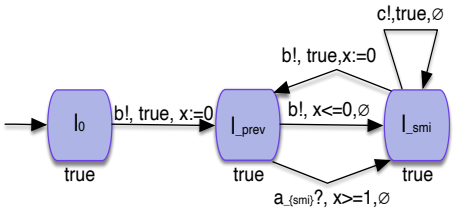

Fig. 6: Removes location l_3 from Fig 5.

EXPTIME [12], if the bisimulation process takes longer than the specified time to analyse a pair of mutants, our algorithm will stop the bisimulation process.

Algorithm 1 describes how MUPPAAL detects duplicates using timed bisimulation [40]. It works as follows. At the first iteration, it checks if there is no timeout involved when comparing the pair of two mutants and if these mutants are bisimilar using the BisimilarAlgo (line 9). Then, it updates $\mathcal{MU}^b$ with one of the two bisimilar mutants (line 11). For each pair of mutants in $\mathcal{LM}$, it assesses whether the two mutants are bisimilar and updates $\mathcal{MU}^b$ accordingly. Otherwise, if there is a timeout involved in the comparison of the pair of two mutants (line 19), then it updates $\mathcal{MU}^{tm}$ (line 14). At the second iteration, it checks if the pair of the two following mutants are bisimilar, up to the n_{th} iteration. Finally, algorithm 1 returns a pair with two sets, where $\mathcal{MU}$ is a set of non-duplicate mutants and $\mathcal{MU}^{tm}$ is a set of mutants whose analysis ended with a timeout.

C. Suggesting Duplicate Mutants

As stated before, timed bisimulation is computationally costly (EXPTIME [12]). To assess this complexity in practice, we present a baseline approach based on a simulation that *suggests* (it is not exact) mutants as potential duplicates. MUPPAAL uses the tools UPPAAL and UPPAAL-TRON to automatically suggest duplicate mutants.

1) Random Simulation.: To suggest duplicates, we take a pair of mutants, generate a random set of traces from one of the two mutants and run them on the other mutant model and reciprocally [16]. We check whether the mutants accept these traces (i.e., whether the mutants can simulate the actions and delays). If a simulation trace fails to run on one of the models, we deduce that the mutants cannot be bisimilar. However, if

```
1  Input: A list of mutants LM and a set of
       mutants MU
2  Output: A pair with two sets (no duplicate and
       timeout mutants ended).
3  MU^b ={}; A set of bisimilar mutants
4  MU^tm ={}; A set of mutants ended with timeout
5  for(i=0; i<LM.size()-1; i++){
6   for(j=i+1; j<LM.size(); j++){
7    //Execute timed bisimulation algorithm
8       if((!timeout()) && (BisimilarAlgo(LM[i],
       LM[j]))){
9         // Add any of the two bisimilar mutants
10        MU^b.add(LM[i]);
11         }else if(timeout()){
12        // Add the i-th mutant ended with timeout
13         MU^tm.add(LM[i],LM[j]);
14         }else{
15          skip; }
16   }
17 }
18 MU = MU - MU^b - MU^tm;
19 return pair(MU, MU^tm);
```

Algorithm 1: Bisimulation Process.

```
1  Input: A mutant M_1 ∈ MU^tm, number of traces to
       generate N, and a simulation time k
2  Output: An array with UPPAAL-TRON traces
3  T=[N];   // The set of N UPPAAL traces
4  T'=[N];  // The set of N UPPAAL-TRON traces
5  for(i=0;i<N; i++){
6      //The seed for the pseudo-random generator
7      r = random();
8      //Get random trace from UPPAAL
9      T[i]=VerifyTA(M_1, k, r);
10     //Translate trace to UPPAAL-TRON format
11     tree=parser(lexer(T[i]));
12     T'[i] = tree.format();
13 }
14 return T';
```

Algorithm 2: Trace generation.

```
1  Input: A set of mutants MU^tm, a number of traces
       to generate N and a simulation time k
2  Output: A set of not-bisimilar mutants
3  // The set of N UPPAAL-TRON traces per mutant
4  T'=[MU^tm.size()][N];
5  for(i=0; i<MU^tm.size(); i++){
6      T'[i]=TraceGeneration(MU^tm[i], N, k);
7      }
8  NBM = [()] // List of no bisimilar mutants
9  for(i=0; i<MU^tm.size()-1; i++){
10     for(j=i+1; j<MU^tm.size(); j++){
11         for(k=0; k<N; k++){
12             Pass1=Tron.check(MU^tm[i], T'[j,k]);
13             Pass2=Tron.check(MU^tm[j], T'[i,k]);
14             if(!(Pass1∧Pass2))
15               NBM=NBM.add((MU^tm[i]));
16         }
17     }
18 }
19 return NBM;
```

Algorithm 3: Random Trace Simulation Algorithm.

all simulation traces are accepted, we consider mutants as probably bisimilar (i.e., we cannot guarantee the existence of the bisimilarity relation).

We use the query *simulate [<=**k**; **N**] 1* using UPPAAL to get traces which simulate **k** units of time and getting **N** traces. Then, we use UPPAAL-TRON to check the validity of the traces of one mutant into the other [21]. To perform trace simulations on UPPAAL-TRON, our translator tool uses ANTLR [44] to parse and translate the traces from UPPAAL into a *preamble* file and a *trace* file. UPPAAL-TRON needs these two files to monitor an execution: (1) the *preamble* file provides the required definitions to configure and prepare UPPAAL-TRON for test execution of the trace, and (2) the *trace* file, which is a sequence of actions and delays to check if the model can execute. Given two mutants, *Automaton* $\mathcal{A}$ and *Automaton* $\mathcal{B}$, our tool proceeds as follows (for **N** random traces): (1) it takes the *Automaton A* to generate traces using UPPAAL. Then the tool reads them and parses trace t_i, builds the preamble, and per each t_i it makes a t_i' file with the UPPAAL-TRON format. (2) once traces are created, the tool uses UPPAAL-TRON to check if the *Automaton* $\mathcal{B}$ can execute the trace, returning as output Passed or Failed. Hence, all pairs not accepting random traces are not bisimilar.

2) Random Simulation Algorithms.: Algorithm 2 describes the random trace generation process of MUPPAAL. The algorithm works as follows: at the first iteration, it computes the N random traces for the first mutant present in $\mathcal{MU}^{tm}$ (line 7) with a simulation time k and a random number generator r (line 9). In addition, we translate the random traces generated from UPPAAL format to UPPAAL-TRON format (lines 13-14). The algorithm repeats until it produces N random traces for all mutants in $\mathcal{MU}^{tm}$ with a simulation time k and a random number generator r.

Algorithm 3 describes the random trace simulation process and uses Algorithm 2 to compute their UPPAAL-TRON traces. The algorithm works as follows. At the first iteration, it checks the N random traces generated by the second mutant $\mathcal{MU}^{tm}[j]$ on the first mutant $\mathcal{MU}^{tm}[i]$ (lines 10-12). In addition, if a pair of mutants are not bisimilar, $\mathcal{PBM}$ is updated with the ith mutants (line 16). At the second iteration, it checks the N random traces generated by the other two mutants in $\mathcal{MU}^{tm}$ and so on up to the n_{th} iteration.

IV. Evaluation

A. MUPPAAL Tool

MUPPAAL automates the whole mutation testing process on top of the UPPAAL verification tools. The tool is written in Java 8 and supports all the operators proposed by Basile *et al.* [8] plus the SMI-NR operator and is easily extendable to new ones. It uses the ANTLR library to parse the model and generate syntactically correct and non-equivalent mutants (thanks to the operators). It then proceeds to duplicate mutants analysis. MUPPAAL is available on our companion website.

B. Case Studies

Our studies stem from UPPAAL specifications of these cases and are available at https://github.com/farkasrebus/XtaBenchmarkSuite. For each case study, we consider the biggest and principal automaton (or process in UPPAAL) from the automata network.

Gear Control (GC). The GC models a simple gear controller for vehicles [36]. The GC model contains 24 states, of which 10 have invariants. All invariants are of the form $x \leq c$ for a clock x and constant c. There are 30 transitions, of which two have guards of the form $x < c$ and two have guards of the form $x \geq c$, for some clock x and constant c.

Collision Avoidance (CA). The CA case models a protocol where different agents want to get access to Ethernet through a shared channel [27]. The CA model has six states and 12 transitions, of which nine have guards of the form $x == c$ and four have guards of the form $x < c$, for some clock x and constant c.

Train Gate Controller (TGC). The TGC models a railway system that controls access to a bridge for several trains [5]. The bridge is a shared resource accessible by only one train at a time. The TGC model has 14 states, all of which have invariants. All invariants are of the form $x < c$ for a clock x and constant c. There are 18 transitions, of which four have guards of the form $x < c$, and four have guards of the form $x > c$, for a clock x and constant c.

A combined Gear control (CGC). The CGC models a (manually) combined gear controller for vehicles [36]. The CGC model contains 85 states, of which 20 have invariants. All invariants are of the form $x \leq c$ for a clock x and constant c. There are 120 transitions, of which ten have guards of the form $x < c$, and 10 have guards of the form $x \geq c$, for some clock x and constant c.

	GC	CA	TGC	CGC
TMI	13	9	14	36
TAD	501	26	179	1,625
SMI	12	2	12	27
SMI-NR	3	0	2	5
CXL	0	1	4	4
CXS	2	1	4	6
CCN	2	2	8	10
Total	533	41	222	1713

TABLE II: Number of generated mutants per operator

C. Research Questions

To evaluate the MUPPAAL workflow depicted in Fig 7, we consider the following research questions:

- **RQ1:** How does random trace simulation compare to timed bisimulation to identify duplicates?
- **RQ2:** What is the scalability and performance of timed bisimulation compared to random trace simulation?
- **RQ3:** How does our novel SMI-NR operator compare to the original SMI operator?

In Fig 7 on our four cases (see Section IV-B). For each case, we first generate (step 1) a set of non-equivalent mutants

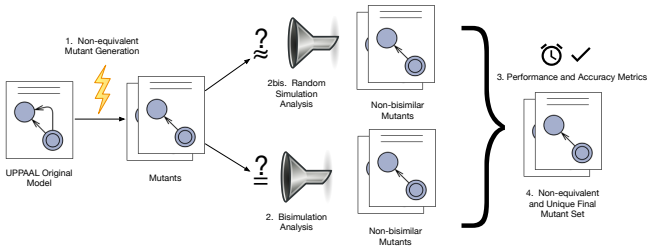

Fig. 7: Experimentation Workflow

($\mathcal{MU}$) using the operators presented in Table I (Basile *et al.* [8]) and our novel operator SMI-NR. This results in 2509 mutants as presented in Table II. Then, we independently apply our timed bisimulation and random trace simulation algorithms on this set (steps 2, 2bis). For both cases: we fix the maximum computation budget to 2100 seconds per pair of mutants or 12GB of RAM, whichever limit the analysis reached first. Regarding random trace simulation, we have three settings: 1) two traces per model and 100-time units; 2) 10 traces per model and 1000 time units; 3) 100 traces per model and 10000-time units. We run each setting ten times to mitigate randomness effects. In step 3, we collect the execution times and the number of duplicates and likely duplicates for analysis (**RQ1** & **RQ2**). We use timed bisimulation to compare SMI-NR and SMI mutants (**RQ3**). We ran our experiments on a UBUNTU 21.10 $\times$ 86_64 GNU/Linux machine with 16 cores, 2.2 GHz, 32GB RAM.

D. Results and Discussion

Case	GC	CA	TGC	CGC
ratio Bisimulation	41/533	12/41	71/222	373/1,713
ratio Trace (**N**=2,**k**=100,**E**=10)	432/533 (**st**=32.2)	38/41 (**st**=9.2)	152/222 (**st**=14.7)	1327/1,713 (**st**=38.3)
ratio Trace (**N**=10, **k**=1000)	247/533 (**st**=18.1)	38/41) (**st**=3.8)	119/222 (**st**=20.7)	774/1,713 (**st**=32.8)
ratio Trace (**N**=100, **k**=10000)	206/533 (**st**=37.4)	27/41) (**st**=10.0)	108/222 (**st**=13.1)	664/1,713 (**st**=57.0)

TABLE III: Proportion of mutant duplicates. For random trace simulation, we report the average with standard deviation (**st**)

1) Answering RQ1.: Table III reveals that mutant duplicates represent up to 32% of the total number of mutants, justifying the need for duplicate prevention and removal techniques. In general, random trace simulation overestimates the number of duplicates up to an order of magnitude. Drastically increasing the number of traces and time units yield only limited improvements. We conclude that *random trace simulation suggests too many duplicates*.

2) Answering RQ2.: Timed bisimulation is EXPTIME-complete, implying that some comparisons could exceed our computation budget. Table IV reports that only 34 mutant comparisons (amongst more than 1.5 million) timed out for CGC when running bisimulation. All settings of the random baseline scaled up. Yet, the largest one is up to 19 times slower for 3 out of the 4 cases. *Timed bisimulation has*

Case	TT	TB	TR (s)	BI (s)
GC	0	0	0.153 (**N**=2, **k**=100, **E**=10, st=0.007) 0.548 (**N**=10, **k**=1000, **E**=10, **st**=0.07) 6.38 (**N**=100, **k**=10000, **E**=10, **st**=0.08)	3.94 (st=0.54)
CA	0	0	0.040 (**N**=2, **k**=100, **E**=10, st=0.006) 1.16 (**N**=10, **k**=1000, **E**=10, **st**=0.07) 10.5 (**N**=100, **k**=10000, **E**=10, **st**=0.23	2.31 (st=0.71)
TGC	0	0	0.018 (**N**=2, **k**=100, **E**=10, st=0.002) 0.13 (**N**=10, **k**=1000, **E**=10, **st**=0.014) 1.69 (**N**=100, **k**=10000, **E**=10, **st**=0.20	2.83 (st=1.05)
CGC	0	34	0.803 (**N**=2, **k**=100, **E**=10, st=0.10) 1.91 (**N**=10, **k**=1000, **E**=10, **st**=0.17) 337.2 (**N**=100, **k**=10000, **E**=10, **st**=10)	17.58 (**st**=3.9)

TABLE IV: The total number of pairs of mutants ended by timeout of traces (TT), ended by timeout of bisimulation (TB), the average execution time(s) using traces (TR), using bisimulation (BI), the number of traces (**N**), of runs (**E**), the units of time (**k**), and the standard deviation (st).

	GC	CA	TGC	CGC
# SMI mutants	12	2	12	27
# SMI duplicates	9	2	10	22
# SMI-NR mutants	3	0	2	5
# Bisimilar pairs SMI-NR-SMI	3	0	2	5

TABLE V: SMI-NR and SMI Operators Comparison

*good scalability overall. Considering **RQ1**, it offers the best compromise between accuracy and performance.*

3) Answering RQ3.: Table V compares the SMI and SMI-NR mutants. SMI can generate a large proportion of duplicates (up to 83%) while SMI-NR by design does not produce any duplicate. The two last rows of Table V allow observing that SMI-NR produce unique mutants while preserving the behaviour of the SMI operator. We conclude that the SMI-NR operator *offers a viable alternative to SMI, introducing the same faults while preventing duplicates.*

E. Threats to Validity

Internal validity. We selected four cases of different natures: a gear controller, a network communication model avoiding collisions, and a train gate controller. These models have different sizes and numbers of clock constraints. They enabled us to observe differences in detecting and removing duplicate mutants. **Construct validity.** We chose our baseline settings to expose diverse tradeoffs between performance and accuracy concerning timed bisimulation. We did not explore larger values of **N** and **k** since accuracy only marginally improved for even higher execution times. We ran each comparison ten times to mitigate randomness effects.
External validity. We cannot guarantee that our results extend to all timed systems expressed in UPPAAL. Our cases were enough to assess diversity regarding mutants types and their analysis times.

V. Related Work

Several works cover the long-standing equivalent mutant problem [39], [16], [37], [41], [28]. Interest in the mutant duplicate problem is more recent [41], mostly at the code level [32]. MBMT gained traction more recently [17], [16], [19], [1]. Researchers applied MBMT for timed specifications [38], [25], [2], [8], [33], [47]. In [38], the authors present six mutation operators for TA, but do not guarantee the absence of equivalent or duplicate mutants. Aichernig *et al.* [2] design eight mutation operators for TA based on [25]. Again, these operators do not prevent generating equivalent or duplicate mutants. Basile *et al.* introduced six mutation operators for TS [8]. These mutation operators follow the same construction as those defined in [2], [25]. However, Basile *et al.* use a timed refinement technique to avoid the generation of equivalent mutants but do not address mutant duplicates [7], [8]. Larsen *et al.* defined an MBMT technique [33] on top of the UPPAAL-ECDAR verification tool [14]. It also uses refinement checking to eliminate equivalent mutants but does not address duplicates. Aichernig *et al.* designed an MBMT tool called MoMuT::TA [1]. MoMuT::TA maps TA to formal semantics and performs a conformance check between mutants and the original model to generate test cases automatically. The tool UPPAL-TRON [21] is an addition to the UPPAAL environment. One can also use it to handle conformance tests on TS. UPPAL-TRON simulates the IUT with input deemed relevant by the model, monitors the outputs, and checks the conformance of these against the behaviour specified in the model. Hessel and Pettersson proposed an MBMT tool called Cover [24]. Cover generates test-cases based on TA and Timed Computation Tree Logic (TCTL). One uses properties written in TCTL to verify the test model. Similar approaches exist [31], [22]. μUTA introduces a test generation method to derive mutants from the specification and executes them via online testing. It focuses on robustness testing of web services [45].

VI. Conclusion

In this paper, we proposed MUPPAAL, a mutation tool suite for model-based timed systems. It integrates equivalence-avoiding operators and focuses on alleviating the mutant duplicate problem (up to 32% of all mutants). MUPPAAL implements a novel duplicate reduction operator and a timed bisimulation algorithm. Our tool offers the best compromise between performance and accuracy compared to a random baseline. In the future, we will design more duplicate-avoiding operators and extend mutations to networks of timed automata.

Acknowledgment

Gilles Perrouin is an FNRS (Fonds National de la Recherche Scientifique) Research Associate. Jaime Cuartas received support from ERASMUS+ while at the University of Namur. Maxime Cordy obtained funding from FNR Luxembourg (grant INTER/FNRS/20/15077233/Scaling Up Variability/Cordy). Work partially funded by ERDF project IDEES. We thank Paul Temple for the early discussions on this work.

References

[1] B. K. Aichernig, J. Auer, E. Jöbstl, R. Korosec, W. Krenn, R. Schlick, and B. V. Schmidt. Model-based mutation testing of an industrial measurement device. In *Tests and Proofs*, volume 8570 of *LNCS*, pages 1–19. Springer, 2014.

[2] B. K. Aichernig, F. Lorber, and D. Nickovic. Time for mutants - model-based mutation testing with timed automata. In M. Veanes and L. Viganò, editors, *Tests and Proofs - 7th International Conference, TAP 2013, Budapest, Hungary, June 16-20, 2013. Proceedings*, volume 7942 of *Lecture Notes in Computer Science*, pages 20–38. Springer, 2013.

[3] B. K. Aichernig, F. Lorber, and D. Ničković. Time for mutants - model-based mutation testing with timed automata. *Tests and Proofs Lecture Notes in Computer Science*, pages 20–38, 2013.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[5] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

[6] D. Basile, M. H. t. Beek, M. Cordy, and A. Legay. Tackling the equivalent mutant problem in real-time systems. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*. ACM, Oct 2020.

[7] D. Basile, M. H. t. Beek, S. Lazreg, M. Cordy, and A. Legay. Static detection of equivalent mutants in real-time model-based mutation testing. *Empirical Software Engineering*, 27(7):160, 2022.

[8] D. Basile, M. H. ter Beek, M. Cordy, and A. Legay. Tackling the equivalent mutant problem in real-time systems: the 12 commandments of model-based mutation testing. In R. E. Lopez-Herrejon, editor, *SPLC '20: 24th ACM International Systems and Software Product Line Conference, Montreal, Quebec, Canada, October 19-23, 2020, Volume A*, pages 30:1–30:11. ACM, 2020.

[9] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[10] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In Hu, A. J. Vardi, and M. Y., editors, *Computer Aided Verification 10th International Conference, CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–549, Vancouver, BC, Canada, June 1998.

[11] T. A. Budd and A. S. Gopal. Program testing by specification mutation. *Computer Languages*, 10(1):63–73, Jan. 1985.

[12] K. Cerāns. Decidability of bisimulation equivalences for parallel timer processes. In G. von Bochmann and D. K. Probst, editors, *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV'92)*, volume 663 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, 1993.

[13] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, third edition*. The MIT Press. MIT Press, 2009.

[14] A. David, K. Larsen, A. Legay, U. Nyman, and A. Wasowski. Ecdar: An environment for compositional design and analysis of real time systems. In *Lecture Notes in Computer Science*, volume 6252/2010, Germany, 2010.

[15] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed i/o automata: A complete specification theory for real-time systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 91–100, New York, NY, USA, 2010. ACM.

[16] X. Devroey, G. Perrouin, M. Papadakis, A. Legay, P.-Y. Schobbens, and P. Heymans. Model-based mutant equivalence detection using automata language equivalence and simulations. *Journal of Systems and Software*, 2018.

[17] X. Devroey, G. Perrouin, M. Papadakis, P.-Y. Schobbens, and P. Heymans. Featured Model-based Mutation Analysis. In *International Conference on Software Engineering, ICSE*, Austin, TX, USA, 2016.

[18] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics 173. Springer-Verlag Berlin Heidelberg, 5 edition, 2017.

[19] S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero. Mutation testing applied to validate specifications based on statecharts. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, ISSRE '99, pages 210–, Washington, DC, USA, 1999.

[20] G. Fraser and A. Arcuri. Achieving scalable mutation-based generation of whole test suites. *Empirical Software Engineering*, pages 1–30, 2014.

[21] K. Guldstrand Larsen, M. Mikucionis, and B. Nielsen. Uppaal tron user manual - docs.uppaal.org, Jun 2017.

[22] T. R. Gundersen, F. Lorber, U. Nyman, and C. Ovesen. Effortless fault localisation: Conformance testing of real-time systems in ecdar. *Electronic Proceedings in Theoretical Computer Science*, 277:147–160, Sep 2018.

[23] T. A. Henzinger, P.-H. Ho, and H. Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.

[24] A. Hessel and P. Pettersson. Cover-a test-case generation tool for timed systems. *Testing of software and communicating systems*, pages 31–34, 2007.

[25] R. M. Hierons, S. Counsell, and M. AbouTrab. Specification mutation analysis for validating timed testing approaches based on timed automata. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 660–669, Los Alamitos, CA, USA, jul 2012.

[26] W. E. Howden. Reliability of the path analysis testing strategy. *IEEE Transactions on Software Engineering*, 2(3):208–215, 1976.

[27] H. Jensen, K. Larsen, and A. Skou. Modelling and analysis of a collision avoidance protocol using spin and uppaal. *BRICS Report Series*, 3, 01 2002.

[28] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Trans. Softw. Eng.*, 37(5):649–678, Sept. 2011.

[29] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *FSE 2014: Proceedings of the ACM SIGSOFT 22nd Symposium on the Foundations of Software Engineering*, pages 654–665, Hong Kong, Nov. 2014.

[30] D. K. Kaynar, N. A. Lynch, R. Segala, and F. W. Vaandrager. *The Theory of Timed I/O Automata, Second Edition*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2010.

[31] J. H. Kim, K. G. Larsen, B. Nielsen, M. Mikucionis, and P. Olsen. Formal analysis and testing of real-time automotive systems using UPPAAL tools. In M. Núñez and M. Güdemann, editors, *Formal Methods for Industrial Critical Systems - 20th International Workshop, FMICS 2015, Oslo, Norway, June 22-23, 2015 Proceedings*, volume 9128 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2015.

[32] B. Kurtz, P. Ammann, J. Offutt, and M. Kurtz. Are we there yet? how redundant and equivalent mutants affect determination of test completeness. In *Ninth IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2016, Chicago, IL, USA, April 11-15, 2016*, pages 142–151. IEEE Computer Society, 2016.

[33] K. G. Larsen, F. Lorber, B. Nielsen, and U. M. Nyman. Mutation-based test-case generation with ecdar. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 319–328, March 2017.

[34] K. G. Larsen, F. Lorber, B. Nielsen, and U. M. Nyman. Mutation-based test-case generation with ecdar. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 319–328, March 2017.

[35] K. G. Larsen, M. Mikucionis, and B. Nielsen. Testing real-time embedded software using uppaal-tron: an industrial case study. In *the 5th ACM international conference on Embedded software*, pages 299 – 306. ACM Press New York, NY, USA, September 18–22 2005.

[36] M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear controller. *International Journal on Software Tools for Technology Transfer*, 3(3):353–368, Aug 2001.

[37] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Jozala. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *IEEE Trans. Software Eng.*, 40(1):23–42, 2014.

[38] R. Nilsson, J. Offutt, and S. F. Andler. Mutation-based testing criteria for timeliness. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01*, COMPSAC '04, pages 306–311, Washington, DC, USA, 2004.

[39] J. Offutt. A mutation carol: Past, present and future. *Information and Software Technology*, 53(10):1098–1107, Oct. 2011.

[40] J. J. Ortiz, M. Amrani, and P. Schobbens. Multi-timed bisimulation for distributed timed automata. In C. Barrett, M. Davies, and T. Kahsai, editors, *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, volume 10227 of *Lecture Notes in Computer Science*, pages 52–67, 2017.

[41] M. Papadakis, Y. Jia, M. Harman, and Y. Le Traon. Trivial compiler equivalence: A large scale empirical study of a simple fast and effective equivalent mutant detection technique. In *International Conference on Software Engineering, ICSE*, pages 936–946. IEEE, 2015.

[42] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman. Mutation testing advances: An analysis and survey. *Advances in Computers*, 112, 2018.
[43] M. Papadakis and N. Malevris. Automatic mutation test case generation via dynamic symbolic execution. In *ISSRE*, pages 121–130. IEEE, 2010.
[44] T. Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, Raleigh, NC, 2 edition, 2013.
[45] F. Siavashi, J. Iqbal, D. Truscan, and J. Vain. *Testing Web Services with Model-Based Mutation*, page 45–67. Springer, 2017.
[46] J. Tretmans. *Formal Methods and Testing – An Outcome of the FORTEST Network (Revised Papers Selection)*, chapter Model Based Testing with Labelled Transition Systems, pages 1–38. Springer-Verlag, 2008.
[47] J. J. O. Vega, G. Perrouin, M. Amrani, and P.-Y. Schobbens. Model-based mutation operators for timed systems: A taxonomy and research agenda. *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2018.
[48] J. M. Voas and G. McGraw. *Software Fault Injection: Inoculating Programs Against Errors*. John Wiley & Sons, Inc., 1997.
[49] J. Zander, I. Schieferdecker, and P. J. Mosterman. *Model-Based Testing for Embedded Systems*. CRC Press, 2017.

APPENDIX

A. Description of mutants

A mutation operator is a function that maps a specific TAIO to a set of TAIO's. The mutants generated by our tool (for UPPAAL) follow the guidelines of the operators proposed by [6].

Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator is a function $\mathcal{M}_\mu$ that generates a set of mutants from a TAIO. We use μ to refer to each specific operator presented in [8].

Definition 8 (TMI operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator TMI ($\mu = tmi$) is a function $\mathcal{M}_{tmi}$ that removes a transition at each possible mutant TAIO.*

$$\mathcal{M}_{tmi}(\textit{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{e\}, I) \mid e \in E, \\ a \in \Sigma_I\}$$

such that:

- $e = (l_1, a, \phi, Y, l_2) \in T$*, the removed transition is a transition of the original model for some* (l_1, a, ϕ, Y, l_2)*,*
- $a \in \Sigma_I$*, the removed transition has an input action.*

Definition 9 (TAD operator)**.** *Let $\mathbb{A}$ be a set of all possibles ATIO. A mutation operator TAD ($\mu = tad$) is a function that adds a transition at each possible mutant TAIO.*

$$\mathcal{M}_{tad}(\textit{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \cup \{e_{tad}\}, I) \mid \\ t_{tad} \notin T, l_1, l2 \in L, a \in \Sigma_C\}$$

such that:

- $t_{tad} = (l_1, a, \phi, Y, l_2) \notin T$ *the added transition is not already a transition in* T*,*
- $a \in \Sigma_C$*, the removed transition has an output or internal (locally-controlled) action.*

Definition 10 (SMI operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator SMI ($\mu = smi$) is a function $\mathcal{M}_{smi}$ that removes a location at each possible mutant TAIO.*

$$\mathcal{M}_{smi}(\textit{TAIO}) = \{(L \setminus \{l_{smi}\}, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T', I') \mid \\ l_{smi} \in L \setminus \{l_0\}, \exists t = (l', a, \phi, Y, l_{smi}) \in T \text{ and } a \in \Sigma_I\}$$

such that:

- $l_{smi} \in L \setminus \{l_0\}$*, the removed location is not an initial location,*
- $\exists\ e = (l', a, \phi, Y, l_{smi})$*, exists a transition $e \in T$ that has an input action $a \in \Sigma_I$ with l_{smi} as a target location, and some l', ϕ, Y,*
- $T' = \{(l_1, a, \phi, Y, l_2) \mid (l_1, a, \phi, Y, l_2) \in T, l \neq l_1, l \neq l_2\}$*, as a consequence of remove a location, the new set of edges does not have the removed location,*
- $I' : L \setminus \{l\} \to \phi(X)$*, where $l \in L$ from the original model. The locations that are not removed in the mutant keep the same invariant.*

Definition 11 (CXL operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator CXL ($\mu = cxl$) is a function $\mathcal{M}_{cxl}$ that increases the constant of clock guard at each possible mutant TAIO.*

$$\mathcal{M}_{cxl}(\textit{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, (T \setminus \{t\}) \cup \{t'\}, I) \mid \\ t \in T,\ t' \notin T\}$$

such that:

- $t = (l_1, a, \phi, Y, l_2) \in T$
- $t' = (l_3, a, \phi', Y, l_4) \notin T$
- *For one of the next two cases*
  - *Output action*
    - $a \in \Sigma_O$*,*
    - $\phi = x \leq k$*, where $x \in X$ is a clock, k is a constant, $\nu(x)$ + d is the clock valuation for the clock x and $d > 0$,*
    - $\phi' = x \leq k + \epsilon$*, for $\epsilon > 0$, where exists at least a clock $k < x' \leq k + \epsilon$, k is a constant, $\nu(x)$ + d is the clock valuation for the clock x and $d > 0$, such that $\nu \models I(l_3)$, $\nu' = \nu[Y \to 0]$ and $\nu' \models I(l_4)$.*
  - *or*
  - *Input action*
    - $a \in \Sigma_I$*;*
    - $\phi = x \geq k$*, where $x \in X$ is a clock, k is a constant, $\nu(x)$ + d is the clock valuation for the clock x and $d > 0$,*
    - $\phi' = x \geq k + \epsilon$*, for $\epsilon > 0$, where exists at least a clock $k < x' \leq k + \epsilon$, k is a constant, $\nu(x)$ + d is the clock valuation for the clock x and $d > 0$, such that $\nu \models I(l_3)$, $\nu' = \nu[Y \to 0]$ and $\nu' \models I(l_4)$.*

Definition 12 (CXS operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator CXS ($\mu = cxs$) is a function $\mathcal{M}_{cxs}$ that decreases the constant of clock guard at each possible mutant TAIO.*

$$\mathcal{M}_{cxs}(\textsf{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, (T \setminus \{t\}) \cup \{t'\}, I) \mid$$

$$t \in T,\ t' \notin T\}$$

such that:

- $t = (l_1, a, \phi, Y, l_2) \in T$
- $t' = (l_3, a, \phi', Y, l_4) \notin T$
  For one of the next two cases
  - *Output action*
    - $a \in \Sigma_O$;
    - $\phi = x \geq k$, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$,
    - $\phi' = x \geq k - \epsilon$, *for* $\epsilon > 0$, *where exists at least a clock valuation* $k - \epsilon \leq x' < k$, k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$, *such that* $\nu \models I(l_3)$, $\nu' = \nu[Y \to 0]$ *and* $\nu' \models I(l_4)$, *and exists an initial finite run ending in the state* (l_3, ν').

    or
  - *Input action*
    - $a \in \Sigma_I$;
    - $\phi = x \leq k$, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$,
    - $\phi' = x \leq k - \epsilon$, *for* $\epsilon > 0$, *where exists at least a clock valuation* $k - \epsilon < x' \leq k$ *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$.

Definition 13 (CCN operator)**.** *Let* $\mathbb{A}$ *be a set of all possibles* TAIO. *A mutation operator* CCN ($\mu = ccn$) *is a function* $\mathcal{M}_{ccn}$*that negates a clock guard at each possible mutant* TAIO.

$$\mathcal{M}_{ccn}(\textsf{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, (T \setminus \{t\}) \cup \{t'\}, I, F) \mid$$

$$t \in T,\ t' \notin T\}$$

such that:

- $t = (l_1, a, \phi, Y, l_2) \in T$
- $t' = (l_3, a, \phi', Y, l_4) \notin T$
- *For one of the next four cases according to the form of* ϕ, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$.
  - *if* $\phi = x \leq k$, *then* $\phi' = x > k$,
  - *if* $\phi = x < k$, *then* $\phi' = x \geq k$,
  - *if* $\phi = x \geq k$, *then* $\phi' = x < k$,
  - *if* $\phi = x > k$, *then* $\phi' = x \leq k$.

B. Equivalent mutants

In 2010 [34] published a work that introduce the definition of *refinement* and the implementation of the theory in the tool UPPAAL TIGA. Here the notion of *refinement* allows comparing two specifications as well as relating an implementation to a specification. The refinement is a binary operator between TAIO and should satisfy the following condition. S_1 is a refinement of S_2 if is possible to replace S_2 with S_1 in every environment and obtain an equivalent system. Formally

Definition 14 (Refinement)**.** *Let* $\mathcal{A}_1 = (L^1, l_0^1, X^1, \Sigma_I^1, \Sigma_O^1, \Sigma^1, T^1, I^1)$ *and* $\mathcal{A}_2 = (L^2, l_0^2, X^2, \Sigma_I^2, \Sigma_O^2, \Sigma^2, T^2, I^2)$ *be* TAIO *with the states* S^1 *and* S^2, *respectively. Let* $v^1 : C^1 \to \mathbb{R}_{\geq 0}$, $v^2 : C^2 \to \mathbb{R}_{\geq 0}$ *as clock valuation functions, and* s_0^1, s_0^2 *as initial states of* $\mathcal{A}_1$, $\mathcal{A}_2$, *respectively.* $\mathcal{A}_1$ *is a refinement of* $\mathcal{A}_2$, *written* $\mathcal{A}_1 \leq \mathcal{A}_2$ *iff there exists a binary relation* $\mathcal{R} \subseteq (S^1 \times S^2)$ *containing* $s = (s_0^1, s_0^2)$ *such that for each pair of states* $(s^1, s^2) = ((l^1, v^1), (l^2, v^2)) \in R$, *it holds:*

1) ***whenever*** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^{2'})$ *for some* $l^{2'}$ *and* $a \in \Sigma_I^2$, ***then*** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^1)$ *for some* $l^{1'}$ *and* $((l^{1'}, v^{1'}), (l^{2'}, v^{2'})) \in \mathcal{R}$
2) ***whenever*** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^{1'})$ *for some* $l^{1'}$ *and* $a \in \Sigma_C^2$, ***then*** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^{2'})$ *for some* $l^{2'}$ *and* $((l^{1'}, v^1), (l^{2'}, v^2)) \in \mathcal{R}$
3) ***whenever*** $(l^1, v^1) \xrightarrow{d} (l^1, v^{1'})$ *for some* $v^{1'}$ *and* $d \in \mathbb{R}_{\geq 0}$, ***then*** $(l^2, v^2) \xrightarrow{d} (l^2, v^{2'})$ *for some* $v^{2'}$ *and* $((l^1, v^{1'}), (l^2, v^{2'})) \in \mathcal{R}$

[34] implemented the theory of refinement equivalence between TAIO in UPPAAL TIGA In the same year [33] published a tool called ECDAR, as an extension of UPPAAL-TIGA it reuses some of its components, including the refinement operation. In 2017 [33] published a work where they collect the mutation operators proposed [3] (for UPPAAL) and used ECDAR to check the conformance between the System Under Test and the mutants with the refinement operation.

Theorem 2 (Bisimulation is Finer than Refinement Equivalence)**.**
TAIO$_1 \sim$ TAIO$_2$ *implies* TAIO$_1 \leq$ TAIO$_2$.

Proof. Suppose that the theorem is false. Then there must be two TAIO for which the theorem fails. Let $\mathcal{A}_i = (L^i, l_0^i, X^i, \Sigma_I^i, \Sigma_O^i, \Sigma^i, T^i, I^i)$, $i = 1, 2$, be two TAIO, for sake of contradiction, that $\mathcal{A}_1$ and $\mathcal{TIO}\mathcal{A}_2$ are bisimilar and $\mathcal{A}_1$ does not refine $\mathcal{A}_2$. In this case, let S_1 and S_2 be any set of states of $\mathcal{A}_1$ and $\mathcal{T}\mathcal{A}_2$, respectively, and $R \subseteq S_1 \times S_2$. Since $\mathcal{A}_1$ does not refine $\mathcal{A}_2$, at least one of the following conditions must be satisfied (the negation of the conditions to be a refinement):

1) $(s_0^1, s_0^2) \notin \mathcal{R}$. For all initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$, where s_0^1 is an initial state of $\mathcal{A}_1$ and s_0^2 is an initial state of $\mathcal{A}_2$
2) **exists** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^2)$ for some $l^{2'}$ and $a \in \Sigma_I^2$, **then** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^1) \notin T^1$ for any $l^{1'}$ thus $((l^{1'}, v^1), (l^{2'}, v^2)) \notin \mathcal{R}$

3) **exists** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^1)$ for some $l^{1'}$ and $a \in \Sigma^1_C$, **then** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^2) \notin T^2$ for any $l^{1'}$ thus $((l^{1'}, v^1), (l^{2'}, v^2)) \notin \mathcal{R}$

4) **exists** $(l^1, v^1) \xrightarrow{d} (l^1, v^{1'})$ for some $v^{1'}$ and $d \in \mathbb{R}_{\geq 0}$, **then** $(l^2, v^2) \xrightarrow{d} (l^2, v^{2'}) \notin E^2$ for any $v^{2'}$ thus $((l^1, v^{1'}), (l^2, v^{2'})) \notin \mathcal{R}$

However, since $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar, $\forall s_1 \in l^1_0 (\exists s_2 \in l^2_0 . (s_1, s_2) \in \mathcal{R}$ and $\forall s_2 \in l^2_0 (\exists s_1 \in l^1_0 . (s_1, s_2) \in \mathcal{R})$. As l^1_0 and l^2_0 are non-empty sets, then at least one pair of initial states (s^1_0, s^2_0) of $\mathcal{A}_1$ and $\mathcal{A}_2$ must be in $\mathcal{R}$ for any automata. Thus, the first (1) condition does not hold. Since $\mathcal{A}_1 \sim \mathcal{A}_2$, by definition, for all, $(s_1, s_2) \in \mathcal{R}$ it holds that if exists $s_2 \xrightarrow{x} s'_2 \in T^2$ for any x then exists $s_1 \xrightarrow{x} s'_1 \in T^1$. Thus, if $x \in \Sigma^2_I$, then the condition holds and exists $s_1 \xrightarrow{x} s'_1 \in T^1$, and the second (2) condition does not hold. Since $\mathcal{A}_1 \sim \mathcal{A}_2$, by definition, for all, $(s_1, s_2) \in \mathcal{R}$ it holds that if exists $s_1 \xrightarrow{x} s'_1 \in T^1$ for any x then exists $s_2 \xrightarrow{x} s'_2 \in T^2$. Thus, if $x \in \Sigma^2_C$, then the condition holds and exists $s_2 \xrightarrow{x} s'_2 \in T^2$ and the third (3) condition does not hold. And if $x \in \mathbb{R}_{\geq 0}$, then the condition holds and exists $s_2 \xrightarrow{x} s'_2 \in T^2$ and the fourth (4) condition does not hold. Thus, is impossible: $\mathcal{A}_1 \sim \mathcal{A}_2$ and $\mathcal{A}_1 \not\leq \mathcal{A}_2$. Therefore, it must be the case of our assumption that $\mathcal{A}_1 \sim \mathcal{A}_2$ implies $\mathcal{A}_1 \leq \mathcal{A}_2$.

□

Theorem 3 (Non-bisimulation and Refinement Equivalence)**.** *For two any TAIO $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1 \leq \mathcal{A}_2$ does not always imply $\mathcal{A}_1 \sim \mathcal{A}_2$.*

Proof. The fact that $\mathcal{A}_1 \leq \mathcal{A}_2$ does not always imply $\mathcal{A}_1 \sim \mathcal{A}_2$ is illustrated by the following example.

Consider the $\mathcal{A}_1 = (L^1, l^1_0, X^1, \Sigma^1_I, \Sigma^1_O, \Sigma^1, T^1, I^1)$, where:

- $L^1 = \{l^1_0, l^1_1\}$,
- $l^1_0 = l_0$,
- $l^1_1 = l_1$,
- $X^1 = \emptyset$,
- $\Sigma^1_I = \{b?, c?\}$,
- $\Sigma^1_O = \{a!\}$,
- $\Sigma^1 = \Sigma^1_I \cup \Sigma^1_O$
- $T^1 = \{(l_0, a!, \emptyset, true, l_0), (l_0, b?, \emptyset, true, l_0), (l_0, c?, \emptyset, true, l^1_1)\}$,
- $I^1(l^1) = true$.

And consider the $\mathcal{A}_2 = (L^2, l^2_0, X^2, \Sigma^2_I, \Sigma^2_O, \Sigma^2, T^2, I^2)$, where:

- $L^2 = \{l^2_0, l^2_1\}$,
- $m_0 = l^2_0$,
- $m_1 = l^2_1$,
- $X^2 = \emptyset$,
- $\Sigma^2_I = \{b?\}$,
- $\Sigma^2_O = \{a!, d!\}$,
- $\Sigma^2 = \Sigma^2_I \cup \Sigma^2_O$
- $T^2 = \{(m_0, a!, \emptyset, true, m_0), (m_0, b?, \emptyset, true, m_0), (m_0, d!, \emptyset, true, m_1)\}$,
- $I^2(m_1) = true$.

Graphically, the $\mathcal{A}_1$ (left) and $\mathcal{A}_2$ (right) in Figure 10:

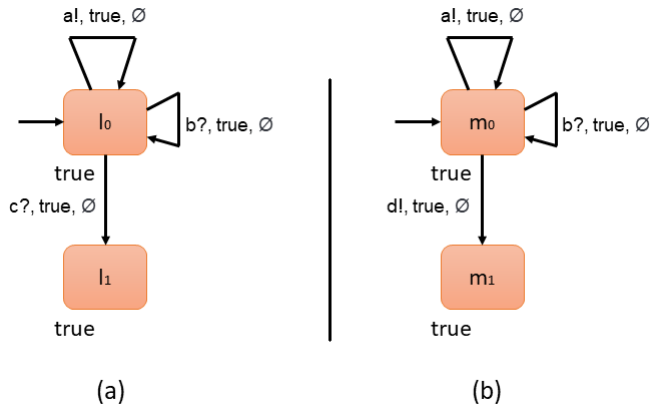

Fig. 8: Refinement equivalent, but not bisimulation equivalent TIOA's. (a) $\mathcal{A}_1$, (b) $\mathcal{A}_2$

As there is no bisimilar state in $\mathcal{A}_1$ that mimics (m_0, v^2) for any v^2, the only candidate would be the state (l_0, v^1) for some v^1 but (l_0, v^1) can not mimic $(m_0, v^2) \xrightarrow{d!} (m_1, v^2)$ since there is no transition in T^1 with action $d!$ and $d! \notin \Sigma^1_O$. Similarly, there is no bisimilar state in $\mathcal{A}_2$ that mimics (l_0, v^1) for any v^1, here the only candidate would be the state (m_0, v^2) for some v^2 but (m_0, v^2) can not mimic $(l_0, v^1) \xrightarrow{c?} (l_1, v^1)$ since there is no transition in T^2 with action $c?$ and $c? \notin \Sigma^2_I$. However, $\mathcal{A}_1$ and $\mathcal{A}_2$ are refinement-equivalent, as every delay and output action in $\mathcal{A}_1$ can be mimicked by $\mathcal{A}_2$, and every input action in $\mathcal{A}_2$ can be mimicked by $\mathcal{A}_1$.

Thus, $\mathcal{A}_1 \not\sim \mathcal{A}_2$ and $\mathcal{A}_1 \leq \mathcal{A}_2$ is possible.

□

Two immediate corollaries to Theorem 5, and Theorem 6 are:

Corollary 1 (Bisimulation is Strictly Finer than Refinement Equivalence)**.** *$\mathcal{A}_1 \sim \mathcal{A}_2$implies $\mathcal{A}_1 \leq \mathcal{A}_2$, but $\mathcal{A}_1 \not\sim \mathcal{A}_2$ and $\mathcal{A}_1 \leq \mathcal{A}_2$ is possible.*

Corollary 2 (Non-bisimulation implies Non-refinement equivalence)**.** *$\mathcal{A}_1 \not\leq \mathcal{A}_2$ implies $\mathcal{A}_1 \not\sim \mathcal{A}_2$.*

C. Duplicate mutants(TMI and SMI operators)

In this section, we will consider the mutants generated by TMI and SMI operators that are duplicates. We define these operators in D section.

1) When a Timed automaton with Input and Output produces a SMI mutant duplicate to TMI mutant duplicate ?: Since we are considering TAIO, and every mutant of the set $\mathcal{M}_{smi}$(TAIO) is not a refinement of TAIO, and as a consequence, they are not bisimilar. Indeed, there is no state in the mutants $\mathcal{M}_{smi}$(TAIO) that can mimic their respective missing state. There is possible to find mutants in the set $\mathcal{M}_{tmi}$(TAIO) where the missing transition makes some state unreachable, generating a mutant bisimilar to some other mutant of the set $\mathcal{M}_{smi}$(TAIO). The following propositions describe the condition of the locations that generate duplicate

mutants between TMI and SMI operators (this proposition will be used in Theorem 1)

Proposition 2. *Let $\mathcal{A}$ be a TAIO and $\mathcal{A}_{tmi} \in \mathcal{M}_{tmi}(TAIO)$ be a mutant from TAIO. The set of all reachable states $F = Reach(TIOTS(\mathcal{A}))$ for $\mathcal{A}$ and $F_{tmi} = Reach(TIOTS(\mathcal{A}_{tmi}))$ for $\mathcal{A}_{tmi}$. Then, $F_{tmi} \subseteq F$.*

Proof. Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ and $\mathcal{A}_{tmi} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \backslash \{t\}, I)$ over the clock valuations v and v_{tmi}, respectively. Such that $t \in T$ and $\mathcal{A}_{tmi} \in \mathcal{M}_{tmi}(\text{TAIO})$. $\mathcal{A}_{tmi}$ has two kind of transitions to reach some state. For every discrete transition (that could be silent or synchronization):

$(l, v_{tmi}) \xrightarrow{a} (l', v'_{tmi})$, for $a \in \Sigma$. The following conditions hold:

- exists a transition $(l, a, \delta, \lambda, l')$
- $v_{tmi} \models \delta$
- $v'_{tmi} = v_{tmi}[\lambda]$
- $v'_{tmi} \models I(l')$

Since, the invariant functions are the same, $(l, \delta, a, \lambda, l') \in T \setminus \{t\}$ and $T \setminus \{t\} \subseteq T$, then $(l, \delta, a, \lambda, l') \in T$. Thus, every discrete transition can be mimicked by $\mathcal{A}$. In addition, for delay transition:

$(l, v_{tmi}) \xrightarrow{d} (l, v_{tmi} + d)$, for $d \in \mathbb{R}_{\geq 0}$. The following condition hold: $v + d \models I(l)$

Since the set locations and the invariant function are the same, then every delay transition can be mimicked by $\mathcal{A}$. Here, every transition in $\mathcal{A}_{tmi}$ can be mimicked by $\mathcal{A}$, therefore every initial finite execution of $\mathcal{A}_{tmi}$ is also an initial finite execution of $\mathcal{A}$.

□

2) Proof of Theorem 1 (See page 5):

Proof. Let $\mathcal{A}$ be a TAIO

- $\mathcal{A} = (L, l_0, C, \Sigma_I, \Sigma_O, \Sigma, T, I)$,
- $\mathcal{A}_{smi} = (L \setminus \{l_{smi}\}, l_0, C, \Sigma_I, \Sigma_O, \Sigma, T_{smi}, I)$,
- $\mathcal{A}_{tmi} = (L, l_0, C, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{t_{tmi}\}, I)$, $t_{tmi} = (l, a, \delta, \lambda, l_{smi})$

And let F as the set of every possible initial finite execution fragment of $\mathcal{A}$ ending in states with the location l_{smi}. We say that $\mathcal{A}_{smi}$ is equivalent to $\mathcal{A}_{tmi}$ (duplicate) when they are bisimilar.

- First, we prove the implication that if $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$ are bisimilar, then every initial finite execution fragment of $\mathcal{A}$ ending in the location l_{smi} takes the same edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi} to simulate the execution. Suppose that $\mathcal{A}_{smi} \sim \mathcal{A}_{tmi}$, but exists at least one execution in F that does not take the edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi}. Since l_{smi} is a location in $\mathcal{A}_{tmi}$, an execution that does not take e has this form:
  $f = (l_0, v_0) \rightarrow ... \rightarrow (l', v') \xrightarrow{a'} (l_{smi}, v'')$
  As f is an initial finite execution fragment ending in the location l_{smi}, $f \in F$ (for proposition 4). Let, $e' = (l', a', \delta', \lambda', l_{smi})$ as the last edge taken in the execution f, and $e' \neq e$. Then exists and state (l', v') able to take this transition for some clock valuation v'. Now, consider the fragment of f before the last discrete transition, $f' = (l_0, v_0) \rightarrow ... \rightarrow (l', v')$. Since they are bisimilar, every transition in f' can be mimicked by $\mathcal{A}_{smi}$. However, $(l', v') \xrightarrow{a'} (l_{smi}, v'')$ in $\mathcal{A}_{tmi}$ can not be mimicked by the state of $\mathcal{A}_{smi}$, due to $\mathcal{A}_{smi}$ has to remove the transition with the target of the removed location and the state is deterministic, then there is no other state that could mimic the transition and has to be $\mathcal{A}_{smi} \not\sim \mathcal{A}_{tmi}$. And, we have reached a contradiction.
- Conversely, we have to prove that if $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$ are non-bisimilar, then not every initial finite execution fragment of $\mathcal{A}$ ending in the location l_{smi} takes the same edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi} to simulate the execution. We use proof by contrapositive. Thus, if every initial finite execution fragment of $\mathcal{A}$ ending in location l_{smi} takes the same edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi} to simulate the execution, then $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$ are bisimilar. $\mathcal{A}_{tmi}$ cannot mimic any initial finite execution of F because everyone takes the missing edge t_{tmi}. Due to the initial finite execution fragments of $\mathcal{A}_{tmi}$ ending in l_{smi} is a subset of F (for proposition 4), this also cannot have any execution that $\mathcal{A}_{tmi}$ could mimic. Thus, l_{smi} is an unreachable location for $\mathcal{A}_{tmi}$. Let S^{smi} over the clock valuations v_{smi} and S^{tmi} over the clock valuations v_{tmi} denote the states of $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$, respectively. The bisimulation for $(TIOTS(\mathcal{A}_{smi}), TIOTS(\mathcal{A}_{tmi}))$ is the binary relation $R \subseteq (S^{smi}, S^{tmi})$ such that every transition from some automata can be mimicked by the other one.
  Their initial states are the same because they both have the same initial location set.
  For every discrete transition in $\mathcal{A}_{tmi}$ (that could be silent or synchronization):
  $(l_1, v_{tmi}) \xrightarrow{a'} (l_2, v'_{tmi})$, for $a' \in \Sigma$. The following conditions hold:
  - exists a transition $(l_1, a', \delta', \lambda', l_2) \neq t_{tmi}$. $t_{tmi} = (l, \delta, a, \lambda, l_{smi})$
  - $v_{tmi} \models \delta'$
  - $v'_{tmi} = v_{tmi}[\lambda']$
  - $v'_{tmi} \models I(l_2)$

  Since l_{smi} is an unreachable location for $\mathcal{A}_{tmi}$, there is not a state with location l_{smi} for any clock valuation v_{tmi}. $l_2 \neq l_{smi}$ for every discrete transition. Also, as a consequence $l_1 \neq l_{smi}$ for every discrete transition.
  Thus, every discrete transition of $\mathcal{A}_{tmi}$ can be mimicked by $\mathcal{A}_{smi}$, due to $(l_1, a', \delta', \lambda', l_2) \in T_{smi}$; furthermore the source and target location $l_1, l_2 \in L \setminus \{l_{smi}\}$.
  Now, the discrete transitions of $\mathcal{A}_{smi}$ should be mimic by $\mathcal{A}_{tmi}$. For every discrete transition in $\mathcal{A}_{smi}$ (that could be silent or synchronization):
  $(l_1, v_{smi}) \xrightarrow{a'} (l_2, v'_{smi})$, for $a' \in \Sigma$. The following conditions hold:
  - exists a transition $(l_1, a', \delta', \lambda', l_2) \in T_{smi}$
  - $v_{smi} \models \delta'$
  - $v'_{smi} = v_{smi}[\lambda']$

– $v'_{smi} \models I(l_2)$

Since, $T_{smi} \subseteq E \setminus \{t_{tmi}\}$, every edge in $\mathcal{A}_{smi}$ is also an edge in $\mathcal{A}_{tmi}$. In addition, every location in $\mathcal{A}_{smi}$ is also a location in $\mathcal{A}_{tmi}$. As the invariant function is the same in $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$, with the same locations, then each delay transition can be mutually mimicked.

Thus, $\mathcal{A}_{tmi}$ and $\mathcal{A}_{smi}$ are bisimilar.

□

The conditions to identify when SMI and TMI operators will produce a duplicate mutant can be computationally expensive. For this reason, we will propose conditions that avoid some duplicate mutants, which are a particular case of the previous proposition. However, it does not guarantee that all created mutants would be non-duplicate. The following propositions are the reasoning that allows us to describe and avoid some duplicate mutants within the SMI and TMI operators. The following proposition describes a non-biconditional condition for finding duplicate mutants.

3) Proof of Proposition 1 (See page 5):

Proof. Suppose the set of every possible initial finite execution fragment F ending in states with the location l_{smi} of a $\mathcal{A}$ over clock X with the clock valuation function $\mathcal{V}$. Which could be an infinite set with finite sequences. Since the execution fragments are from a $\mathcal{A}$, exists at least one reachable state with the location l_{smi}, thus $F \neq \emptyset$. Moreover, every execution fragment in F has the following form, where we use Kleene star notation to denote the possible sequence: The initial state is $s_0 = (l_0, v_0)$, $l_i \in L$, $v_j \in \mathcal{V}(x)$, $d_k \in \mathbb{R}$, $a_m \in \Sigma$, for all i, j, k, m.

$$(l_0, v_0)(\xrightarrow{d_k}, \xrightarrow{a_m})((l_i, v_j) \xrightarrow{d_k} (l_1, v_j + d_k), (l_i, v_j) \xrightarrow{a_m} (l'_i, v'_j))^*(\xrightarrow{d_k}, \xrightarrow{a_m})(l_{smi}, v')$$

With this form, the initial state is followed by an action or a number, after that, when there is a number, only change the clock valuation, but if there is an action the clock valuation and location can change.

If all initial finite execution of fragment of $\mathcal{A}$ ending in location l_{smi} has the same previous location $l'_{smi} \neq l_{smi}$ and l'_{smi} only has one edge $e = (l'_{smi}, a, \delta, \lambda, l_{smi})$ to l_{smi}. Then the sub-sequence $(l'_{smi}, v_j) \xrightarrow{a_m} (l_{smi}, v'_j)$ has to be part of every sequence in F.

As consequence, removes the transition $e = (l'_{smi}, a, \delta, \lambda, l_{smi})$ makes l_{smi} unreachable to any clock valuation and every outgoing edge from l_{smi} can not be taken. Thus, removing the transition e has the same effect as removing l_smi in this case.

□

Example 3. *Consider the $\mathcal{A}$ in Figure 11. There is no need to build both mutants (TMI or SMI) because one is enough, and the other one is redundant. In order to check if removing location l_2 is equivalent to removing a transition, we are going to check the conditions of the previous proposition.*

First, consider the set F of the initial finite execution fragment ending in a state with location l_2:

$$\begin{aligned} F = \{ \quad & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0), \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{b!} (l_3, v_0) \xrightarrow{b!} (l_2, v_0), \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{b!} (l_3, v_0) \xrightarrow{b!} (l_2, v_0) \xrightarrow{c!} (l_2, v_0), \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{c!} (l_2, v_0) \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{c!} (l_2, v_0) \xrightarrow{c!} (l_2, v_0) \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{1} (l_2, v_0 + 1) \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{2} (l_2, v_0 + 2) \\ & \ldots \} \end{aligned}$$

As the system can have infinite states, and is infinitely branching, we can find infinite sets of finite executions. However, we can notice the location l_1 as a previous location in every initial execution in F for some occurrence of l_2 (underlined in the representation of set F), also l_1 has only one edge to l_2. For this reason, the SMI mutant that removes the location l_2 is duplicate to the TMI mutant that removes the transition, $(l_1, a?, true, \emptyset, l_2)$ as indicated in Proposition 1 and illustrated by the figure 11.

Next, we will propose a strategy to avoid infinite trace sets. This requires the following definition acyclic timed execution fragment

Definition 15 (Acyclic timed execution fragment)**.** *It is an initial, finite execution fragment of a TAIO. Holds the property: the location of its final state does not have a previous occurrence in the sequence.*

Example 4. *Consider the TAIO of figure 11. The following initial, finite executions are acyclic timed execution fragments:*

- $(\underline{l_0}, v_0)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (\underline{l_2}, v_0)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (l_2, v_0) \xrightarrow{3} (l_2, v_0 + 3) \xrightarrow{b!} (\underline{l_3}, v_0 + 3)$

*And the following initial, finite executions are **not acyclic timed execution fragments***

- $(\underline{l_0}, v_0) \xrightarrow{1} (\underline{l_0}, v_0 + 1)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{c!} (\underline{l_2}, v_0)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{b!} (l_3, v_0) \xrightarrow{b!} (\underline{l_2}, v_0)$

This new definition suggests how to avoid checking within an infinite set. The following lemma describes that we can equivalently check acyclic timed execution fragments for all initial, finite execution fragments.

Proposition 3. *For any $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$, every acyclic timed execution fragment has the same previous location l' iff every initial, finite execution fragment ending in a location $l \in L$ has the same previous location $l' \neq l$ for some occurrence of l.*

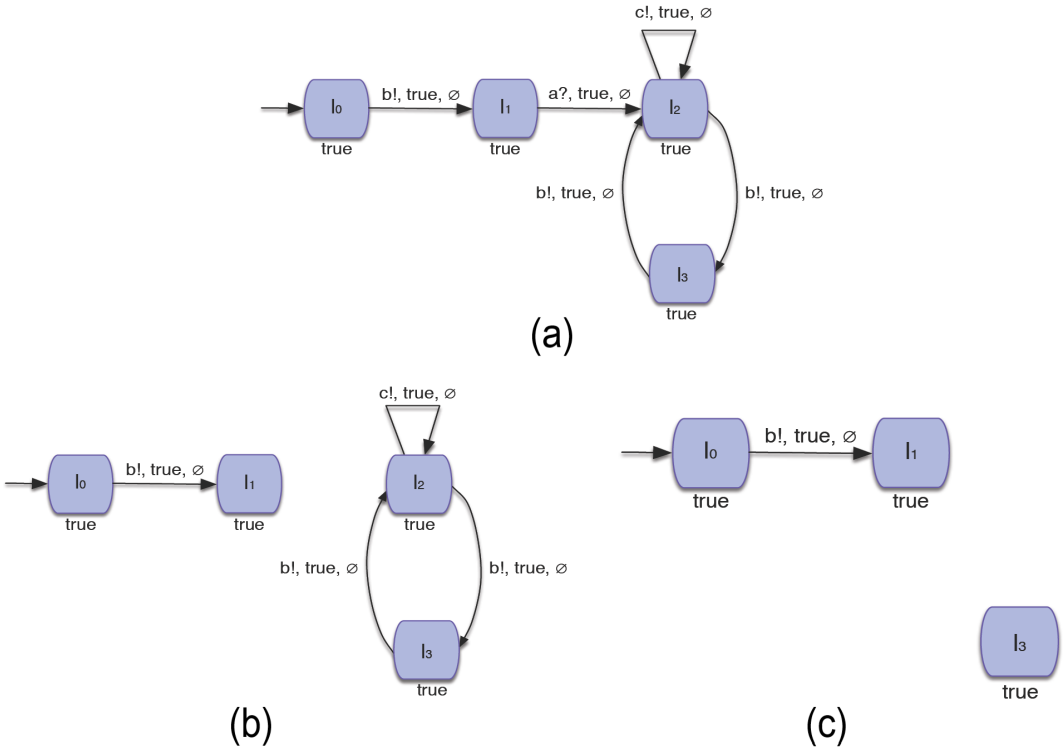

Fig. 9: A TAIO, with TMI and SMI duplicate mutants. (a) TAIO model (b) TMI mutant; (c) SMI mutant

Proof. Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ be a TAIO. And for any location $l \in L$, let F as the set of every initial finite execution fragment ending in a state with the location l, and let F_a as the set of every acyclic timed execution fragments ending in a state with the same location l.

As every acyclic timed execution fragment is also initial and finite, $F_a \subseteq F$

Every element of the set $F \setminus F_a$ with some clock valuations and more than one occurrence of l has the following form:

$f_1 = (l_0, v_0) \rightarrow ... \rightarrow (l, v') \rightarrow ... \rightarrow (l, v'')$

$f_1 \in F$, moreover, the subsequence with the first occurrence of l, $f_1' \in F_a$, for being an acyclic execution fragment. So we can write f_1 as:

$f_1 = f_1' \rightarrow ... \rightarrow (l, v'')$.

- First, we assume every acyclic timed execution fragment has the same previous location l'.
  We need to consider the execution fragments that are elements of the set $F \setminus F_a$. They have the subsequence with the form of f_1'. Since $f_1' \in F_a$, its previous is l'. And, as a consequence, l' is indeed the previous location for some occurrence of l in a sequence with the form of f_1.
  Thus, the execution fragment that is an element of F_a which does not have the same previous location l' for some occurrence of l if every acyclic timed execution fragment has the same previous location l'.
- Regarding the converse implication of the proposition, we assume that every initial, finite execution fragment ending in a location $l \in L$ has the same previous location $l' \neq l$ for some occurrence of l.
  Since $F_a \subseteq F$, F_a hold the property of F. Furthermore, as every element of F_a is an acyclic-timed execution fragment with only one occurrence of l, l' has to be the previous location.

□

Since we consider TAIO's, we are interested in their transitions. We use a directed multigraph to analyse the TAIO mutations.

Definition 16 (Directed Multigraph [18])**.** *A multigraph is a directed graph that could have more than one edge between two nodes (i.e., multiple edges). A multigraph is an ordered pair (V, E), where:*

- *V is a non-empty finite set of vertices,*
- *E is a multiset of ordered pairs of elements of V called arcs or directed edges.*

Theorem 4 (graph with duplicate mutants)**.** *Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ be a TAIO. And $G(\mathcal{A})$ be a directed multigraph from $\mathcal{A}$. If Every simple path to a node n has the same last arc and its multiplicity is equal to one, then every possible timed acyclic execution fragment of $\mathcal{A}$ ending in location n has the same previous location $n' \neq n$ for some occurrence of n and only has one edge from n' to n.*

Proof. Let $\mathcal{A}$ be a TAIO

- $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$;
- $\mathcal{A}_{smi} = (L \setminus \{l_{smi}\}, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T_{smi}, I)$;
- $\mathcal{A}_{tmi} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{t_{tmi}\}, I)$, $t_{tmi} = (l, a, \delta, \lambda, l_{smi})$;

- F as the set of every possible timed acyclic execution fragments of $\mathcal{A}$ ending in states with the location l_{smi};
- $G(\mathcal{A})$ as the Directed Multigraph from $\mathcal{A}$;
- P as the set of every simple path from l_0 to l_{smi}.

We say that $\mathcal{A}_{smi}$ is equivalent to $\mathcal{A}_{tmi}$ when they are bisimilar. By contradiction. Assume that every simple path to the node l_{smi} has the same last arc, its multiplicity is equal to one, and $\mathcal{A}_{tmi} \not\sim \mathcal{A}_{smi}$. By proposition 5, we only consider executions in which location l_{smi} is the first occurrence. Hence, by theorem 1, since they are not bisimilar, the $\mathcal{A}$ has at least two timed acyclic execution fragments ending in the location l_{smi} that takes different discrete transitions for the occurrence of l_{smi} to simulate the executions. Consider these two execution fragments are e_1 and e_2 that take a different last discrete transition with some clock valuations.

$e_1 = (l_0, v_0) \rightarrow ... \rightarrow (l_1, v_1) \rightarrow (l_{smi}, v_1')$
$e_2 = (l_0, v_0) \rightarrow ... \rightarrow (l_2, v_2) \rightarrow (l_{smi}, v_2')$

Since e_1 and e_2 can be simulated by $\mathcal{A}$, must exist the edges $(l_1, a_1, \delta_1, \lambda_1, l_{smi})$ and $(l_2, \delta_2, a_2, \lambda_2, l_{smi})$ to execute the last discrete transitions of the executions e_1 and e_2, respectively. There must be one of the following two cases:

- If $l_1 \neq l_2$. Then $G(\mathcal{A})$ has two simple paths with different last arcs (l_1, l_{smi}), and (l_2, l_{smi}). Contradicting that every simple path to the node l_{smi} has the same last arc.
- If $l_1 = l_2$. Then $m((e_1, e_2)) > 1$. Contradicting that its multiplicity is equal to one.

□

Here, we have a condition that can be checked by finding a path from the initial node to the candidate location to be removed, creating a new mutant. To do this, we create a multigraph (which is the input-timed automata without guards and invariants), and we implement Breadth-first search, which has $O(|v|+|e|)$ as time complexity in the worst-case performance [13] to check the theorem condition. Given the new SMI mutant, each mutant of the SMI set (remove location) that we avoid would be a duplicate of some mutant from the TMI set (delete transition). However, since Theorem 7 is not a biconditional proposition, we may skip some duplicates before the generation.

D. Description of mutants

A mutation operator is a function that maps a specific TAIO to a set of TAIO's. The mutants generated by our tool (for UPPAAL) follow the guidelines of the operators proposed by [6].

Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator is a function $\mathcal{M}_\mu$ that generates a set of mutants from a TAIO. We use μ to refer to each specific operator presented in [8].

Definition 17 (TMI operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator TMI ($\mu = tmi$) is a function $\mathcal{M}_{tmi}$ that removes a transition at each possible mutant TAIO.*

$$\mathcal{M}_{tmi}(\textit{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{e\}, I) \mid e \in E, a \in \Sigma_I\}$$

such that:

- $e = (l_1, a, \phi, Y, l_2) \in T$*, the removed transition is a transition of the original model for some* (l_1, a, ϕ, Y, l_2)*,*
- $a \in \Sigma_I$*, the removed transition has an input action.*

Definition 18 (TAD operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator TAD ($\mu = tad$) is a function that adds a transition at each possible mutant TAIO.*

$$\mathcal{M}_{tad}(\textit{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \cup \{e_{tad}\}, I) \mid t_{tad} \notin T, l_1, l2 \in L, a \in \Sigma_C\}$$

such that:

- $t_{tad} = (l_1, a, \phi, Y, l_2) \notin T$ *the added transition is not already a transition in* T*,*
- $a \in \Sigma_C$*, the removed transition has an output or internal (locally-controlled) action.*

Definition 19 (SMI operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator SMI ($\mu = smi$) is a function $\mathcal{M}_{smi}$ that removes a location at each possible mutant TAIO.*

$$\mathcal{M}_{smi}(\textit{TAIO}) = \{(L \setminus \{l_{smi}\}, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T', I') \mid l_{smi} \in L \setminus \{l_0\}, \exists t = (l', a, \phi, Y, l_{smi}) \in T \text{ and } a \in \Sigma_I\}$$

such that:

- $l_{smi} \in L \setminus \{l_0\}$*, the removed location is not an initial location,*
- $\exists\ e = (l', a, \phi, Y, l_{smi})$*, exists a transition* $e \in T$ *that has an input action* $a \in \Sigma_I$ *with* l_{smi} *as a target location, and some* l', ϕ, Y*,*
- $T' = \{(l_1, a, \phi, Y, l_2) \mid (l_1, a, \phi, Y, l_2) \in T, l \neq l_1, l \neq l_2\}$*, as a consequence of remove a location, the new set of edges does not have the removed location,*
- $I' : L \setminus \{l\} \rightarrow \phi(X)$*, where* $l \in L$ *from the original model. The locations that are not removed in the mutant keep the same invariant.*

Definition 20 (CXL operator)**.** *Let $\mathbb{A}$ be a set of all possibles TAIO. A mutation operator CXL ($\mu = cxl$) is a function $\mathcal{M}_{cxl}$ that increases the constant of clock guard at each possible mutant TAIO.*

$$\mathcal{M}_{cxl}(\textit{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, (T \setminus \{t\}) \cup \{t'\}, I) \mid t \in T,\ t' \notin T\}$$

such that:

- $t = (l_1, a, \phi, Y, l_2) \in T$
- $t' = (l_3, a, \phi', Y, l_4) \notin T$
- *For one of the next two cases*
  - *Output action*
    - $a \in \Sigma_O$*,*
    - $\phi = x \leq k$*, where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x) + d$ *is the clock valuation for the clock* x *and* $d > 0$*,*

* $\phi' = x \leq k + \epsilon$, *for* $\epsilon > 0$, *where exists at least a clock* $k < x' \leq k + \epsilon$, k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$, *such that* $\nu \models I(l_3)$, $\nu' = \nu[Y \to 0]$ *and* $\nu' \models I(l_4)$.

– *or*

– *Input action*

* $a \in \Sigma_I$;
* $\phi = x \geq k$, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$,
* $\phi' = x \geq k + \epsilon$, *for* $\epsilon > 0$, *where exists at least a clock* $k < x' \leq k + \epsilon$, k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$, *such that* $\nu \models I(l_3)$, $\nu' = \nu[Y \to 0]$ *and* $\nu' \models I(l_4)$.

Definition 21 (CXS operator)**.** *Let* $\mathbb{A}$ *be a set of all possibles* TAIO. *A mutation operator* CXS ($\mu = cxs$) *is a function* $\mathcal{M}_{cxs}$ *that decreases the constant of clock guard at each possible mutant* TAIO.

$$\mathcal{M}_{cxs}(\textsf{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, (T \setminus \{t\}) \cup \{t'\}, I) \mid t \in T,\ t' \notin T\}$$

such that:

- $t = (l_1, a, \phi, Y, l_2) \in T$
- $t' = (l_3, a, \phi', Y, l_4) \notin T$

For one of the next two cases

– *Output action*

* $a \in \Sigma_O$;
* $\phi = x \geq k$, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$,
* $\phi' = x \geq k - \epsilon$, *for* $\epsilon > 0$, *where exists at least a clock valuation* $k - \epsilon \leq x' < k$, k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$, *such that* $\nu \models I(l_3)$, $\nu' = \nu[Y \to 0]$ *and* $\nu' \models I(l_4)$, *and exists an initial finite run ending in the state* (l_3, ν').

or

– *Input action*

* $a \in \Sigma_I$;
* $\phi = x \leq k$, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$,
* $\phi' = x \leq k - \epsilon$, *for* $\epsilon > 0$, *where exists at least a clock valuation* $k - \epsilon < x' \leq k$ *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$.

Definition 22 (CCN operator)**.** *Let* $\mathbb{A}$ *be a set of all possibles* TAIO. *A mutation operator* CCN ($\mu = ccn$) *is a function* $\mathcal{M}_{ccn}$*that negates a clock guard at each possible mutant* TAIO.

$$\mathcal{M}_{ccn}(\textsf{TAIO}) = \{(L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, (T \setminus \{t\}) \cup \{t'\}, I, F) \mid t \in T,\ t' \notin T\}$$

such that:

- $t = (l_1, a, \phi, Y, l_2) \in T$
- $t' = (l_3, a, \phi', Y, l_4) \notin T$
- *For one of the next four cases according to the form of* ϕ, *where* $x \in X$ *is a clock,* k *is a constant,* $\nu(x)$ + d *is the clock valuation for the clock* x *and* $d > 0$.

  – *if* $\phi = x \leq k$, *then* $\phi' = x > k$,
  – *if* $\phi = x < k$, *then* $\phi' = x \geq k$,
  – *if* $\phi = x \geq k$, *then* $\phi' = x < k$,
  – *if* $\phi = x > k$, *then* $\phi' = x \leq k$.

E. Equivalent mutants

In 2010 [34] published a work that introduce the definition of *refinement* and the implementation of the theory in the tool UPPAAL TIGA. Here the notion of *refinement* allows comparing two specifications as well as relating an implementation to a specification. The refinement is a binary operator between TAIO and should satisfy the following condition. S_1 is a refinement of S_2 if is possible to replace S_2 with S_1 in every environment and obtain an equivalent system. Formally

Definition 23 (Refinement)**.** *Let* $\mathcal{A}_1 = (L^1, l_0^1, X^1, \Sigma_I^1, \Sigma_O^1, \Sigma^1, T^1, I^1)$ *and* $\mathcal{A}_2 = (L^2, l_0^2, X^2, \Sigma_I^2, \Sigma_O^2, \Sigma^2, T^2, I^2)$ *be* TAIO *with the states* S^1 *and* S^2, *respectively. Let* $v^1 : C^1 \to \mathbb{R}_{\geq 0}$, $v^2 : C^2 \to \mathbb{R}_{\geq 0}$ *as clock valuation functions, and* s_0^1, s_0^2 *as initial states of* $\mathcal{A}_1$, $\mathcal{A}_2$, *respectively.* $\mathcal{A}_1$ *is a refinement of* $\mathcal{A}_2$, *written* $\mathcal{A}_1 \leq \mathcal{A}_2$ *iff there exists a binary relation* $\mathcal{R} \subseteq (S^1 \times S^2)$ *containing* $s = (s_0^1, s_0^2)$ *such that for each pair of states* $(s^1, s^2) = ((l^1, v^1), (l^2, v^2)) \in R$, *it holds:*

1) ***whenever*** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^{2'})$ *for some* $l^{2'}$ *and* $a \in \Sigma_I^2$, ***then*** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^1)$ *for some* $l^{1'}$ *and* $((l^{1'}, v^{1'}), (l^{2'}, v^{2'})) \in \mathcal{R}$
2) ***whenever*** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^{1'})$ *for some* $l^{1'}$ *and* $a \in \Sigma_C^2$, ***then*** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^{2'})$ *for some* $l^{2'}$ *and* $((l^{1'}, v^1), (l^{2'}, v^2)) \in \mathcal{R}$
3) ***whenever*** $(l^1, v^1) \xrightarrow{d} (l^1, v^{1'})$ *for some* $v^{1'}$ *and* $d \in \mathbb{R}_{\geq 0}$, ***then*** $(l^2, v^2) \xrightarrow{d} (l^2, v^{2'})$ *for some* $v^{2'}$ *and* $((l^1, v^{1'}), (l^2, v^{2'})) \in \mathcal{R}$

[34] implemented the theory of refinement equivalence between TAIO in UPPAAL TIGA In the same year [33] published a tool called ECDAR, as an extension of UPPAAL-TIGA it reuses some of its components, including the refinement operation. In 2017 [33] published a work where they collect the mutation operators proposed [3] (for UPPAAL) and used ECDAR to check the conformance between the System Under Test and the mutants with the refinement operation.

Theorem 5 (Bisimulation is Finer than Refinement Equivalence)**.**
$\textsf{TAIO}_1 \sim \textsf{TAIO}_2$ *implies* $\textsf{TAIO}_1 \leq \textsf{TAIO}_2$.

Proof. Suppose that the theorem is false. Then there must be two TAIO for which the theorem fails. Let $\mathcal{A}_i$ =

$(L^i, l_0^i, X^i, \Sigma_I^i, \Sigma_O^i, \Sigma^i, T^i, I^i)$, $i = 1, 2$, be two TAIO, for sake of contradiction, that $\mathcal{A}_1$ and $\mathcal{TIOA}_2$ are bisimilar and $\mathcal{A}_1$ does not refine $\mathcal{A}_2$. In this case, let S_1 and S_2 be any set of states of $\mathcal{A}_1$ and $\mathcal{TA}_2$, respectively, and $R \subseteq S_1 \times S_2$. Since $\mathcal{A}_1$ does not refine $\mathcal{A}_2$, at least one of the following conditions must be satisfied (the negation of the conditions to be a refinement):

1) $(s_0^1, s_0^2) \notin \mathcal{R}$. For all initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$, where s_0^1 is an initial state of $\mathcal{A}_1$ and s_0^2 is an initial state of $\mathcal{A}_2$
2) **exists** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^2)$ for some $l^{2'}$ and $a \in \Sigma_I^2$, **then** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^1) \notin T^1$ for any $l^{1'}$ thus $((l^{1'}, v^1), (l^{2'}, v^2)) \notin \mathcal{R}$
3) **exists** $(l^1, v^1) \xrightarrow{a} (l^{1'}, v^1)$ for some $l^{1'}$ and $a \in \Sigma_C^1$, **then** $(l^2, v^2) \xrightarrow{a} (l^{2'}, v^2) \notin T^2$ for any $l^{1'}$ thus $((l^{1'}, v^1), (l^{2'}, v^2)) \notin \mathcal{R}$
4) **exists** $(l^1, v^1) \xrightarrow{d} (l^1, v^{1'})$ for some $v^{1'}$ and $d \in \mathbb{R}_{\geq 0}$, **then** $(l^2, v^2) \xrightarrow{d} (l^2, v^{2'}) \notin E^2$ for any $v^{2'}$ thus $((l^1, v^{1'}), (l^2, v^{2'})) \notin \mathcal{R}$

However, since $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar, $\forall s_1 \in l_0^1 (\exists s_2 \in l_0^2 . (s_1, s_2) \in \mathcal{R}$ and $\forall s_2 \in l_0^2 (\exists s_1 \in l_0^1 . (s_1, s_2) \in \mathcal{R})$. As l_0^1 and l_0^2 are non-empty sets, then at least one pair of initial states (s_0^1, s_0^2) of $\mathcal{A}_1$ and $\mathcal{A}_2$ must be in $\mathcal{R}$ for any automata. Thus, the first (1) condition does not hold. Since $\mathcal{A}_1 \sim \mathcal{A}_2$, by definition, for all, $(s_1, s_2) \in \mathcal{R}$ it holds that if exists $s_2 \xrightarrow{x} s_2' \in T^2$ for any x then exists $s_1 \xrightarrow{x} s_1' \in T^1$. Thus, if $x \in \Sigma_I^2$, then the condition holds and exists $s_1 \xrightarrow{x} s_1' \in T^1$, and the second (2) condition does not hold. Since $\mathcal{A}_1 \sim \mathcal{A}_2$, by definition, for all, $(s_1, s_2) \in \mathcal{R}$ it holds that if exists $s_1 \xrightarrow{x} s_1' \in T^1$ for any x then exists $s_2 \xrightarrow{x} s_2' \in T^2$. Thus, if $x \in \Sigma_C^2$, then the condition holds and exists $s_2 \xrightarrow{x} s_2' \in T^2$ and the third (3) condition does not hold. And if $x \in \mathbb{R}_{\geq 0}$, then the condition holds and exists $s_2 \xrightarrow{x} s_2' \in T^2$ and the fourth (4) condition does not hold. Thus, is impossible: $\mathcal{A}_1 \sim \mathcal{A}_2$ and $\mathcal{A}_1 \not\leq \mathcal{A}_2$. Therefore, it must be the case of our assumption that $\mathcal{A}_1 \sim \mathcal{A}_2$ implies $\mathcal{A}_1 \leq \mathcal{A}_2$.

□

Theorem 6 (Non-bisimulation and Refinement Equivalence)**.** *For two any TAIO $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1 \leq \mathcal{A}_2$ does not always imply $\mathcal{A}_1 \sim \mathcal{A}_2$.*

Proof. The fact that $\mathcal{A}_1 \leq \mathcal{A}_2$ does not always imply $\mathcal{A}_1 \sim \mathcal{A}_2$ is illustrated by the following example.

Consider the $\mathcal{A}_1 = (L^1, l_0^1, X^1, \Sigma_I^1, \Sigma_O^1, \Sigma^1, T^1, I^1)$, where:

- $L^1 = \{l_0^1, l_1^1\}$,
- $l_0^1 = l_0$,
- $l_1^1 = l_1$,
- $X^1 = \emptyset$,
- $\Sigma_I^1 = \{b?, c?\}$,
- $\Sigma_O^1 = \{a!\}$,
- $\Sigma^1 = \Sigma_I^1 \cup \Sigma_O^1$
- $T^1 = \{(l_0, a!, \emptyset, true, l_0), (l_0, b?, \emptyset, true, l_0), (l_0, c?, \emptyset, true, l_1^1)\}$,
- $I^1(l^1) = true$.

And consider the $\mathcal{A}_2 = (L^2, l_0^2, X^2, \Sigma_I^2, \Sigma_O^2, \Sigma^2, T^2, I^2)$, where:

- $L^2 = \{l_0^2, l_1^2\}$,
- $m_0 = l_0^2$,
- $m_1 = l_1^2$,
- $X^2 = \emptyset$,
- $\Sigma_I^2 = \{b?\}$,
- $\Sigma_O^2 = \{a!, d!\}$,
- $\Sigma^2 = \Sigma_I^2 \cup \Sigma_O^2$
- $T^2 = \{(m_0, a!, \emptyset, true, m_0), (m_0, b?, \emptyset, true, m_0), (m_0, d!, \emptyset, true, m_1)\}$,
- $I^2(m_1) = true$.

Graphically, the $\mathcal{A}_1$ (left) and $\mathcal{A}_2$ (right) in Figure 10:

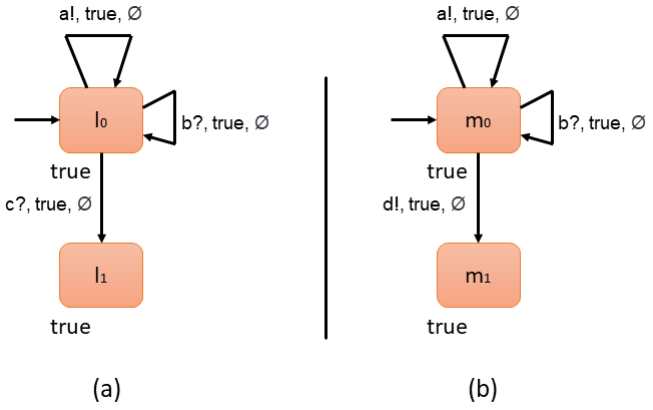

Fig. 10: Refinement equivalent, but not bisimulation equivalent TIOA's. (a) $\mathcal{A}_1$, (b) $\mathcal{A}_2$

As there is no bisimilar state in $\mathcal{A}_1$ that mimics (m_0, v^2) for any v^2, the only candidate would be the state (l_0, v^1) for some v^1 but (l_0, v^1) can not mimic $(m_0, v^2) \xrightarrow{d!} (m_1, v^2)$ since there is no transition in T^1 with action $d!$ and $d! \notin \Sigma_O^1$. Similarly, there is no bisimilar state in $\mathcal{A}_2$ that mimics (l_0, v^1) for any v^1, here the only candidate would be the state (m_0, v^2) for some v^2 but (m_0, v^2) can not mimic $(l_0, v^1) \xrightarrow{c?} (l_1, v^1)$ since there is no transition in T^2 with action $c?$ and $c? \notin \Sigma_I^2$. However, $\mathcal{A}_1$ and $\mathcal{A}_2$ are refinement-equivalent, as every delay and output action in $\mathcal{A}_1$ can be mimicked by $\mathcal{A}_2$, and every input action in $\mathcal{A}_2$ can be mimicked by $\mathcal{A}_1$.

Thus, $\mathcal{A}_1 \not\sim \mathcal{A}_2$ and $\mathcal{A}_1 \leq \mathcal{A}_2$ is possible.

□

Two immediate corollaries to Theorem 5, and Theorem 6 are:

Corollary 3 (Bisimulation is Strictly Finer than Refinement Equivalence)**.** *$\mathcal{A}_1 \sim \mathcal{A}_2$implies $\mathcal{A}_1 \leq \mathcal{A}_2$, but $\mathcal{A}_1 \not\sim \mathcal{A}_2$ and $\mathcal{A}_1 \leq \mathcal{A}_2$ is possible.*

Corollary 4 (Non-bisimulation implies Non-refinement equivalence)**.** *$\mathcal{A}_1 \not\leq \mathcal{A}_2$ implies $\mathcal{A}_1 \not\sim \mathcal{A}_2$.*

F. Duplicate mutants(TMI and SMI operators)

In this section, we will consider the mutants generated by TMI and SMI operators that are duplicates. We define these operators in D section.

1) When a Timed automaton with Input and Output produces a SMI mutant duplicate to TMI mutant duplicate ?: Since we are considering TAIO, and every mutant of the set $\mathcal{M}_{smi}(\mathsf{TAIO})$ is not a refinement of TAIO, and as a consequence, they are not bisimilar. Indeed, there is no state in the mutants $\mathcal{M}_{smi}(\mathsf{TAIO})$ that can mimic their respective missing state. There is possible to find mutants in the set $\mathcal{M}_{tmi}(\mathsf{TAIO})$ where the missing transition makes some state unreachable, generating a mutant bisimilar to some other mutant of the set $\mathcal{M}_{smi}(\mathsf{TAIO})$. The following propositions describe the condition of the locations that generate duplicate mutants between TMI and SMI operators (this proposition will be used in Theorem 1)

Proposition 4. *Let $\mathcal{A}$ be a TAIO and $\mathcal{A}_{tmi} \in \mathcal{M}_{tmi}(TAIO)$ be a mutant from TAIO. The set of all reachable states $F = Reach(TIOTS(\mathcal{A}))$ for $\mathcal{A}$ and $F_{tmi} = Reach(TIOTS(\mathcal{A}_{tmi}))$ for $\mathcal{A}_{tmi}$. Then, $F_{tmi} \subseteq F$.*

Proof. Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ and $\mathcal{A}_{tmi} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T\backslash\{t\}, I)$ over the clock valuations v and v_{tmi}, respectively. Such that $t \in T$ and $\mathcal{A}_{tmi} \in \mathcal{M}_{tmi}(\mathsf{TAIO})$. $\mathcal{A}_{tmi}$ has two kind of transitions to reach some state. For every discrete transition (that could be silent or synchronization):

$(l, v_{tmi}) \xrightarrow{a} (l', v'_{tmi})$, for $a \in \Sigma$. The following conditions hold:

- exists a transition $(l, a, \delta, \lambda, l')$
- $v_{tmi} \models \delta$
- $v'_{tmi} = v_{tmi}[\lambda]$
- $v'_{tmi} \models I(l')$

Since, the invariant functions are the same, $(l, \delta, a, \lambda, l') \in T \setminus \{t\}$ and $T \setminus \{t\} \subseteq T$, then $(l, \delta, a, \lambda, l') \in T$. Thus, every discrete transition can be mimicked by $\mathcal{A}$. In addition, for delay transition:

$(l, v_{tmi}) \xrightarrow{d} (l, v_{tmi} + d)$, for $d \in \mathbb{R}_{\geq 0}$. The following condition hold: $v + d \models I(l)$

Since the set locations and the invariant function are the same, then every delay transition can be mimicked by $\mathcal{A}$. Here, every transition in $\mathcal{A}_{tmi}$ can be mimicked by $\mathcal{A}$, therefore every initial finite execution of $\mathcal{A}_{tmi}$ is also an initial finite execution of $\mathcal{A}$.

□

2) Proof of Theorem 1 (See page 5):

Proof. Let $\mathcal{A}$ be a TAIO

- $\mathcal{A} = (L, l_0, C, \Sigma_I, \Sigma_O, \Sigma, T, I)$,
- $\mathcal{A}_{smi} = (L \setminus \{l_{smi}\}, l_0, C, \Sigma_I, \Sigma_O, \Sigma, T_{smi}, I)$,
- $\mathcal{A}_{tmi} = (L, l_0, C, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{t_{tmi}\}, I)$, $t_{tmi} = (l, a, \delta, \lambda, l_{smi})$

And let F as the set of every possible initial finite execution fragment of $\mathcal{A}$ ending in states with the location l_{smi}. We say that $\mathcal{A}_{smi}$ is equivalent to $\mathcal{A}_{tmi}$ (duplicate) when they are bisimilar.

- First, we prove the implication that if $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$ are bisimilar, then every initial finite execution fragment of $\mathcal{A}$ ending in the location l_{smi} takes the same edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi} to simulate the execution. Suppose that $\mathcal{A}_{smi} \sim \mathcal{A}_{tmi}$, but exists at least one execution in F that does not take the edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi}. Since l_{smi} is a location in $\mathcal{A}_{tmi}$, an execution that does not take e has this form:
  $f = (l_0, v_0) \rightarrow ... \rightarrow (l', v') \xrightarrow{a'} (l_{smi}, v'')$
  As f is an initial finite execution fragment ending in the location l_{smi}, $f \in F$ (for proposition 4). Let, $e' = (l', a', \delta', \lambda', l_{smi})$ as the last edge taken in the execution f, and $e' \neq e$. Then exists and state (l', v') able to take this transition for some clock valuation v'. Now, consider the fragment of f before the last discrete transition, $f' = (l_0, v_0) \rightarrow ... \rightarrow (l', v')$. Since they are bisimilar, every transition in f' can be mimicked by $\mathcal{A}_{smi}$. However, $(l', v') \xrightarrow{a'} (l_{smi}, v'')$ in $\mathcal{A}_{tmi}$ can not be mimicked by the state of $\mathcal{A}_{smi}$, due to $\mathcal{A}_{smi}$ has to remove the transition with the target of the removed location and the state is deterministic, then there is no other state that could mimic the transition and has to be $\mathcal{A}_{smi} \not\sim \mathcal{A}_{tmi}$. And, we have reached a contradiction.
- Conversely, we have to prove that if $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$ are non-bisimilar, then not every initial finite execution fragment of $\mathcal{A}$ ending in the location l_{smi} takes the same edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi} to simulate the execution. We use proof by contrapositive. Thus, if every initial finite execution fragment of $\mathcal{A}$ ending in location l_{smi} takes the same edge $e = (l, a, \delta, \lambda, l_{smi})$ for some occurrence of l_{smi} to simulate the execution, then $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$ are bisimilar. $\mathcal{A}_{tmi}$ cannot mimic any initial finite execution of F because everyone takes the missing edge t_{tmi}. Due to the initial finite execution fragments of $\mathcal{A}_{tmi}$ ending in l_{smi} is a subset of F (for proposition 4), this also cannot have any execution that $\mathcal{A}_{tmi}$ could mimic. Thus, l_{smi} is an unreachable location for $\mathcal{A}_{tmi}$. Let S^{smi} over the clock valuations v_{smi} and S^{tmi} over the clock valuations v_{tmi} denote the states of $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$, respectively. The bisimulation for $(TIOTS(\mathcal{A}_{smi}), TIOTS(\mathcal{A}_{tmi}))$ is the binary relation $R \subseteq (S^{smi}, S^{tmi})$ such that every transition from some automata can be mimicked by the other one.
  Their initial states are the same because they both have the same initial location set.
  For every discrete transition in $\mathcal{A}_{tmi}$ (that could be silent or synchronization):
  $(l_1, v_{tmi}) \xrightarrow{a'} (l_2, v'_{tmi})$, for $a' \in \Sigma$. The following conditions hold:
  - exists a transition $(l_1, a', \delta', \lambda', l_2) \neq t_{tmi}$. $t_{tmi} = (l, \delta, a, \lambda, l_{smi})$
  - $v_{tmi} \models \delta'$
  - $v'_{tmi} = v_{tmi}[\lambda']$

– $v'_{tmi} \models I(l_2)$

Since l_{smi} is an unreachable location for $\mathcal{A}_{tmi}$, there is not a state with location l_{smi} for any clock valuation v_{tmi}. $l_2 \neq l_{smi}$ for every discrete transition. Also, as a consequence $l_1 \neq l_{smi}$ for every discrete transition.

Thus, every discrete transition of $\mathcal{A}_{tmi}$ can be mimicked by $\mathcal{A}_{smi}$, due to $(l_1, a', \delta', \lambda', l_2) \in T_{smi}$; furthermore the source and target location $l_1, l_2 \in L \setminus \{l_{smi}\}$.

Now, the discrete transitions of $\mathcal{A}_{smi}$ should be mimic by $\mathcal{A}_{tmi}$. For every discrete transition in $\mathcal{A}_{smi}$ (that could be silent or synchronization):

$(l_1, v_{smi}) \xrightarrow{a'} (l_2, v'_{smi})$, for $a' \in \Sigma$. The following conditions hold:

– exists a transition $(l_1, a', \delta', \lambda', l_2) \in T_{smi}$
– $v_{smi} \models \delta'$
– $v'_{smi} = v_{smi}[\lambda']$
– $v'_{smi} \models I(l_2)$

Since, $T_{smi} \subseteq E \setminus \{t_{tmi}\}$, every edge in $\mathcal{A}_{smi}$ is also an edge in $\mathcal{A}_{tmi}$. In addition, every location in $\mathcal{A}_{smi}$ is also a location in $\mathcal{A}_{tmi}$. As the invariant function is the same in $\mathcal{A}_{smi}$ and $\mathcal{A}_{tmi}$, with the same locations, then each delay transition can be mutually mimicked.

Thus, $\mathcal{A}_{tmi}$ and $\mathcal{A}_{smi}$ are bisimilar.

□

The conditions to identify when SMI and TMI operators will produce a duplicate mutant can be computationally expensive. For this reason, we will propose conditions that avoid some duplicate mutants, which are a particular case of the previous proposition. However, it does not guarantee that all created mutants would be non-duplicate. The following propositions are the reasoning that allows us to describe and avoid some duplicate mutants within the SMI and TMI operators. The following proposition describes a non-biconditional condition for finding duplicate mutants.

3) Proof of Proposition 1 (See page 5):

Proof. Suppose the set of every possible initial finite execution fragment F ending in states with the location l_{smi} of a $\mathcal{A}$ over clock X with the clock valuation function $\mathcal{V}$. Which could be an infinite set with finite sequences. Since the execution fragments are from a $\mathcal{A}$, exists at least one reachable state with the location l_{smi}, thus $F \neq \emptyset$. Moreover, every execution fragment in F has the following form, where we use Kleene star notation to denote the possible sequence: The initial state is $s_0 = (l_0, v_0)$, $l_i \in L$, $v_j \in \mathcal{V}(x)$, $d_k \in \mathbb{R}$, $a_m \in \Sigma$, for all i, j, k, m.

$$(l_0, v_0)(\xrightarrow{d_k}, \xrightarrow{a_m})((l_i, v_j) \xrightarrow{d_k} (l_1, v_j + d_k), (l_i, v_j) \xrightarrow{a_m} (l'_i, v'_j))^*(\xrightarrow{d_k}, \xrightarrow{a_m})(l_{smi}, v')$$

With this form, the initial state is followed by an action or a number, after that, when there is a number, only change the clock valuation, but if there is an action the clock valuation and location can change.

If all initial finite execution of fragment of $\mathcal{A}$ ending in location l_{smi} has the same previous location $l'_{smi} \neq l_{smi}$ and l'_{smi} only has one edge $e = (l'_{smi}, a, \delta, \lambda, l_{smi})$ to l_{smi}. Then the sub-sequence $(l'_{smi}, v_j) \xrightarrow{a_m} (l_{smi}, v'_j)$ has to be part of every sequence in F.

As consequence, removes the transition $e = (l'_{smi}, a, \delta, \lambda, l_{smi})$ makes l_{smi} unreachable to any clock valuation and every outgoing edge from l_{smi} can not be taken. Thus, removing the transition e has the same effect as removing l_smi in this case.

□

Example 5. *Consider the $\mathcal{A}$ in Figure 11. There is no need to build both mutants (TMI or SMI) because one is enough, and the other one is redundant. In order to check if removing location l_2 is equivalent to removing a transition, we are going to check the conditions of the previous proposition.*

First, consider the set F of the initial finite execution fragment ending in a state with location l_2:

$$\begin{aligned} F = \{ \quad & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0), \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{b!} (l_3, v_0) \\ & \xrightarrow{b!} (l_2, v_0), \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{b!} (l_3, v_0) \\ & \xrightarrow{b!} (l_2, v_0) \xrightarrow{c!} (l_2, v_0), \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{c!} (l_2, v_0) \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{c!} (l_2, v_0) \\ & \xrightarrow{c!} (l_2, v_0) \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{1} (l_2, v_0 + 1) \\ & (l_0, v_0) \xrightarrow{b!} (\underline{l_1}, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{2} (l_2, v_0 + 2) \\ & \ldots \} \end{aligned}$$

As the system can have infinite states, and is infinitely branching, we can find infinite sets of finite executions. However, we can notice the location l_1 as a previous location in every initial execution in F for some occurrence of l_2 (underlined in the representation of set F), also l_1 has only one edge to l_2. For this reason, the SMI mutant that removes the location l_2 is duplicate to the TMI mutant that removes the transition, $(l_1, a?, true, \emptyset, l_2)$ as indicated in Proposition 1 and illustrated by the figure 11.

Next, we will propose a strategy to avoid infinite trace sets. This requires the following definition acyclic timed execution fragment

Definition 24 (Acyclic timed execution fragment)**.** *It is an initial, finite execution fragment of a TAIO. Holds the property: the location of its final state does not have a previous occurrence in the sequence.*

Example 6. *Consider the TAIO of figure 11. The following initial, finite executions are acyclic timed execution fragments:*

- $(\underline{l_0}, v_0)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (\underline{l_2}, v_0)$

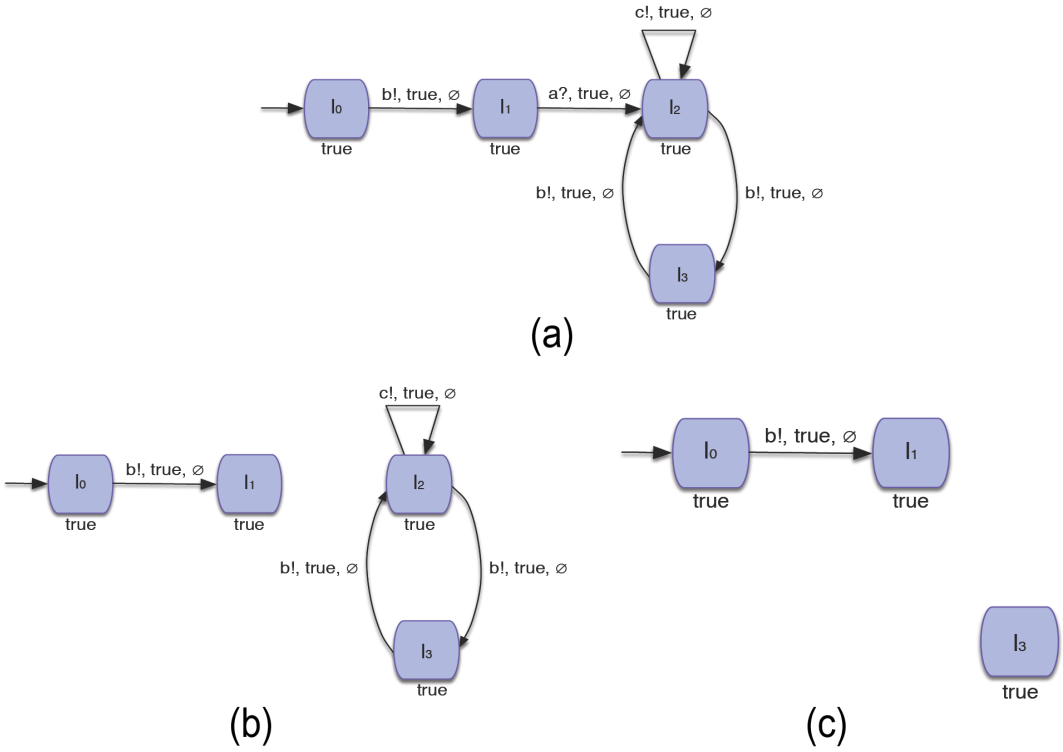

Fig. 11: A TAIO, with TMI and SMI duplicate mutants. (a) TAIO model (b) TMI mutant; (c) SMI mutant

- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (l_2, v_0) \xrightarrow{3} (l_2, v_0 + 3) \xrightarrow{b!} (\underline{l_3}, v_0 + 3)$

And the following initial, finite executions are ***not acyclic timed execution fragments***

- $(\underline{l_0}, v_0) \xrightarrow{1} (\underline{l_0}, v_0 + 1)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{c!} (\underline{l_2}, v_0)$
- $(l_0, v_0) \xrightarrow{b!} (l_1, v_0) \xrightarrow{a?} (\underline{l_2}, v_0) \xrightarrow{b!} (l_3, v_0) \xrightarrow{b!} (\underline{l_2}, v_0)$

This new definition suggests how to avoid checking within an infinite set. The following lemma describes that we can equivalently check acyclic timed execution fragments for all initial, finite execution fragments.

Proposition 5. *For any $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$, every acyclic timed execution fragment has the same previous location l' iff every initial, finite execution fragment ending in a location $l \in L$ has the same previous location $l' \neq l$ for some occurrence of l.*

Proof. Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ be a TAIO. And for any location $l \in L$, let F as the set of every initial finite execution fragment ending in a state with the location l, and let F_a as the set of every acyclic timed execution fragments ending in a state with the same location l.

As every acyclic timed execution fragment is also initial and finite, $F_a \subseteq F$

Every element of the set $F \setminus F_a$ with some clock valuations and more than one occurrence of l has the following form:

$f_1 = (l_0, v_0) \rightarrow ... \rightarrow (l, v') \rightarrow ... \rightarrow (l, v'')$

$f_1 \in F$, moreover, the subsequence with the first occurrence of l, $f_1' \in F_a$, for being an acyclic execution fragment. So we can write f_1 as:

$f_1 = f_1' \rightarrow ... \rightarrow (l, v'')$.

- First, we assume every acyclic timed execution fragment has the same previous location l'.
  We need to consider the execution fragments that are elements of the set $F \setminus F_a$. They have the subsequence with the form of f_1'. Since $f_1' \in F_a$, its previous is l'. And, as a consequence, l' is indeed the previous location for some occurrence of l in a sequence with the form of f_1.
  Thus, the execution fragment that is an element of F_a which does not have the same previous location l' for some occurrence of l if every acyclic timed execution fragment has the same previous location l'.
- Regarding the converse implication of the proposition, we assume that every initial, finite execution fragment ending in a location $l \in L$ has the same previous location $l' \neq l$ for some occurrence of l.
  Since $F_a \subseteq F$, F_a hold the property of F. Furthermore, as every element of F_a is an acyclic-timed execution fragment with only one occurrence of l, l' has to be the previous location.

□

Since we consider TAIO's, we are interested in their transitions. We use a directed multigraph to analyse the TAIO mutations.

Definition 25 (Directed Multigraph [18])**.** *A multigraph is a directed graph that could have more than one edge between two nodes (i.e., multiple edges). A multigraph is an ordered pair (V, E), where:*

- *V is a non-empty finite set of vertices,*
- *E is a multiset of ordered pairs of elements of V called arcs or directed edges.*

Theorem 7 (graph with duplicate mutants)**.** *Let $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$ be a TAIO. And $G(\mathcal{A})$ be a directed multigraph from $\mathcal{A}$. If Every simple path to a node n has the same last arc and its multiplicity is equal to one, then every possible timed acyclic execution fragment of $\mathcal{A}$ ending in location n has the same previous location $n' \neq n$ for some occurrence of n and only has one edge from n' to n.*

Proof. Let $\mathcal{A}$ be a TAIO

- $\mathcal{A} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T, I)$;
- $\mathcal{A}_{smi} = (L \setminus \{l_{smi}\}, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T_{smi}, I)$;
- $\mathcal{A}_{tmi} = (L, l_0, X, \Sigma_I, \Sigma_O, \Sigma, T \setminus \{t_{tmi}\}, I)$, $t_{tmi} = (l, a, \delta, \lambda, l_{smi})$;
- F as the set of every possible timed acyclic execution fragments of $\mathcal{A}$ ending in states with the location l_{smi};
- $G(\mathcal{A})$ as the Directed Multigraph from $\mathcal{A}$;
- P as the set of every simple path from l_0 to l_{smi}.

Here, we have a condition that can be checked by finding a path from the initial node to the candidate location to be removed, creating a new mutant. To do this, we create a multigraph (which is the input-timed automata without guards and invariants), and we implement Breadth-first search, which has $O(|v|+|e|)$ as time complexity in the worst-case performance [13] to check the theorem condition. Given the new SMI mutant, each mutant of the SMI set (remove location) that we avoid would be a duplicate of some mutant from the TMI set (delete transition). However, since Theorem 7 is not a biconditional proposition, we may skip some duplicates before the generation.

We say that $\mathcal{A}_{smi}$ is equivalent to $\mathcal{A}_{tmi}$ when they are bisimilar. By contradiction. Assume that every simple path to the node l_{smi} has the same last arc, its multiplicity is equal to one, and $\mathcal{A}_{tmi} \not\sim \mathcal{A}_{smi}$. By proposition 5, we only consider executions in which location l_{smi} is the first occurrence. Hence, by theorem 1, since they are not bisimilar, the $\mathcal{A}$ has at least two timed acyclic execution fragments ending in the location l_{smi} that takes different discrete transitions for the occurrence of l_{smi} to simulate the executions. Consider these two execution fragments are e_1 and e_2 that take a different last discrete transition with some clock valuations.

$e_1 = (l_0, v_0) \rightarrow ... \rightarrow (l_1, v_1) \rightarrow (l_{smi}, v'_1)$
$e_2 = (l_0, v_0) \rightarrow ... \rightarrow (l_2, v_2) \rightarrow (l_{smi}, v'_2)$

Since e_1 and e_2 can be simulated by $\mathcal{A}$, must exist the edges $(l_1, a_1, \delta_1, \lambda_1, l_{smi})$ and $(l_2, \delta_2, a_2, \lambda_2, l_{smi})$ to execute the last discrete transitions of the executions e_1 and e_2, respectively. There must be one of the following two cases:

- If $l_1 \neq l_2$. Then $G(\mathcal{A})$ has two simple paths with different last arcs (l_1, l_{smi}), and (l_2, l_{smi}). Contradicting that every simple path to the node l_{smi} has the same last arc.
- If $l_1 = l_2$. Then $m((e_1, e_2)) > 1$. Contradicting that its multiplicity is equal to one.

□