Tackling the Equivalent Mutant Problem in Real-Time Systems:
The 12 Commandments of Model-Based Mutation Testing

Davide Basile
ISTI–CNR, Pisa, Italy
davide.basile@isti.cnr.it

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

Maurice H. ter Beek
ISTI–CNR, Pisa, Italy
maurice.terbeek@isti.cnr.it

Axel Legay
UCLouvain, Belgium
axel.legay@uclouvain.be

ABSTRACT
Mutation testing can effectively drive test generation to reveal faults in software systems. However, it faces a typical efficiency issue as it can produce many mutants that are equivalent to the original system, making it impossible to generate test cases from them. We consider this problem when model-based mutation testing is applied to real-time system product lines, represented as timed automata. We define novel, time-specific mutation operators and formulate the equivalent mutant problem in the frame of timed refinement relations.

Further, we study in which cases a mutation yields an equivalent mutant. Our theoretical results provide guidance to system engineers, allowing them to eliminate mutations from which no test case can be produced. Our evaluation, based on a proof-of-concept tool and an industrial case from the automotive domain, confirms the validity of our theory and demonstrates that our approach can eliminate many of the equivalent mutants (88% in our case study).

CCS CONCEPTS
• Software and its engineering → Software product lines; Software testing and debugging; Real-time systems software.

KEYWORDS
Software product lines, mutation-based testing, real-time systems

ACM Reference Format:

1 INTRODUCTION
Testing a real-time system against safety-critical requirements is a hard problem due to the time-sensitiveness of its behaviour. To help in this task, model-based testing methods automate the generation of test cases by using a formal model of the system [25]. The model drives the generation of test cases according to different criteria, such as state coverage. Testing a formal model rather than source code allows detecting, among others, misinterpretations of requirements or systemic issues arising from time-dependent interactions of the system with its environment. Such detections would be harder at source code level.

Mutation testing [1, 8] is a technique commonly used to evaluate the thoroughness of test cases or to support their generation [4, 22]. It can be applied both to the implementation (source code) and to the specification (model). A set of mutation operators, simulating possible faults in the system, are applied to the model, obtaining a so-called mutant. Thus, given a set of mutants, the effectiveness of a set of test cases can be evaluated according to the number of mutants it detects (i.e., mutants that produce different output than the original system). Test cases generated from a mutant are capable to detect bugs mimicked by that mutation. It has been shown [4] that mutation-based testing is more effective in finding real faults than other techniques [2, 6, 22].

Scalability of this approach is of paramount importance, because a large number of mutations is required in order to build effective test cases. The problem of equivalent mutants hinders this technique, as a mutation is keen to generate a mutant whose behaviour is equivalent to (or a subset of) the original system model [21]. In this case, no test case can be generated to differentiate the mutant from the original one. One viable method is to organise the mutants as a product line of mutations, in the featured mutant model [16]. Such a product line enables the effective generation and validation of mutants against given test cases. However, an efficient featured mutant model should be built upon a set of effective mutations (i.e., those producing non-equivalent mutants), rather than from random mutations. This constitutes an important contribution to avoiding the equivalent mutant problem.

In this paper, we tackle the problem of testing real-time systems effectively and efficiently. We adopt the model-based mutation testing approach for real-time systems presented in [19]. We augment the set of existing mutations with new mutation operators that affect the timing of the system behaviour (e.g., one of our operators delays the execution of some action by the system). Then, we address the equivalent mutant problem [21]: we formally prove the conditions under which some mutations inherently produce equivalent mutants. We achieve this on the basis of refinement relations, which can be used to show that a model (the system) subsumes
another (the mutant). Our endeavour yields clear guidelines for real-
time system engineers, which they can follow in order to reduce
their testing effort by ignoring equivalent mutants.

To summarize, our contributions are as follows.

1. We propose novel time-specific mutation operators for real-
time models.

2. We study and formally prove under which conditions muta-
tion operators ("old" and "new") yield equivalent (or sub-
sumed) mutants, from which no test case can be generated.

3. We implement our approach in a proof-of-concept tool and
validate it based on an industrial system from the automotive
domain.

4. We generalise our theoretical results to the case where muta-
tion testing is applied to a product line of systems, instead of
a single system. In such a case, we use the featured mutant
model to capture both the variability of the system and the
different mutations that can be applied. This requires the
definition of a feature-aware extension of timed graphs (the
mathematical structure used to check refinement relation), in
a similar way that single-system formalisms were extended
with variability [10, 11, 13, 24].

Outline. In Section 2, we discuss related work, followed by back-
ground material on (featured) timed games in Section 3. Our con-
tribution starts in Section 4, where we introduce novel mutation
operators and the updated featured mutant model. Our main con-
tributions are presented in Section 5, where we classify mutation
operators and present guidelines for selecting effective mutations,
followed by Section 6, where we report the results of an evaluation
of our approach. In Section 7, finally, we conclude the paper
and provide some ideas for future work.

2 RELATED WORK

This paper mainly builds upon two recent results on mutation-based
testing [16, 19]. Featured mutant models were introduced in [16] for
efficiently validating test cases against different possible mutations.
Indeed, a single execution on the generated featured transition
system [10] suffices to check all mutants at once. However, there are
no guidelines on how to select the mutations to generate the
featured mutant model, that is, the mutations are selected randomly.

While [16] studies the problem of checking given test cases, [19]
considers the problem of generating valid test cases for real-time
system models. Basically, a test case generated through mutation-
ated testing is guaranteed by construction to distinguish certain
mutants from the system model. Real-time systems are modelled as
timed I/O automata, in which input actions are defined controllable
and output actions are defined uncontrollable. The main idea is to
perform a refinement check between the mutant and the system
model, using Eedar [15]. Eedar is a tool built on top of Uppaal
TIGA [7] that implements the timed interface theory from [14].

Compared to [16], in [19] the mutants are not organised as a
product line and thus have to be checked one by one to generate the
test cases. Moreover, both approaches generate random mutations
that may result ineffective for generating/validating the test cases.

Earlier, in [2], mutation-based testing for timed automata was in-
troduced, extending standard mutation operators presented in [17]
with new mutations tailored for timed automata. We use some of
those mutations but also introduce new ones. Compared to [19], a
$k$-bounded language inclusion test between the mutant and the
system model is used rather than refinement checking with Eedar.

In [2], a Car Alarm System (CAS) model of Ford is used as case
study for experiments and evaluation; here we use the same case
study. Similar to [19], the approach in [2] comes without a proce-
dure or guidelines for selecting effective mutations, and no product
line is used either. In particular, 471 out of a total of 1099 generated
mutants are tested and subsequently discarded, because they can-
not be used for generating test cases. We present a technique for
avoiding the generation of ineffective mutants.

Mutation-based test-case generation is also discussed in [1], for
the case of UML state machine diagrams. The technique for compar-
ing the mutant with the system model is similar to the one in [2],
and the CAS case study is used for experiments. Mutations are
applied randomly and ineffective mutants (i.e., mutants equivalent
to the system model) are discarded subsequent to their generation.

Finally, the survey in [18] points out that "one barrier to wider
application of mutation testing centers on the problems associated
with equivalent mutants". Our paper is an effort in the direction of
reducing the generation of ineffective mutants, within the frame-
work proposed by [19] and adopting the featured mutant model
construction of [16].

3 BACKGROUND

In this section, we provide some background needed for the sequel.

3.1 Timed Games

Timed games (TG) are transition systems which can remain in a
certain state or location only a specific amount of time, can execute
a transition only within a certain time interval, and distinguish
between controllable and uncontrollable actions. TG are based on
timed (game) automata [5, 12] and form the underlying behavioral
structure of featured timed game (automata) [12, 13].

In reactive systems, one usually distinguishes between uncontrol-
able and controllable actions, that are assigned to inputs and
outputs, respectively, if the environment is uncontrollable and vice
versa otherwise. Time is represented by clocks whose values evolve
continuously. Clocks can be regarded as chronometers: their value
can be inspected and reset, but not modified arbitrarily. Conditions
over clock values are called clock constraints.

Definition 3.1 (Clock constraints). A clock constraint over a set
$C$ of clocks is formed according to the grammar $g := T \mid n \sim c \mid g \land g$,
with $n \in \mathbb{N}$, $c \in C$, and $\sim \in \{<, \leq, =, \geq, >\}$.

We denote by $CC(C)$ the set of clock constraints over $C$. In TG,
a clock constraint can label a state or a transition. In the first case,
the constraint is a location invariant, which defines the interval of
time in which the system can be in the state. In the second case, it
is a transition guard specifying the interval of time during which
the system can execute the transition. Note that the domain of
the numeric constants in clock constraints is limited to natural
numbers. Without loss of generality, we could use real numbers.
However, natural numbers facilitate the implementation of clock
constraint by allowing efficient data structures.
We may also write\footnote{10} the set of total functions $\mathcal{T}_G$ by $\mathcal{T}_G = \{\mathcal{T} : \mathbb{R} \rightarrow \mathcal{P}(\mathbb{R})\}$, where $\mathcal{T}$ is a total function associating locations with invariants;

\begin{itemize}
  \item $\mathcal{AP}$ is a set of atomic propositions; and
  \item $\mathcal{L} : \mathcal{L} \rightarrow \mathcal{AP}$ is a total function associating locations with atomic propositions satisfied in those locations.
\end{itemize}

For a transition $t = (\tau, g, a, R, t')$, $\tau$ is the starting location, $g$ is the transition guard, $a$ is the action triggering the transition, $R$ is the subset of clocks to reset, and $t'$ is the target location. We may also write $t$ as $t \rightarrow_{\mathsf{g}, a, R} t'$ and omit $g$ and/or $R$ when immaterial, and instead of $\{x\}$ for a reset of clock $x$, we may also write $x := 0$.

Example 3.3. In Fig. 1 (bottom-right), a TG model of a soda vending machine is depicted. From its initial state $s_0$, the insertion of a euro coin ($\mathcal{E}$) results in the clock being (re)set to zero and a move to state $s_1$. This input action is modeled as a controllable transition (drawn as a solid arc). The vending machine can remain in this state for at most 5 time units but only within 2 time units it can deliver a soda bottle ($\bullet$), returning to its initial state. The latter action is modeled as an uncontrollable transition (drawn as a dotted arc). Note that we may speak of (un)controllable transitions when their action labels are (un)controllable. A TG model of a tea vending machine is depicted in Fig. 1 (top-right).

The semantics of a TG is commonly defined as an infinite transition system (TS) whose states consist of a location and a valuation of the clocks. The transitions can be categorized into two types. Delay transitions do not change the location of the system, but only represent the passing of time. They may occur only if the invariant of the current location is still satisfied after the delay modelled by the transition. Discrete transitions instead occur when the system moves from one location to another. They may occur only if the current clock values satisfy both the guard of the executed transition and the invariant of the target location. After the execution of such transitions, clock values can be reset.

Definition 3.4 (TG semantics). We define the semantics of a TG $\mathcal{T}_G = (\mathcal{L}, \mathcal{Act}, C, \mathcal{Trans}, \mathcal{AP}, \mathcal{L}, \mathcal{Val}, \mathcal{AP}, \mathcal{L}, \mathcal{Val}, \mathcal{AP})$ as the semantics of the TS $\mathcal{TS} = (\mathcal{L} \times \mathcal{Val}(\mathcal{C}), \mathcal{Act} \cup \mathbb{R}_{\geq 0}, \mathcal{Trans}', (s_0, \eta_0), \mathcal{AP} \cup \mathcal{CC}(\mathcal{C}), L')$, denoted by $\mathcal{T}_G$, and such that $\mathcal{Val}(\mathcal{C})$ is the set of clock evaluations, i.e., the set of total functions $\eta : \mathbb{C} \rightarrow \mathbb{R}$ that assign a non-negative real value to every clock; $\mathcal{AP}$ is the set of atomic propositions; $\mathcal{L}$ is the starting location, and $\mathcal{L}$ is the set of locations.

\begin{itemize}
  \item $\mathcal{T}_G$ has $\mathcal{TS}$ as its domain of values;
  \item $\mathcal{T}_G$ has $\mathcal{TS}$ as its range of values;
  \item $\mathcal{T}_G$ is a total function associating locations with invariants; and
  \item $\mathcal{T}_G$ is a transition relation.
\end{itemize}

We may also write $\mathcal{T}_G = \{\mathcal{T} : \mathcal{L} \rightarrow \mathcal{P}(\mathcal{L})\}$ for at most $\mathcal{G}$ time units it can deliver a soda bottle ($\bullet$) before 2 time units have passed. Vending machines with feature $t$ can deliver a cup of tea ($\bullet$) after at least 2 time units have passed (producing tea takes more time). Note that in the presence of both features, after precisely 2 time units have passed, a choice occurs. Both (output) actions are uncontrollable.

FTG model real-time behaviour of a product line. Moreover, from an FTG we can derive TG modelling behaviour of specific products.

3.2 Featured Timed Games

Featured timed games (FTG) extend TG with variability in the same way that featured transition systems (FTS) \footnote{10} extend (labelled) transition systems (LTS). FTS concisely model the behaviour of all products of a product line in a single superimposed LTS through the annotation of transitions with feature expressions, i.e., conditions expressing their existence in products, based on a feature model.

We assume products to be represented by sets of Boolean features and a feature model to be defined as a pair $(\mathcal{F}, \mathcal{P} \subseteq 2^\mathcal{F})$, where $\mathcal{F}$ is a set of features and $\mathcal{P}$ is the set of valid products. The semantics of a feature model $\mathcal{F}$, denoted by $\mathcal{F}^{\mathcal{F}}$, is then its set of valid products. It can be represented by either a propositional formula or by the usual feature diagram. Let $\mathcal{B} = \mathcal{T}, \mathcal{L}$ denote the Boolean constants true (\mathcal{T}) and false (\mathcal{L}), and let $\mathcal{B}(\mathcal{F})$ denote the set of Boolean expressions over $\mathcal{F}$ (i.e., using features as propositional variables).

The elements of $\mathcal{B}(\mathcal{F})$ are also called feature expressions. Formally, a feature expression $\chi$ is a total function $\chi : \{\mathcal{T}, \mathcal{L}\} \rightarrow \{\mathcal{T}, \mathcal{L}\}$ that associates every combination of features with a truth value. A feature expression can be interpreted as a set of products $\mathcal{F}^{\chi}$ defined as all products $p$ for which the induced truth assignment $\tau (\mathcal{F}^{\chi})$ validates $\chi$. Feature expressions and clock constraints allow modeling the behaviour of real-time variable-intensive systems.

Definition 3.5 (Featured timed games). A featured timed game (FTG) is a decuple $(\mathcal{L}, \mathcal{Act}, \mathcal{C}, \mathcal{Trans}, \mathcal{Loc}, \mathcal{Inv}, \mathcal{AP}, \mathcal{L}, \mathcal{F}, \mathcal{P})$ where $(\mathcal{L}, \mathcal{Act}, \mathcal{C}, \mathcal{Trans}, \mathcal{Loc}, \mathcal{Inv}, \mathcal{AP}, \mathcal{L})$ is a TG and

- $\phi$ is a feature model over a finite set $\mathcal{F}$ of features; and
- $\mathcal{F} : (\mathcal{Trans} \cup (\mathcal{Loc} \rightarrow \mathcal{CC}(\mathcal{C}))) \rightarrow \mathcal{B}(\mathcal{F})$ is a total function associating feature expressions to transitions and invariants.

As for FTS, the function $\mathcal{F}$ associates a feature expression $\mathcal{F}$ to some transition $\mathcal{F} = (\mathcal{G}, \mathcal{A}, \mathcal{R}, \mathcal{F}')$ such that $\mathcal{F}(\mathcal{G}) = \chi$ encodes the set of products able to execute $t$. We may also write $\mathcal{F}(\mathcal{G})$ as $\mathcal{F}(\mathcal{G}) = \chi$ and omit $\mathcal{A}$ and/or $\mathcal{R}$ when immaterial. The function $\mathcal{F}$ moreover associates a feature expression $\mathcal{F}$ to a location invariant $\mathcal{F}(\mathcal{L}) = \mathcal{G}$, for some $\mathcal{L} \subseteq \mathcal{L}$, such that $\mathcal{F}(\mathcal{L}) = \chi$, which we may also write as $\mathcal{F}(\mathcal{L}) = \chi$, encodes the set of products with the invariant $\mathcal{G}$ in location $\mathcal{L}$. Note that $\mathcal{F}$ stands for a feature expression that is always satisfied (by any product).

Example 3.6. In Fig. 1 (left), an FTG $\mathcal{FTG}$ of a product line of vending machines is depicted. The feature model is $s \forall t$, with features $s$ for soda and $t$ for tea. From the initial state $s_0$, the insertion of a euro coin ($\mathcal{E}$), which is always possible (the feature expression is always true) and which results in the clock being (re)set to zero, leads to state $s_1$. This is a controllable (input) action. A vending machine can remain in this state for at most 5 time units. Vending machines with feature $t$ can deliver a soda bottle ($\bullet$) before 2 time units have passed. Vending machines with feature $t$ can deliver a cup of tea ($\bullet$) after at least 2 time units have passed (producing tea takes more time). Note that in the presence of both features, after precisely 2 time units have passed, a choice occurs. Both (output) actions are uncontrollable.
This is achieved by projection of an FTG onto a product $p$ obtained in much the same way that an LTS is obtained from an FTS: all transitions and invariants unavailable in product $p$ are removed.

**Definition 3.7 (FTG projections).** The projection of an FTG $ftg = (Loc, Act, C, Trans, Loco, Inv, AP, L, \phi, y)$ onto a valid product $p \in \mathbb{P}^{FMM}$ is the FTG $ftg|_p = (Loc, Act, C, Trans', Loco, Inv', AP, L)$ where

$$Trans' = \{ (t, g, \alpha, r, \ell') \mid t \in Trans \land p \models y(t) \};$$

and

$$Inv'(t) = Inv(t)|_p \quad \forall t \in Loc$$

and the projection of an invariant in much the same way that an LTS is obtained from an FTS: all transitions and invariants unavailable in product $p$ are removed.

**Example 3.8.** In Fig. 1(right), products $ftg|_{\{s\}}$ and $ftg|_{\{t\}}$ of the FTG $ftg$ are depicted. The TG $ftg|_{\{s\}}$ in Fig. 1(bottom-right) is a model of the vending machine that can only deliver soda bottles, whereas the TG $ftg|_{\{t\}}$ in Fig. 1(top-right) is a model of the vending machine that can only deliver tea. Product $ftg|_{\{s\}}$ is not shown.

The semantics of an FTG model of a product line is defined as a function that associates every valid product with the semantics of its projection.

**Definition 3.9 (FTG semantics).** The semantics of an FTG $ftg = (Loc, Act, C, Trans, Loco, Inv, AP, L, \phi, y)$ is defined as the function $\llbracket ftg \rrbracket_{FTG} : \mathbb{P}^{FMM} \to \llbracket ftg \rrbracket_{TG}$ such that

$$\forall p \in \mathbb{P}^{FMM} \cdot \llbracket ftg \rrbracket_{FTG}(p) = \llbracket ftg \rrbracket_{TG}$$

### 4 FEATURED MUTANTS MODEL

The idea underlying mutation-based testing is to guide test-case generation by mutants, which are typically obtained through random mutations of the original model. Organizing the mutants as a product line of mutations, a family of variations of the System Under Test (SUT), coined the Featured Mutant Model (FMM) in [16], enables the efficient generation, configuration, and execution of mutants. Each feature in the FMM corresponds to a single application of one mutant operator on the original model.

#### 4.1 Building Featured Mutants Models

Like [16], we use a selection of the operators proposed by Fabbi et al. [17], based on [9, 26], to generate mutants from a TS:

- **TMI** Transition Missing operator removes a transition;
- **TAD** Transition ADD operator adds a transition between two states;
- **SMI** State Missing operator removes a state (other than the initial state) and all its incoming/outgoing transitions.

Additionally, we introduce the following operators specific to timed models, which change the constant in clock constraints, which we recall to be either a transition guard or location invariant:

- **CXL** Constant eXchange L operator increases the constant of a clock constraint;
- **CSX** Constant eXchange S operator decreases the constant of a clock constraint.
- **CCN** Clock Constraint Negation operator negates a clock constraint.

The CCN operator is inspired by the $\mu_{inv}$ operator from [2], where only clock constraints appearing as transition guards are negated.

Each operator can be used to generate mutants using either the enumerative approach or the FMM approach. In the enumerative approach, each mutation transforms an FTG model $ftg$, representing the SUT behavior, into a mutant $ftg_{\text{fm}}$.

**Example 4.1.** The FTG in Fig. 2(left) has been obtained from the FTG in Fig. 1(left) by applying the mutation operators TMI, CXL, and CSX. The transition labelled with a soda bottle was removed. Moreover, constant 2 in the clock constraint that acts as transition guard was increased to 4 to model that producing a tea takes more time. Instead, constant 5 in the clock constraint that acts as location invariant was decreased to 4 to model that the vending machine takes less time to produce a drink. Thus, the transition from $s_1$ to $s_0$ that models the delivery of a cup of tea now occurs (instantaneously) precisely when $x = 4$. The feature model is not changed.

In the FMM approach, each mutation operator is added as a feature to the existing feature model. When considering first-order mutation (only one mutation can be applied to the original system), the features/mutations are mutually exclusive. For higher order of mutations, disjunction is used instead.

**Example 4.2.** Adding the TMI, CXL, and CSX operators to the FTG from Fig. 1(left), results in the FTG $ftg_{\text{fm}}$ depicted in Fig. 2(right) with feature model $\phi_{\text{fm}}$ depicted in Fig. 3. We now explain this. To begin with, the TMI operator removes the transition $t_1 = s_1 \xrightarrow{x \leq 5} s_0$ of the base model in the following way:

1. The feature expression $\neg tmi$ is added to the feature expression of $t_1$, resulting in transition $s_1 \xrightarrow{x \leq 5} s_0$, meaning that this transition may be fired only if the $tmi$ mutation is deactivated (and if $x$ is true).
2. The feature $tmi$ is added to the feature model $\phi_{\text{fm}}$, representing the application of the mutation operator (cf. Fig. 3).

Moreover, the CXL operator increases the constant 2 to 4 in the clock constraint that acts as guard on the transition $t_2 = s_1 \xrightarrow{x \leq 2} s_0$ of the base model, in the following way:
cxl of a mutation operator (e.g., expression \((\text{FTG})\)). In other words, for a given transition or state, the feature expression. Since we choose to only add mutation operators to the original FMM under construction, mutation operators are added to the FMM under constructing the application of the mutation operator (cf. Fig. 3).

\[
\begin{align*}
&\forall \neg \text{cxl} x \geq 4 \not\in S_0, \text{meaning that this transition may be fired only if the cxl mutation is deactivated (and if } t \text{ is true); \\
&(1) \text{ the feature expression } \neg \text{cxl} \text{ is added to the feature expression of } t_2, \text{ resulting in transition } s_1 \rightarrow t \rightarrow \neg \text{cxl} x \geq 4 \not\in S_0, \text{ meaning that this transition may be fired only if the cxl mutation is deactivated (and if } t \text{ is true);} \\
&(2) \text{ the transition } s_1 \rightarrow t \rightarrow \neg \text{cxl} x \geq 4 \not\in S_0, \text{ is added, meaning that this transition with feature expression } t \wedge \text{cxl} \text{ and clock constraint } x \geq 4 \text{ may be fired only if the cxl mutation is activated (and if } t \text{ is true);} \\
&(3) \text{ the feature cxl is added to the feature model } \varphi_{fmm} \text{ representing the application of the mutation operator (cf. Fig. 3).} \\
\end{align*}
\]

Finally, the CXS operator decreases the constant 5 to 4 in the featured clock constraint \([\neg \text{cxl}] x \leq 5\), which acts as invariant of the state \(s_1\) of the base model, in the following way:

\[
\begin{align*}
&(1) \text{ the feature expression } \neg \text{cxs} \text{ is added to the featured clock constraint of } s_1, \text{ meaning that the updated featured clock constraint } [\neg \text{cxs}] x \leq 5 \text{ acts as invariant } x \leq 5 \text{ of } s_1 \text{ only if the cxs mutation is deactivated;} \\
&(2) \text{ the feature expression cxs is added to the featured clock constraint of state } s_1, \text{ meaning that the updated featured clock constraint } [\text{cxs}] x \leq 4 \text{ acts as invariant } x \leq 4 \text{ of } s_1 \text{ only if the cxs mutation is activated;} \\
&(3) \text{ the feature cxs is added to the feature model } \varphi_{fmm} \text{ representing the application of the mutation operator (cf. Fig. 3).} \\
\end{align*}
\]

Hence, mutation operators are added to the FMM under construction. Since we choose to only add mutation operators to the original FTG, this prohibits applying mutations to mutants (i.e., products of the FTG). In other words, for a given transition or state, the feature expression \((y)\) cannot contain more than one activatable feature of a mutation operator (e.g., \(\text{cxl} \wedge \neg \text{cxs} \wedge \neg \ldots\) is allowed, whereas \(\text{cxl} \wedge \text{cxs} \wedge \ldots\) is not).

**5 CLASSIFYING MUTATIONS**

Our main contribution is a classification of mutations to identify those that are effective (can be used to generate test cases). Our key idea is that, by construction, some mutations produce mutants that have the same (or a subset of the) behaviour of the SUT. Discarding them will speed-up the mutation testing process, as we would avoid fruitless attempts to generate test cases. Thus, we aim to characterize these mutations by formally proving under which conditions (i.e., mutation operator and the elements of the model to which it is applied) the produced mutant is equivalent to (or subsumed by) the SUT.

Recall that a test case generated from a mutant provides a sequence of inputs that makes the mutant behave differently than the SUT (in terms of accepted inputs, produced outputs, or execution time). Thus, the goal of the test case is to distinguish whether the system on which it is executed is the original one or the mutant. For a mutant to remain "live" (as named in the jargon), there must be no test case that can distinguish it from the SUT. This is equivalent to proving that the mutant is a refinement of the SUT \([19]\). Refinement checking is solved as a two-player timed game, where one player (playing the "whenever" transitions of the forthcoming Definition 5.1) wins if the mutant is not a refinement of the system (the mutant is killed) and the other player (playing the "then" transitions of Definition 5.1) wins if the mutant is a refinement (the mutant is alive). If the mutant is not a refinement, then the counterexample represents the test case that distinguishes the mutant from the SUT.

In what follows, we consider the mutation operators mentioned in Section 4 and prove under which conditions their application results in a refinement of the original model. While our endeavour primarily concerns first-order mutation, which was shown to offer a higher fault-revealing ability \([23]\), our theoretical results hold for higher-order mutation as well. As such, when proving refinement relations, we consider the general case where mutations are applied to mutants of the SUT (either equivalent or not). Similarly, our work generalizes to the case where the original model represents the behaviour of not only one system, but of a whole product line of systems. Thus, our theoretical developments are defined over FTG rather than single TG. To summarize, all results described hereafter apply to (1) any-order mutations and (2) families of systems.

To begin with, we formalize the notion of refinement between TG, adapted from \([14, 19]\). Basically, a refinement model (i.e., a live mutant) must be able to mimic all controllable transitions of the original system model, while the original model must be able to mimic all uncontrollable transitions of the refinement. In our case, controllable transitions correspond to inputs (since a live mutant must accept all inputs that the original system accepts), whereas uncontrollable transitions correspond to outputs and delays (since a live mutant should not exhibit any behaviour that does not belong to the system). Note that this is the opposite of the standard notion of modal refinement, where the inputs are seen as sent by an uncontrollable environment \([20]\). In other words, here the viewpoint is switched to the environment \([14, 19]\).

**Definition 5.1 (Refinement).** A TG \(\mathcal{G}_1 = (\text{Loc}_1, \text{Act}_1, \text{C}_1, \text{Trans}_1, \text{l}_1, \text{Inv}_1, \text{AP}_1, \text{L}_1)\) is a refinement of a TG \(\mathcal{G}_2 = (\text{Loc}_2, \text{Act}_2, \text{C}_2, \text{Trans}_2, \text{l}_2, \text{Inv}_2, \text{AP}_2, \text{L}_2)\), denoted as \(\mathcal{G}_1 \preceq \mathcal{G}_2\), if there exists a...
binary relation $R \subseteq (\text{Loc}_1, \text{Val}_1(C_1)) \times (\text{Loc}_2, \text{Val}_2(C_2))$ that contains $s = ((l_1, \eta_1), (l_2, \eta_2))$ and is such that for each pair of locations $((l_1, \eta_1), (l_2, \eta_2)) \in R$, it holds:

- whenever $(l_2, \eta_2) \xrightarrow{a}(l'_2, \eta'_2)$ for some $l'_2$ and $a \in \text{Act}_r$, then $(l_1, \eta_1) \xrightarrow{a}(l'_1, \eta'_1)$ for some $l'_1$ and $((l'_2, \eta'_2), (l'_1, \eta'_1)) \in R$
- whenever $(l_1, \eta_1) \xrightarrow{a}(l'_1, \eta'_1)$ for some $l'_1$ and $a \in \text{Act}_r$, then $(l_2, \eta_2) \xrightarrow{a}(l'_2, \eta'_2)$ for some $l'_2$ and $((l'_1, \eta'_1), (l'_2, \eta'_2)) \in R$
- whenever $(l_1, \eta_1) \xrightarrow{\delta}(l_1, \eta'_1)$ for some $\eta'_1$ and $\delta \in \mathbb{R}_{>0}$, then $(l_2, \eta_2) \xrightarrow{\delta}(l_2, \eta'_2)$ for some $\eta'_2$ and $((l_1, \eta'_1), (l_2, \eta'_2)) \in R$

We now provide a definition of subsumed mutant, where $\text{Opt}_{\text{fmm}}$ is the set of mutations. Basically, after applying an additional mutation the resulting mutant is a refinement of the former one on which the additional mutation was not applied.

**Definition 5.2 (Subsumed mutant).** Let $f(t) be an FTG and let $\mathcal{M}$ be the set of mutants with $m, m' \in \mathcal{M}$. We say that $m$ is an increment of $m'$ iff $m = m' \cup \mathcal{O}$ for some $\mathcal{O} \in \text{Opt}_{\text{fmm}}$ (called the increment). Moreover, we say that $m$ is subsumed by $m'$ iff $f(t)_{m'} \subseteq f(t)_{m'}$, and we say that it is non-subsumed otherwise.

A TG is said to be non-redundant if every location $l$ is reachable in at least one trace, it is not time-locked (i.e., delay is possible), and every transition is executable in at least one trace. We will only consider non-redundant TG, both for the specification and for the generated mutants (for the non-subsumed lemmata). Note that to transform a redundant TG into a non-redundant one, it suffices to remove such redundant locations or transitions, and time-locked or redundant specifications are ill-defined and should be amended anyway. Moreover, we consider only deterministic TG, as usual [1, 2, 19].

**Proposition 5.3 (Subsumed Mutant).** Let $f(t) be an FTG, let $\mathcal{M}$ be the set of mutants with $m, m' \in \mathcal{M}$, and let $m$ be an increment of $m'$ with increment $\mathcal{O}$. Then $m$ is a subsumed mutant of $m'$ if $\mathcal{O}$ has introduced either less uncontrollable or more controllable behavior.

**Proof.** By Definitions 5.1 and 5.2, where delays are uncontrollable, and the assumption that $f(t)_{m'}$ is non-redundant. $\Box$

In the remainder of this section, we present several results for identifying mutations that generate (non-)subsumed mutants by construction. We start with the works that were proposed by Fabbri et al. [17], followed by novel ones introduced in this paper.

### 5.1 TMI mutation

The TMI mutation is used to remove a transition from the system. The following lemma shows that the application of a mutation TMI on a transition $t$ (tm$t_i$ in the following) of a mutant $m'$, i.e., removing such a transition from $m'$, produces by construction a mutant $m$ that is non-subsumed by $m'$, in case $t$ is controllable.

**Lemma 5.4 (TMI Non-Subsumed).** Let $f(t) be an FTG and let $\mathcal{M}$ be the set of mutants with $m, m' \in \mathcal{M}$, and $m = \{tm_i\} \cup m'$ for some $t \in Trans_{f(t)_{m'}}$ with action in $\text{Act}_{r}$. Then $f(t)_{m}$ is non-subsumed by $f(t)_{m'}$.

**Proof.** By Proposition 5.3 and the fact that $tm_i$ has removed controllable behaviour, and the assumption that the mutant $m'$ is non-redundant. $\Box$

The next lemma shows that removing an uncontrollable transaction from a mutant, by construction the resulting mutant is subsumed by the original one.

**Lemma 5.5 (TMI Subsumed).** Let $f(t) be an FTG and let $\mathcal{M}$ be the set of mutants with $m, m' \in \mathcal{M}$, and $m = \{tm_i\} \cup m'$ for some $t \in Trans_{f(t)_{m'}}$ with action in $\text{Act}_{r}$. Then $f(t)_{m}$ is subsumed by $f(t)_{m'}$.

**Proof.** By Proposition 5.3, and the fact that $tm_i$ has introduced less uncontrollable behaviour. $\Box$

**Example 5.6.** Let us illustrate the usefulness of these results by means of an example. Recall that test-case generation is more effective if the number of subsumed mutants is minimised. First consider the FTG $f(t)_{\text{fmm}}$ from Example 4.2, depicted in Fig. 2(right), and its mutants $m_1 = \{s, tm_i\} \cup \{s'\}$, depicted in Fig. 4(left) and Fig. 1(bottom-right), respectively, i.e., with $t_1 = s_2 \xrightarrow{\infty} s_3$. Since $t_1$ is an uncontrollable transition, Lemma 5.5 implies that $f(t)_{m_1}$ is non-subsumed by $f(t)_{m_{1'}},$ i.e., this is not a good candidate mutation for the configuration $m_1'.$

Next consider the FTG $f(t)_{\text{fmm}}'$ depicted in Fig. 4(middle), and its mutants $m_2 = \{tm_i\} \cup \{s_{1'}, s_{2'}, \ldots, s_{n'}\}$, depicted in Fig. 4(right) and Fig. 4(left), respectively, i.e., with $t_2 = s_0 \xrightarrow{t_{2}} s_1$. Since $t_2$ is a controllable transition, Lemma 5.4 implies that $f(t)_{m_2}$ is non-subsumed by $f(t)_{m_{2'}},$ i.e., this is a good candidate mutation for the configuration $m_{2'}.$

### 5.2 TAD mutation

The TAD mutation is used to add a transition to the system. Under the (assumed) hypothesis that the added transition is executable in at least one trace, such a mutation produces a non-subsumed mutant if an uncontrollable transition is added and a subsumed mutant if a controllable transition is added.

The next lemma shows that the application of a mutation TAD on a transition $t$ (tad$t_i$ in the following) of a mutant $m'$, i.e., adding such a transition to $m'$, produces by construction a mutant $m$ that is non-subsumed by $m'$, in case $t$ is uncontrollable.

**Lemma 5.7 (TAD Non-Subsumed).** Let $f(t) be an FTG and let $\mathcal{M}$ be the set of mutants with $m, m' \in \mathcal{M}$, and $m = \{tad_i\} \cup m'$ for some $t \in Trans_{f(t)_{m'}}$ with action in $\text{Act}_{r}$. Then $f(t)_{m}$ is non-subsumed by $f(t)_{m'}$.

**Proof.** By Proposition 5.3, the fact that $tad_i$ has added uncontrollable behaviour, and the assumption that the mutant $m'$ is non-redundant. $\Box$
The next lemma shows that by adding a controllable transition to a mutant, the obtained mutant is subsumed by the original one.

Lemma 5.8 (TAD Subsumed). Let $\overline{ftg} \in FTG$ and $[[\varphi]]$ be the set of mutants with $m, m' \in [[\varphi]]$ and $m = \{tad\} \cup m'$ for some $t \in Trans_{\overline{ftg}m'}$ with action in Act$^e$. Then $\overline{ftg} | m'$ is subsumed by $\overline{ftg} | m$. 

Proof. By Proposition 5.3, the fact that $tad$ has introduced more controllable behaviour, and the assumption that the mutant $m'$ is non-redundant.

5.3 SMI mutation

The state missing SMI mutation removes a location from the system (not the initial location however). This is equivalent to making the location unreachable, i.e., removing all its incoming transitions. Hence, the results on T111 can be applied.

Lemma 5.9 (SMI Subsumed). Let $ftg$ be an FTG and let $[[\varphi]]$ be the set of mutants with $m, m' \in [[\varphi]]$ and $m = \{\text{smi}\} \cup m'$ for some $l \in Loc_{ftg,m'}$. Then $ftg | m$ is subsumed by $ftg | m'$ if there exists no transition $t$ with target location $l$ and action $a \in Act^e$.

Proof. By contradiction, assume w.l.o.g. that $l$ has one incoming transition $t$, action $a \in Act^e$, and that $ftg | m$ is subsumed by $ftg | m'$ (since we are only removing transitions this case can be extended to many incoming transitions). Hence, the mutant $m = \{\text{smi}\} \cup m'$ is equivalent to the mutant $m = \{\text{smi}\} \cup m''$, and by applying Lemma 5.4, $ftg | m$ is non-subsumed by $ftg | m'$, a contradiction.

In the next two sections, we continue with the newly introduced mutation operators.

5.4 CXL mutation

We first turn our attention to the mutation CXL, that increases the constant of a clock constraint. Let $c$ be a clock and let $k$ be some constant. Then such a mutation does not generate a subsumed mutant when applied to a guard of the form $x \leq k$ of an uncontrollable transition or to a guard of the form $x \geq k$ of a controllable transition, under conditions discussed in the next lemma.

Lemma 5.10 (CXL Non-subsumed Transitions). Let $ftg$ be an FTG and let $[[\varphi]]$ be the set of mutants with $m, m' \in [[\varphi]]$ and $m = \{cxl\} \cup m'$ for some $t \in Trans_{ftg,m'}$ with source $l$ and either (i) action $a \in Act^e$, guard $g = x \leq k$ and there exists a valuation of clock $k < v \leq k'$ for $k'$ mutation such that $(l, v) \notin ftg | m'$ or (ii) action $a \in Act^e$, guard $g = x \geq k$ and there exists a valuation of clock $k < v \leq k'$ for $k'$ mutation such that $(l, v) \notin ftg | m'$. Then $ftg | m$ is non-subsumed by $ftg | m'$.

Proof. By Proposition 5.3 and the fact that $t$ is non-redundant, $m$ has introduced either (i) more uncontrollable behaviour or (ii) less controllable behaviour.

The next lemma instead shows when the mutation operator CXL applied on a transition produces a mutant that is subsumed.

Lemma 5.11 (CXL Subsumed Transitions). Let $ftg$ be an FTG and let $[[\varphi]]$ be the set of mutants with $m, m' \in [[\varphi]]$ and $m = \{cxl\} \cup m'$ for some $t \in Trans_{ftg,m'}$ with source $l$ and either (i) action in Act$^e$ and guard $g = x \leq k$ or (ii) action in Act$^e$, guard $g = x \geq k$ and Inv$(l) = x \leq k$ or (iii) action in Act$^e$ and guard $g = x \geq k$. Then $ftg | m$ is subsumed by $ftg | m'$.

Proof. By Proposition 5.3, the increment mutation introduces (i) more controllable or (ii,iii) less uncontrollable behaviour.

Example 5.12. Recall from Example 4.2 the mutation operator CXL applied to the FTG $ftg_{fmm}$ depicted in Fig. 2(right), and consider its mutants $m_1 = \{t, cxl_x, cxl_{s_0}\}$ and $m'_1 = \{t, cxl_{s_0}\}$, depicted in Fig. 5(left) and Fig. 5(right), respectively, i.e., with $t_2 = s_1 \xrightarrow{s_2} s_0$. Since $t_2$ is an uncontrollable transition, Lemma 5.11(iii) implies that $ftg_{fmm} | m_1$ is subsumed by $ftg_{fmm} | m'_1$, i.e., this is not a good candidate mutation for the configuration $m'_1$.

The next lemma provides the conditions under which the application of the mutation operator CXL on an invariant of a location $l$ (written as cxl$_l$), yields a non-subsumed mutant.

Lemma 5.13 (CXL Non-subsumed Invariants). Let $ftg$ be an FTG and let $[[\varphi]]$ be the set of mutants with $m, m' \in [[\varphi]]$ and $m = \{cxl\} \cup m'$ for some location $l \in Loc_{ftg,m'}$ and either (i) Inv$(l) = x \geq k$ and there exists a valuation $v$ of clock $x$ such that $k < v \leq k'$ for $k'$ mutation such that $(l, v)$ is reached through a transition with action in Act$^e$ and target $l$ or (ii) Inv$(l) = x \leq k$. Then $ftg | m$ is non-subsumed by $ftg | m'$.

Proof. For case (i), the increment mutation has less controllable behaviour and (ii) the increment mutation has more uncontrollable (timing) behaviour. In both cases, by Proposition 5.3, $ftg | m$ is non-subsumed by $ftg | m'$.

Finally, the next lemma identifies the conditions under which applying CXL on an invariant yields a subsumed mutant.

Lemma 5.14 (CXL Subsumed Invariants). Let $ftg$ be an FTG and let $[[\varphi]]$ be the set of mutants with $m, m' \in [[\varphi]]$ and $m = \{cxl\} \cup m'$ for some location $l \in Loc_{ftg,m'}$ with Inv$(l) = x \geq k$ and for all valuations $v$ of clock $x$ such that $k < v \leq k'$ for $k'$ mutation, $(l, v)$ can only be reached through a transition with action in Act$^e$ and target $l$. Then $ftg | m$ is subsumed by $ftg | m'$.

Proof. The mutation has more uncontrollable behaviour. By Proposition 5.3, $ftg | m$ is non-subsumed by $ftg | m'$.

5.5 CXS mutation

We now turn our attention to the mutation CXS that decreases the constant of a clock constraint. Again, let $c$ be a clock and let $k$ be some constant. Then such a mutation produces a non-subsumed mutant when applied to a guard of the form $x \leq k$ of a controllable transition or to a guard of the form $x \geq k$ of an uncontrollable transition, as the next lemma shows.
LEMMA 5.15 (CXS Non-subsumed Transitions). Let ftg be an FTG and let \([\varphi]\) be the set of mutants with \(m, m' \in [\varphi]\) and \(m = \{\text{cxs}_x\} \cup m'\) for some \(t \in \text{Trans}_{\text{fmm}}\) with source \(t\) and either (i) action in \(\text{Act}^t\), \(g = x \leq k\) and there exists a clock valuation \(k' < u \leq k\) with \(k'\) mutation such that \((l, o) \not\rightarrow_{\text{fmm}}^t\), or (ii) action in \(\text{Act}^u\), \(g = x \geq k\) and there exists a clock valuation \(k' \leq u < k\) with \(k'\) mutation such that \((l, o) \not\rightarrow_{\text{fmm}}^t\). Then \(\text{ftg}\mid_{\text{m}}\) is non-subsumed by \(\text{ftg}\mid_{\text{m'}}\).

Proof. By Proposition 5.3 and the fact that \(t\) is reachable and thus executable, \(m\) has introduced either (i) less controllable behaviour or (ii) more uncontrollable behaviour. This is because (i) \(g = x \leq k\) and the mutation has changed the guard into \(g' = x \leq k'\) for some \(k' < k\) or (ii) \(g = x \geq k\) and the mutation has changed the guard into \(g' = x \geq k'\) for some \(k' < k\).

The next lemma shows that the mutation operator \(\text{cxs}\) applied to a form \(x \leq k\) of an uncontrollable transition or to a guard of the form \(x \geq k\) of a controllable transition produces a mutant that is subsumed.

LEMMA 5.16 (CXS Subsumed Transitions). Let ftg be an FTG and let \([\varphi]\) be the set of mutants with \(m, m' \in [\varphi]\) and \(m = \{\text{cxs}_x\} \cup m'\) for some \(t \in \text{Trans}_{\text{fmm}}\) with either (i) action in \(\text{Act}^t\) and guard \(g = x \leq k\); or (ii) action in \(\text{Act}^t\) and guard \(g = x \geq k\). Then \(\text{ftg}\mid_{\text{m}}\) is subsumed by \(\text{ftg}\mid_{\text{m'}}\).

Proof. By Proposition 5.3 and hypothesis, the increment mutation is introducing (i) more controllable behaviour or (ii) less uncontrollable behaviour. □

The next lemma provides the conditions under which the application of the mutation operator \(\text{cxs}\) on an invariant of a location \(l\) (written as \(\text{cxs}_x\)) produces a non-subsumed mutant.

LEMMA 5.17 (CXS Non-subsumed Invariants). Let ftg be an FTG and let \([\varphi]\) be the set of mutants with \(m, m' \in [\varphi]\) and \(m = \{\text{cxs}_x\} \cup m'\) for some \(l \in \text{Loc}_{\text{fmm}}\) and either (i) \(\text{Inv}(l) = x \geq k\) and there exists a valuation \(v\) of clock \(x\) such that \(k' \leq v < k\) for \(k'\) mutation such that \((l, o) \rightarrow_{\text{fmm}} t\) and target \(l\) or (ii) \(\text{Inv}(l) = x \leq k\) and there exists a valuation \(v\) of clock \(x\) such that \(k' \leq v < k\) for \(k'\) mutation such that \((l, o) \rightarrow_{\text{fmm}} t\) and target \(l\). Then \(\text{ftg}\mid_{\text{m}}\) is non-subsumed by \(\text{ftg}\mid_{\text{m'}}\).

Proof. By Proposition 5.3, the increment mutation has more (i) uncontrollable behaviour or (ii) less controllable behaviour. □

Example 5.18. Recall from Example 4.2 the mutation operator \(\text{cxs}\) applied to the FTG \(\text{ftg}_{\text{fmm}}\) depicted in Fig. 2(right), and consider its mutants \(m_1 = \{t, \text{cxs}_x\}\) and \(m'_1 = \{t\}\), depicted in Fig. 5(right) and Fig. 1(top-right), respectively. Since the clock control \(x \leq 5\) acting as an invariant of \(s_1\) is reached through the controllable transition \(s_0 \rightarrow s_1\), Lemma 5.17(ii) implies that \(\text{ftg}_{\text{fmm}}\mid_{m_1}\) is non-subsumed by \(\text{ftg}_{\text{fmm}}\mid_{m'_1}\), i.e., this is a good candidate mutation for the configuration \(m'_1\).

Finally, the next lemma identifies the conditions under which the application of \(\text{cxs}\) on an invariant yields a subsumed mutant.

LEMMA 5.19 (CXS Subsumed Invariants). Let ftg be an FTG and let \([\varphi]\) be the set of mutants with \(m, m' \in [\varphi]\) and \(m = \{\text{cxs}_x\} \cup m'\) for some \(l \in \text{Loc}_{\text{fmm}}\) and either (i) \(\text{Inv}(l) = x \leq k\) and for all valuations \(v\) of clock \(x\) such that \(k' \leq v < k\) for \(k'\) mutation, \((l, o) \rightarrow_{\text{fmm}} t\) can only be reached through a transition with action in \(\text{Act}^t\) and target \(l\) or (ii) \(\text{Inv}(l) = x \geq k\) and for all valuations \(v\) of clock \(x\) such that \(k' \leq v < k\) for \(k'\) mutation, \((l, o) \rightarrow_{\text{fmm}} t\) can only be reached through a transition with action in \(\text{Act}^t\) and target \(l\). Then \(\text{ftg}\mid_{\text{m}}\) is subsumed by \(\text{ftg}\mid_{\text{m'}}\).

Proof. By Proposition 5.3, the increment mutation has either (i) less uncontrollable or (ii) more controllable behaviour. □

5.6 CCN mutation

Finally, we turn our attention to the CCN operator that negates a clock constraint of a transition. For all non-redundant TG, this mutation always generates a non-subsumed mutant.

LEMMA 5.20 (CCN Non-subsumed). Let ftg be an FTG and let \([\varphi]\) be the set of mutants with \(m, m' \in [\varphi]\) and \(m = \{\text{ccn}_x\} \cup m'\) for some \(t \in \text{Trans}_{\text{fmm}}\). Then \(\text{ftg}\mid_{\text{m}}\) is non-subsumed by \(\text{ftg}\mid_{\text{m'}}\).

Proof. Negating a constraint always removes some behaviour from one side and adds some behaviour to the other side, thus the result holds by Proposition 5.3 and the assumption of non-redundancy of the mutant. □

5.7 Classifying Mutations

The following main theorem sums up the results presented so far. The specific additional conditions that need to hold for each mutation operator can be found in the corresponding lemmata.

THEOREM 5.21 (Classifying Mutations). Let ftg be an FTG and let \([\varphi]\) be the set of mutants with \(m, m' \in [\varphi]\) and with \(m\) increment of \(m'\) for some mutation \(o \in \text{Op}_{\text{fmm}}\): Table 1 summarizes when \(\text{ftg}\mid_{\text{m}}\) is (non-)subsumed by \(\text{ftg}\mid_{\text{m'}}\) based on the applied mutation, provided that both \(\text{ftg}\mid_{\text{m}}\) and \(\text{ftg}\mid_{\text{m'}}\) are non-redundant.

Proof. The proof is obtained by cases, applying the lemmata discussed in this section. □

5.8 Generating Effective Mutations

Based on our results, we provide guidelines for generating effective FMM and their corresponding FTG. The FTG in Fig. 2(right) is an example of a "bad" model. This is because two out of three mutants of the model are subsumed, and a subsumed mutant cannot be used to generate effective test cases [19]. Hence, while building the FMM and the corresponding FTG (cf. Section 4.1), ideally one wants to minimise the number of subsumed mutants, thus maximising the effectiveness of the test-case generation phase. To do so, we select from Table 1 those results applicable by only checking the syntax of the original model, rather than those based on the semantics.

Finally, we note that such guidelines could be implemented directly as constraints in the feature model of the FMM (cf. Fig. 3), such that subsumed mutants are prevented from being generated. In the next section, we provide an empirical evaluation of the results presented in this section, i.e., certain subsets of mutations are guaranteed to produce mutants that are a refinement of the SUT and thus there is no need to use them.
6 Evaluation

To further validate our theoretical results and their benefits, we conduct an empirical evaluation based on a proof-of-concept tool we developed.

The 12 Commandments of Model-Based Mutation Testing:

1. the SUT shall not be redundant;
2. TMI shall not be applied to uncontrollable transitions;
3. TAD shall not be applied to uncontrollable transitions;
4. SMI shall not be applied when all incoming transitions are uncontrollable;
5. CXL shall not be applied to uncontrollable transitions with guards of the form \( x \leq k \);
6. CXL shall not be applied to uncontrollable transitions with guards either (i) \( x \geq k \) or (ii) \( x = k \) and source invariant \( x \leq k \);
7. CXL can be applied to invariants of the form \( x \leq k \);
8. CXL shall not be applied to invariants of the form \( x \geq k \) whenever all incoming transitions are uncontrollable;
9. CXS shall not be applied to uncontrollable transitions with guards of the form \( x \geq k \);
10. CXS shall not be applied to uncontrollable transitions with guards of the form \( x \leq k \);
11. CXS shall not be applied to invariants of the form \( x = k \) whenever all incoming transitions are uncontrollable;
12. CXS shall not be applied to invariants of the form \( x \geq k \) whenever all incoming transitions are controllable.

Table 1: Refinement subsumption for mutation operators.

<table>
<thead>
<tr>
<th>subject</th>
<th>operator</th>
<th>TMI</th>
<th>TAD</th>
<th>SME</th>
<th>CXL</th>
<th>CXS</th>
<th>CCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>controllable transition</td>
<td>clock constraint ( \leq )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>controllable transition</td>
<td>clock constraint ( \geq )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>uncontrollable transition</td>
<td>clock constraint ( \leq )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>uncontrollable transition</td>
<td>clock constraint ( \geq )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>location</td>
<td>( \not) controllable transition</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>invariant clock constraint</td>
<td>( \not) controllable transition</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>invariant clock constraint</td>
<td>clock constraint ( \leq )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>invariant clock constraint</td>
<td>clock constraint ( \geq )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.1 Research Questions and Methodology

The objective of our work is to identify the mutation operators and the conditions under which a non-effective (i.e., subsumed) mutant is generated. We already addressed this by formally proving that mutants resulting from specific operators are subsumed under specific conditions, as reported in Theorem 5.21 and Table 1. To raise confidence in our results, we confront our theory with a practical implementation. Thus, we ask:

RQ1: Are our guidelines sound, i.e., are all mutants rejected by the guidelines indeed subsumed mutants?

Our next question concerns the benefits of avoiding the generation of mutants that are subsumed by construction. In practice, the saved computation time is dependent on the concrete test-case generation and execution platform. Instead, we measure these benefits in a relative way, as the percentage of subsumed mutants that our guidelines can detect. Thus, we ask:

RQ2: How complete are our guidelines in detecting equivalent/subsumed mutants?

To answer these questions, we apply the mutation operators to a given original model to produce first-order mutants. Then, we check whether those mutants violate guidelines and whether they are subsumed by the original system model, using the refinement check implemented in the Eedar tool [15]. Each operator is systematically applied to each relevant element (location or transition) of the model, producing the complete set of first-order mutants. This allows us to validate our approach on a variety of occurrences. Thus, higher-order mutants would not bring additional insights.

6.2 Tools and System

6.2.1 Implementations. A proof-of-concept tool \(^1\) has been implemented to automatically generate mutants of a system model. The generated mutants are then processed through a batch script. They are checked against the refinement and the results, together with the performance, are stored. To enable the replication of our results, we provide all the mutants analysed, together with documents describing for each mutation the element mutated by each mutant, as well as the log results of Eedar.

6.2.2 Subject System. The Car Alarm System (CAS) model stems from Ford’s automatic demonstrator in the MOGENTES project. This model has already been used for experiments in [1, 2, 19]. The system model allows us to simulate unlocking, locking, closing, and opening of a car’s door. The outputs are the signals for arming, unarming, and turning the sound and flash alarms on and off. We mainly used the adaption of the model to Eedar in [19], in which all input transitions are marked as controllable and all output transitions as uncontrollable. The CAS model is composed of 17 states, of which 11 states have invariants. All invariants are of the form \( x \leq k \) for a clock \( x \) and constant \( k \). There are 88 transitions, out of which 17 are uncontrollable and 71 are controllable. Only 20 transitions are guarded, out of which 14 transitions have guards of the form \( x \leq k \), 3 transitions have guards of the form \( x \geq k \), and 3 transitions have guards of the form \( x = k \), for some clock \( x \) and constant \( k \).

\(^1\) available at https://bitbucket.org/maxcordy/timed-mutation
6.3 Validation Results

Here we interpret the results of our experiments for RQ1 and RQ2.

A total of 721 mutants were generated. We applied all mutation operators discussed in the paper, namely TMI, TAD, SMd, CXL, CXS, and CN. For each mutation, the results in Table 2 report the total number of mutants, of non-subsumed mutants, of subsumed and redundant mutants, and of mutants violating the guidelines, as well as the ratio of mutants that need not be generated by our approach, computed as \(\frac{\text{non-subsumed + redundant}}{\text{violating guidelines}}\). From the guidelines given in Section 5.8, rules (8), (10), and (12) were not applicable to the CAS model.

The experiments confirmed that all mutants violating guidelines were subsumed, as expected, thus providing further confidence in our results and answering our first research question. Note that all subsumed and redundant mutants that are violating the guidelines are those that are indeed not generated if one applies the rules (guidelines) presented in Section 5.8.

To answer the second research question, our results confirm the gain of our approach: 88% of the subsumed mutants generated for this case study can be avoided. Moreover, we avoid the potential generation of up to \(2^{326}\) featured mutant models whose mutations do not produce any test case. Those mutants that have not been detected by the guidelines are discussed next. For CXS, one location did not have all incoming transitions uncontrollable and thus rule (11) could not be applied. The controllable input transitions, however, are not redundant in the mutant and it is subsumed. Similarly, for SMd, one location did not have all incoming transitions uncontrollable and thus rule (4) could not be applied. However, removing such a state makes the mutant redundant. Finally, for TAD, 42 mutants that were not detected have redundant uncontrollable transitions that have been added by the mutation, and thus are subsumed.

6.4 Threats to Validity

One threat to the generality of our results is that we considered only one case study in our evaluation. While we are confident in our theoretical results, the practical benefits may depend on the considered case and the generated mutants. In particular, we note that in the case study 80% of the transitions are uncontrollable and 27% of the guidelines were not applicable. Nevertheless, our set of guidelines touch upon all elements of the model and, as such, should be able to capture most of the equivalent mutants.

Our empirical results essentially rely on the refinement relation as implemented in Ecdar. Should this implementation deviate (even slightly) from the definition we employ (itself based on the paper introducing Ecdar [15]), we might witness the occurrence of false positives (mutants wrongly labelled as subsumed). To mitigate this risk, we conducted manual analyses on sampled mutants. Nevertheless, removing false positives would actually improve our results, as it would increase the percentage of subsumed mutants detected by our guidelines.

7 CONCLUSION AND FUTURE WORK

We presented a methodology for identifying effective mutations for testing real-time systems. An effective mutant can be used to generate test cases that distinguish the mutant from the original system model. The framework of TG and Ecdar refinement checking of [19] was adopted, and mutants are organised as a product line of mutations using the approach of [16]. Our guidelines to the construction of such a featured mutant model can be encoded as constraints in the feature model, to guarantee that effective mutants will be generated. Our experiments confirmed the soundness of our approach and demonstrated that our actionable guidelines can significantly reduce the number of equivalent/subsumed mutants.

As future work, a family-based technique for checking refinements all-at-once directly on the FTG will be investigated, in order to take further advantage of the product-line approach and of our technique for building effective featured mutant models. That would allow the generation of the smallest set of test cases that can distinguish all killable mutants. One way to do so is to design a feature-aware extension of refinement checking procedure of [14, 15]. By associating a feature to each mutant [16], one can then collect the feature expressions identifying all mutants for which the refinement holds, and those for which it does not, in a single play. This problem was studied in the non-timed case [11] but remain unaddressed for real-time systems. The addition of time makes this problem challenging, as there is no known efficient way to encode time and variability in a single data structure [13].

Our work also provides the foundations to evaluate real-time test cases. To this aim, one can apply the approach of [16] on a featured timed model to identify which mutants are killed. Again, this would require data structures combining time with variability.

Finally, regarding higher-order mutations, the possibility of mutating a mutant would allow the incremental application of the results presented in this paper. For example, a second-order mutant can be obtained either by applying two mutations to the original system model or one mutation to a first-order mutant. Moreover, we would like to study the partial order of mutants induced by the refinement relation. We conjecture that this would allow to further reduce the space of mutants to be generated. For example, two first-order mutants \(m_1\) and \(m_2\) could be non-subsumed by the original model, but \(m_2\) could be a refinement of \(m_1\) (because \(m_2\) is mutating an element redundant in \(m_1\)). In such cases, the test case generated from \(m_2\) could be used to detect both \(m_1\) and \(m_2\). Thus, \(m_1\) could be discarded in favour of \(m_2\). Generally, the least elements in the partial order could be used for generating test cases to detect all subsuming mutants. This reasoning could be automated such that only the set of subsuming mutants is generated.

Table 2: Validation results for RQ1 and RQ2.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Mutants</th>
<th>Non-subsumed</th>
<th>Subsumed</th>
<th>Redundant</th>
<th>Violating guidelines</th>
<th>Violating subsumed + redundant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMI</td>
<td>88</td>
<td>71</td>
<td>17</td>
<td>17</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>TAD</td>
<td>578</td>
<td>247</td>
<td>331</td>
<td>289</td>
<td>87%</td>
<td></td>
</tr>
<tr>
<td>SMd</td>
<td>16</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>89%</td>
<td></td>
</tr>
<tr>
<td>CXL</td>
<td>20</td>
<td>14</td>
<td>6</td>
<td>6</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>CXS</td>
<td>13</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>86%</td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td>721</td>
<td>351</td>
<td>370</td>
<td>326</td>
<td>88%</td>
<td></td>
</tr>
</tbody>
</table>
REFERENCES

svr.1552


