




## Edición especial 25 años del doctorado en ingeniería

Applying parallelism to a bisimulation algorithm to improve efficiency in software testing of time-critical systems

**Aplicando paralelismo a un algoritmo de bisimulación para mejorar la eficiencia de pruebas de software de sistemas de tiempo crítico**

Cómo citar: Betancourt, J.S., Aranda, J., Ortiz, J. Applyn parallelism to a bisimulation algorithm to improve efficiency in software testing of time-critical systems. Ingeniería y Competitividad. 25(suplemento), e-20713144. doi: 10.25100/iyc.v25iSuplemento.13144

Joan S. Betancourt<sup>1§</sup> , Jesús Aranda<sup>1</sup> , James Ortiz<sup>2</sup> 

<sup>1</sup> Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Cali, Colombia

<sup>2</sup> Université de Namur, Namur, Belgium

§ joan.betancourt@correounivalle.edu.co, jesus.aranda@correounivalle.edu.co, james.ortizvega@unamur.be

Este trabajo está licenciado bajo una licencia internacional Creative Commons Reconocimiento-No Comercial-CompartirIgual4.0.

## Abstract

Time-Critical Systems (TCS) play a crucial role in environments where strict timing constraints are essential to ensure reliability and correctness. Model-Based Mutation Testing (MBMT) is considered a valuable strategy for quality assurance of TCS, but it suffers from the equivalent mutant problem, which is known to increase computational cost and reduce confidence in MBMT. To address this problem, a strong timed bisimulation equivalence (STBE) algorithm can be used when TCS are modeled as Timed Automata (TA). STBE is computationally expensive and can benefit from parallelism. We survey available STBE implementations, identify opportunities to apply parallelism, build an extension that takes advantage of them, and test its effects. The resulting solution is a Java program that receives multiple TAs expressed in UPPAAL format and determines which TAs are equivalent using an STBE implementation such as TimBrCheck or MUTES and process-based parallelism. Compared to existing solutions, our tests show that our proposal is more efficient, reducing the runtimes of STBE by more than half. This could improve the reach, reliability, and effectiveness of MBMT for TCS.

Keywords: Timed Automata, Mutation Testing, Timed Bisimulation, Parallelism, UPPAAL.

## Resumen

Los Sistemas de Tiempo Crítico (STC) son importantes cuando estrictos requerimientos temporales garantizan fiabilidad y correctitud de los sistemas. Las Pruebas de Mutación (PM) son una valiosa técnica para el aseguramiento de la calidad de STCs, pero sufren del Problema de los Mutantes Equivalentes (PME), que incrementa costos de cómputo y reduce la confiabilidad de PM. Para combatir el PME, un algoritmo de Bisimulación Temporizada (BT) puede ser usado si los STC se expresan como Autómatas de Tiempo (AT). BT es computacionalmente costoso, y podría beneficiarse de técnicas de paralelismo. En este trabajo, se analizan implementaciones de BT disponibles en la literatura, se identifican oportunidades para aplicar técnicas de paralelismo, se construye una solución que aprovecha estas oportunidades, y se comprueba su efectividad. La solución construida es un programa en Java que recibe múltiples ATs en formato UPPAAL y determina qué ATs son equivalentes, usando una implementación de BT como TimBrCheck o MUTES y paralelismo basado en procesos. Comparado con las soluciones disponibles previamente, los experimentos muestran que nuestra propuesta es más eficiente, ya que los tiempos de ejecución de BT se reducen a menos de la mitad. Esto podría mejorar el alcance, la confiabilidad y efectividad de PM para STC.

Palabras clave: Autómatas de Tiempo, Pruebas de mutación, Bisimulación con Tiempo, Paralelismo, UPPAAL



## Introduction

Time-Critical Systems (TCS) are systems that must respond in time to external events (11). Examples of TCS include device drivers, coffee makers, communication protocols, and ATMs. The behavior of TCS is typically subject to strict timing constraints. For example, a railroad crossing gate must be closed within a specified time after a train is detected, or cars and pedestrians may be injured. Similarly, a radiation machine must deliver a high dose of radiation to a cancer patient within a specified time, or the patient may not receive the necessary treatment.

The consequences of a TCS failing to meet its timing constraints can be severe. Because of the potential consequences, it is essential that TCS be thoroughly tested to ensure that they meet the timing constraints agreed with the customer. This can be a challenging task, as TCS are often complex and have many interacting components (11). However, ensuring the quality assurance of TCS remains a paramount concern.

To address this problem, Model-Based Testing (MBT) (1) is emerging as a valuable strategy. MBT uses (timed) specifications to generate test cases that systematically evaluate the behavior of a system. Importantly, it avoids the scalability problems associated with exhaustive verification (2). MBT has been used successfully to analyze industry TCS like the UNISIG railway communication standard (28), the SCADA programmable logic controller interface for critical systems (29), the FlexRay automotive communication protocol (30), a cardiac pacemaker (31) or a mechanical ventilator for medical aid (32).

When MBT is applied to TCS, the functional requirements are specified as temporal properties, e.g. "the airbag of the car should be deployed within 50ms when a crash is detected". Therefore, temporal defects of TCS models can be detected by identifying functional failures through MBT. One notable modality of MBT is Model-Based Mutation Testing (MBMT), which provides a robust solution for automatically finding defects associated with both missing functionality and misinterpreted specifications through the application of mutation testing (3). Mutation testing, a technique at the heart of MBMT, involves the deliberate injection of artificial defects into an original system, creating new systems known as mutants (4). By evaluating whether the tests can distinguish these mutants from the original system, we gain greater confidence in the effectiveness of the defect detection tests.

However, not all mutants are relevant. Some mutants may be equivalent, having the behavior of the original system despite syntax variations (5). In such cases, test cases will not be able to distinguish between these mutants. In addition, some mutants may be duplicates, exhibiting the same behavior as other mutants (5). Equivalent and duplicate mutants (i.e., useless mutants) increase costs approximately 20% to 40% (26). Identifying and eliminating such useless mutants reduces the computational cost of MBMT while increasing confidence in the final test results. This task is encapsulated in the Equivalent Mutant Problem (EMP) (6).

To address the EMP, it is essential to compute behavioral equivalence between the original model and its mutants. However, determining equivalence between two temporal systems can be computationally expensive (7). Furthermore, when mutants are generated in large batches, resolving the EMP can become very time-consuming. This challenge un-

derscores why behavioral equivalence has not been widely adopted to address the EMP within MBMT (6). Improving the practicality of this solution could effectively expand the horizons of model-based mutation testing, thereby increasing the caliber and security of critical systems.

Recently, the speed-up of sequential chip design has slowed due to various factors such as power consumption and heat generation. In contrast, parallel hardware continues to improve its computational performance (8). To take advantage of this, solutions must be developed that take advantage of parallel acceleration. In this work, we explore a way to apply parallelization to the computation of behavioral equivalence between TCSs to improve the viability of MBMT solutions.

**Document structure.** Section 2 sets the theoretical context. Then, in section 3, we lay out our plan: We review related work in detail (3a, b), implement a usable prototype solution (3c), and compare the results with other approaches (3d). Section 4 presents the results, detailing the architecture of the created tool (4a) and showing how it is twice as fast as other available solutions (4b).

## Theoretical Framework

### a. Timed Automata (TA)

TAs are used to model the behavior of TCS. They can be thought of as a directed graph, where locations are the states of the system and edges are transitions that specify how the system evolves. Locations and transitions can be enriched with time constraints that specify the conditions for the system to remain in a particular state or transition to a new state. These time constraints use clocks. Clocks model the continuous time domain. Clocks are non-negative real-valued variables that increase synchronously at the same rate and can be reset by a transition. Transitions in TA can be enabled or disabled based on clock constraints.

**Definition 1. (Clock Constraints) (11).** Let  $\mathcal{C}$  be a finite set of clock variables ranging over  $\mathbb{R}^+$  (non-negative real numbers). Let  $\mathcal{C}_c$  be a set of clock constraints over  $\mathcal{C}$ . A clock constraint  $c$  is of the form  $c \leq t$  where  $t \in \mathbb{R}^+$  and  $c \in \mathcal{C}$ .

Conditions to remain at a certain location can also be modeled as clock invariants.

**Definition 2. (Clock Invariants) (11).** Let  $\mathcal{C}$  be a finite set of clock variables ranging over  $\mathbb{R}^+$ . Let  $\mathcal{C}_i$  be a set of clock invariants over  $\mathcal{C}$ . A clock invariant  $i$  is of the form  $i \leq t$  where  $t \in \mathbb{R}^+$  and  $i \in \mathcal{C}$ .

Clocks can be modeled with resets and progress via clock valuations.

**Definition 3. (Clock Valuations) (11).** Given a finite set of clocks  $\mathcal{C}$ , a clock valuation function assigning to each clock  $c$  a value  $v(c) \in \mathbb{R}^+$ . We denote  $\mathcal{V}$  the set of all valuations. For a clock valuation  $v$  and a time value  $t \in \mathbb{R}^+$ ,  $v \models t$  is the valuation satisfied by  $v$  for each  $c \in \mathcal{C}$ . Given a clock subset  $\mathcal{C}'$ , we denote the valuation defined as follows: if  $c \in \mathcal{C}'$  and  $v(c) \leq t$  otherwise.

With these previous concepts, we are ready to define timed automata formally.

**Definition 4. (Timed Automaton) (11).** A timed automaton is a tuple  $(\mathcal{L}, \mathcal{C}, \mathcal{C}_c, \mathcal{C}_i, \mathcal{V}, \mathcal{E})$ , where  $\mathcal{L}$  is a finite set

of locations, is an initial location, is a finite set of clocks, is a finite set of actions, is a finite set of edges between locations, and assigns invariants to locations.

For a transition we will write and call and the source and target location, the guard, the action and the set of clocks to reset. TAs will be our formalism of choice to model TCS. Figure 1 depicts an example of TA that models a simplified coffee machine, which starts in the “idle” state until a button is pressed. When the “press” action occurs, clock  $x$  is reset to zero while the machine jumps to the “warm up” location and remains there while . The “Coffee” action can only be performed when the clock is and the system goes to the “Fill Cup” location. The “Sugar” action resets clock , can only be performed when , and returns the machine to its initial idle state.

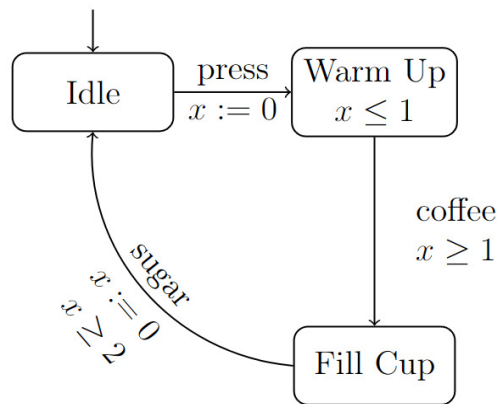


Figure 1: Example of TA modeling: A coffee machine.

The semantics of a TA are carried by a Timed Transition System (TTS) where a state is a pair  $(l, v)$ , where  $l$  denotes the current location with its accompanying clock valuation  $v$ , starting at  $v_0$  where  $v_0$  maps each clock to 0. The transitions can be of two types: Delay transitions and discrete transitions. Delay transitions only let time pass without changing location. Discrete transitions occur instead between a source and a target location. The transition can only happen if the current clock values satisfy both the guard of the transition and the invariant of the target location.

**Definition 5. (Semantics of TA) (11).** Let  $A$  be a TA. The semantics of TA  $A$  are given by a TTS  $(S, s_0, \rightarrow)$  where  $S$  is a set of states, with  $s_0$  for all  $c \in C$  and  $v_0(c) = 0$ , and  $\rightarrow$  is a transition relation defined by two rules: Discrete transitions for  $(l, v) \rightarrow (l', v')$  iff  $(l, v) \xrightarrow{a, g} l', v'$  and  $(l', v') \models I_{l'}$  and delay transitions  $(l, v) \rightarrow (l, v')$  for some  $t \geq 0$  iff  $(l, v) \models I_l$  and  $v' = v + t$ .

## b. Strong Timed Bisimulation Equivalence (STBE)

Bisimulation equivalences identify automata that have the same branching behavior, simulating each other's steps (11). One bisimulation equivalence is the Strong Timed Bisimulation Equivalence (STBE).

**Definition 6. Strong Timed Bisimulation Equivalence (STBE) (11).** Let  $A, B$  be two TTS over a common set of actions  $\Sigma$ . A timed bisimulation for  $A, B$  is a binary relation  $\sim$  where  $(l, v) \sim (l', v')$  implies two

things: First, for every discrete transition with , there exists a matching transition such that and vice versa. Second, for every delay transition with , there exists a transition such that and vice versa.

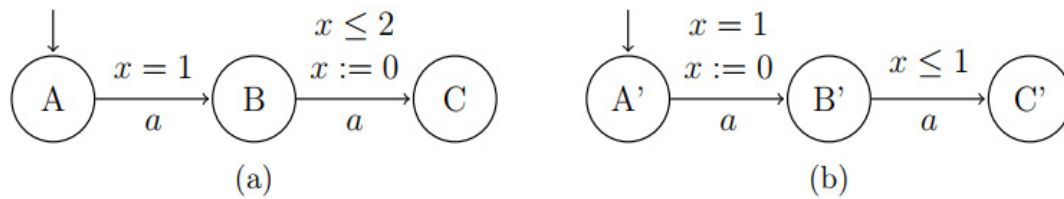


Figure 2: Two TAs which are strongly bisimilar

Two TAs and are equivalent under STBE if there is a timed bisimulation relation over and such that their initial locations are related. In the example of Figure 2, the two TAs are strongly timed bisimilar. Intuitively, they “behave similarly”: Starting from the initial locations, every transition one TA takes can be mirrored by the other. The clocks might not advance in the same way, but actions have the same relative time span: Action from location to can be taken in the same exact way from and , and action from to can only be done within the next unit of time, just as action from to .

To compute whether two TAs are equivalent under STBE, complex operations must be performed in order to operate over the continuous time domain (7). Nevertheless, STBE is one of the few behavioral equivalences that is decidable (6). If two TCS expressed as TA are equivalent under STBE, they show the same behavior, and thus they should be equivalent regarding tests in MBMT (12), despite their potential syntactic differences. Therefore, it will be our equivalence relation of choice for this work. Where clarity is not compromised, STBE may be referred to as timed bisimulation, or simply the equivalence.

### c. UPPAAL

UPPAAL (13) is an integrated environment popular for modeling, validation, and verification of real-time systems. UPPAAL was designed for modeling non-deterministic processes with finite control structures and real-valued clocks, communicating through channels and shared data structures. In other words, it allows modeling TCS with data types and structures, saving them in an XML-based format. The UPPAAL model-checker is built upon the theory of TA (13). UPPAAL is a common choice to apply MBMT to TCS, for example, an automotive gear controller (33), wire brake (34), car alarm (35) and web service (36) UPPAAL models have been tested with MBMT. For this reason, UPPAAL is our chosen operational format to express TAs. Figure 3 is an example of a TCS in UPPAAL, a model of the previous coffee machine.

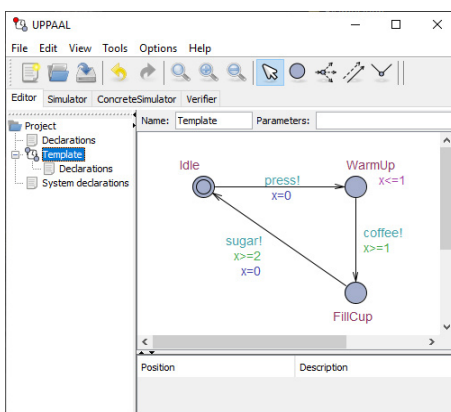


Figure 3: A simplified model of a coffee machine modeled in UPPAAL.

## Methodology

In an attempt to help with the burden of EMP in MBMT, we aim to apply parallelism to the identification of equivalent TCS expressed as TA. More specifically, we want to explore the parallelization potential of deciding STBE between several UPPAAL models. To achieve this, we will:

- Step 1. Identify the approaches to STBE currently available in the literature,
- Step 2. Determine opportunities to apply parallelism or other improvements in the internal operations carried out by these.
- Step 3. Build a solution that exploits those opportunities to compute STBE between UPPAAL models.
- Step 4. Run experiments to compare our approach to other possible solutions.

### d. Step 1: Literature review and state-of-the-art survey for STBE solutions

Timed bisimulation was shown to be decidable by characterizing TA semantics on a finite representation called region graphs (13). This, however, was just a theoretical tool that was not viable for automatic verification. Later, Weise and Lenkes proposed the use of zone graphs, a representation that uses less space (14). With these advancements, automatic verification of timed bisimulation has been attempted:

1. In the early 2000s, the Concurrency Workbench of the New Century (15) supported timed bisimulation, but it is no longer accessible or usable.
2. In 2015, Andersen et al. released CAAL, a web suite for educational purposes that models TCS with a similar formalism to TA. CAAL includes a bisimulation equivalence (16).
3. In 2017, Ortiz and colleagues (10) implemented MUTES, a STBE checker that accepts UPPAAL models as inputs.

4. In 2019, TimBrCheck (9) was made available. This is also a STBE checker for UPPAAL models, with the added value of zone-history graphs, a representation proposal that claims to have better accuracy.

Deciding bisimilarity for different formalisms has been thoroughly investigated in the literature. Kanellakis and Smolka (17) proposed an algorithm with quadratic complexity. Paige and Tarjan (18) found an alternative that runs in linearithmic time. In May 2021, Martens and colleagues (19) proposed a parallel algorithm that verifies bisimilarity for (untimed) automata in linear time by using partition refinement and GPU acceleration. To the best of our knowledge, no parallel application to STBE has been published yet.

### e. Step 2: Review of the internal operations needed for STBE

After surveying available STBE implementations, we chose to continue our work by analyzing MUTES and TimBrCheck, which are the two options that suit our context of TA and UPPAAL.

As MUTES (10) is a product of the work of one of the authors, we had direct access to first-hand details and source code. MUTES is a Java application that processes two XML files as input. ANTLR, a parser generator (20), is utilized to parse the XML files generated by UPPAAL. These XML files contain information about clocks, transitions, invariants, guards, and the structure of the TA. Upon successful parsing of the XML input files representing two TA models, the following steps are executed:

- 1) **Parallel Composition:** MUTES generates a new TA, which is a parallel composition of two individual TAs.
- 2) **Creation of Symbolic Timed Zone Graph:** The tool proceeds to build a finite symbolic Zone Graph (ZG) based on the generated parallel composition. This graph serves as a representation of the potential states and transitions of the composed automaton.
- 3) **Refinement Algorithm for Timed Bisimulation:** MUTES performs a refinement algorithm on the ZG. This algorithm is employed to rigorously examine and verify the concept of timed bisimulation between the two original TAs.

Figure 4 provides an overall visual representation of the MUTES tool and its functionality in these stages.

TimBrCheck (9) is another Java application that also accepts two UPPAAL models as input. This application builds upon ZG to achieve more accuracy. ZGs are enriched to become Zone-History Graphs (ZHG). ZHG are very complex to handle and require the third-party IBM ILOG CPLEX Optimizer to run (36).



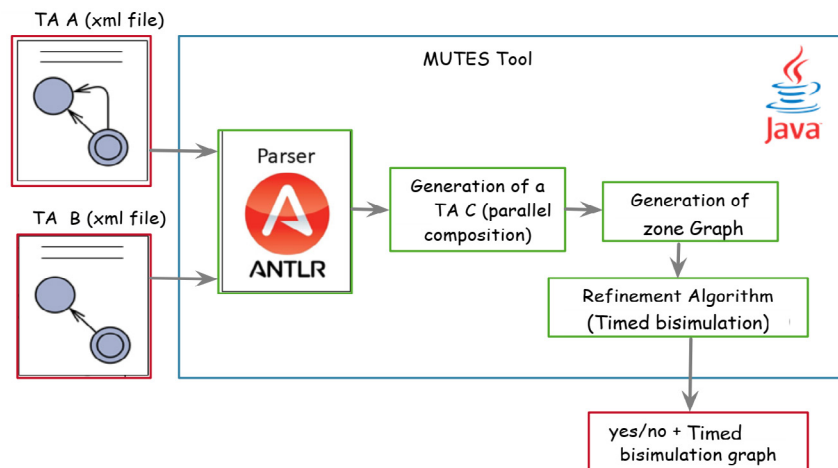


Figure 4: General software architecture of MUTES

After reviewing these two approaches, we found the following opportunities for improvement:

- OP1. Both MUTES and TimBrCheck decide STBE between two TAs at a time. Because mutants come in batches of TA, we want to know, for each TA of the batch, to which other TAs it is equivalent to. Therefore, we need to compute STBE with each TA against all others to compare them in a similar fashion to round-robin tournaments. Because each equivalence is independent, we can run each comparison in parallel.
- OP2. STBE is a transitive relation. This property can be used to infer bisimilarities based on already-known results and save up computing time.
- OP3. MUTES is simpler in terms of theoretical complexity. However, its performance could be refined by auditing its code implementation. Our investigation detailed in (21) suggests room for improvement based on observations of memory usage and logging output.
- OP4. Because ZG bisimilarity is computed with a refinement algorithm similar to untimed bisimulation, previously surveyed techniques of parallelism, like GPU acceleration of (19), may be applied to that step.

We will focus mainly on opportunities OP1 to OP3. Opportunity OP4 was explored in (21) and remains a promising future work.

### f. Step 3: Building a solution

Developing software that takes a set of TAs in UPPAAL format and efficiently computes the partition induced by the STBE relation will be our goal. A partition is a set of sets whose union is the original input set. Each set of the partition is called an equivalence class. Each TA in an equivalence class is equivalent to every other member of the same class.

Based on the previously shown opportunities, the work will be twofold:

- A. Build a program that receives the set of TAs and schedules a MUTES or TimBrCheck run for every pair in the set. These runs will be concurrent, exploiting opportunity OP1.

For OP2, the program should be able to infer equivalences from previous results. To see this, imagine an input set that consists of TAs A, B, and C. At the beginning of the operation, the equivalence between A and B is decided as true, and then equivalence A vs. C is computed as also true. In this case, the software should be able to infer, without any further STBE computation, that B and C are equivalent too. This feature will be referred to as Transitivity Inference (TI).

- B. MUTES source code will be audited. This comes from an inspection of performance detailed in (21).

### g. Step 4: Comparison between approaches

To measure the impact of our work, we need to determine the performance of the resulting solution. The performance will be evaluated under the previously defined use case of computing equivalence between mutants. This will be accomplished by running the following experiment: Three TAs that describe real-world systems will be mutated with various operators to produce a pool of mutants. Five samples of ten mutants will be picked from the pool, and then every sample will be submitted as input to the available solutions five times. The total duration of the runs will be timed. The fastest solution should be the best for our use case.

Considered solutions. In this experiment, four competing solutions will be featured:

1. **Approach SM: Sequential MUTES.** This solution consists of scheduling equivalences one after the other by running MUTES once, waiting for a result, and, once it is finished, running the next one. MUTES will be called once at a time for every distinct pair of TA in the input set. This solution was the one available before the present work was carried out and will serve as control.
2. **Approach ST: Sequential TimBrCheck.** Same setting as approach A, but makes use of TimBrCheck. TimBrCheck uses a bound parameter to limit the resources used

by the ZHG mechanism (9). Because its competitor MUTES do not make use of this mechanism, the parameter will be set, disabling the histories component, so that the results are a fair comparison.

3. **Approach PM2: Parallel MUTES v2.** This solution will take advantage of the proposed work in the previous section. A program will receive a set of TA and schedule parallel executions of the audited and improved version of MUTES, called MUTES v2. On top of that, the TI feature will be active to infer equivalences. This approach will reflect the effectiveness of our work.
4. **Approach PT: Parallel TimBrCheck.** This solution will also use the program that schedules runs in parallel and is able to deduce equivalences a priori with TI, but this time TimBrCheck will be used as STBE implementation, with .

Comparison metrics. To improve efficiency, we are trying to answer the following research question:

**Speed criterion.** How much time does the solution need in order to decide if two automata are timed bisimilar or not?

To measure this criterion, Average Comparison Time (ACT) will be used. This is the ratio in seconds of total execution time divided by the number of timed bisimulations successfully computed. For example, imagine a run of approach SM that received a set of 10 automata as input and lasted 90 seconds before producing the partition induced by STBE. All possible distinct pairs in a set of 10 automata is , therefore 45 equivalences need to be computed. Then, this run would yield of ACT. The choice of this measurement is justified by the use case of MBMT with big sets of mutants: The total time to process the whole set can be estimated by knowing the number of mutants and multiplying the ACT and the number of distinct pairs.

Subject systems. We will consider three different TA models taken from community benchmarks, frequently used in recent experimental evaluation of TCS analysis techniques:

1. Gear Controller (GC) (22): Component of the control system operating in a modern vehicle.
2. Collision Avoidance (CA) (23): Protocol of communication for an Ethernet-like medium.
3. Combined Gear Control (CGC) (24): Combined manual gear control for vehicles.

Table 1 provides an overview of key properties of the models to have a sense of their magnitudes.

Table 1. Key properties of subject TA for comparison experiments

<b>Property</b>	<b>GC</b>	<b>CA</b>	<b>CGC</b>
<b>Locations</b>	24	7	85
<b>Transitions</b>	32	15	120
<b>Clocks</b>	1	1	4
<b>Resets</b>	12	2	31

These models will be mutated with the following operators, most of them proposed in (25):

1. Transition MIssing (TMI) removes a random transition.
2. Transition ADd (TAD) adds a transition between two random locations.
3. State MIssing (SMI) removes an arbitrary location other than the initial and all its transitions.
4. State MIssing No Redundant (SMI-NR) Removes an arbitrary location, avoiding duplicates (26).
5. Constant eXchange L (CXL) increases the constant of a clock constraint.
6. Constant eXchange C (CXS) decreases the constant of a clock constraint.
7. Clock Constraint Negation (CCN) negates a clock constraint.

These mutants and systems were taken from Cuartas et al. work (26). Table 2 shows the number of mutants generated per operator and system. These quantities depended on the properties of the base model.

Table 2. Number of generated mutants per operator

<b>System</b>	<b>TMI</b>	<b>TAD</b>	<b>SMI</b>	<b>SMI-NR</b>	<b>CXL</b>	<b>CXS</b>	<b>CCN</b>	<b>Total</b>
<b>GC</b>	13	501	12	3	0	2	2	533
<b>CA</b>	9	26	2	0	1	1	2	41
<b>CGC</b>	36	1625	27	5	4	46	10	1.713

From the set of mutants of each system, five samples of 10 mutants will be randomly picked. Each sample will be submitted as input to the four approaches five times. To measure the total execution time, two timestamps are stored: one at the beginning, right after the input TAs in XML files are found, and another at the end, after the result output

is printed, when all OS processes of TimBrCheck or MUTES are terminated. The result of the difference between these two timestamps is the total execution time. These timestamps are taken by executing **System.currentTimeMillis()**, a Java method that returns the current time with a granularity of milliseconds, and then rounding up to seconds. Because MBMT is not a time-critical process, but a process which time-critical processes go through in the software design phase, we believe this time measuring method is enough for our purposes.

After that, the total elapsed time will be divided by the 45 equivalences that need to be decided between the ten systems, and the resulting quotient will be the ACT. The runtime of each execution will be limited to 15 minutes.

Execution environment. The experiment was executed with the OpenJDK JRE 11.0.17 on Windows 10 with 16 GB of memory and an Intel(R) Core (TM) i3-12100F with 8 cores at 3.30GHz.

## Results and discussion

### h. Building of a parallel solution

The resulting software product is called ThreadsBash. ThreadsBash is a Java application that can be executed with a Command Line Interface (CLI) command. An execution command may be **java -jar threadsbash.jar -E JARPATH -F FOLDERPATH**, where **JARPATH** is the path to an STBE implementation (e.g. MUTES or TimBrCheck) and **FOLDERPATH** is a path to a folder that should contain UPPAAL XML models. Other flags are also available to tune other variables for execution. The project can be accessed via GitHub <https://github.com/SebastianBetancourt/threadsbash>. Figure 5 lays out its general workflow architecture.

ThreadsBash implements Process-based parallelism. Process-based parallelism creates different processes at the Operative System (OS) level and lets the OS manage concurrent resources. ThreadsBash accomplishes this by using the ProcessBuilder class to implement a Runnable that can be executed by Java's ThreadPoolExecutor, a concurrency mechanism provided by the Java Virtual Machine (27). ProcessBuilder is instantiated by passing a CLI command as a parameter, which in this case would be the command to run the STBE program that will be used to spawn a process at OS level. Because this class implements the interface Runnable, a ThreadPoolExecutor can receive the object and assign the task of executing it to one of

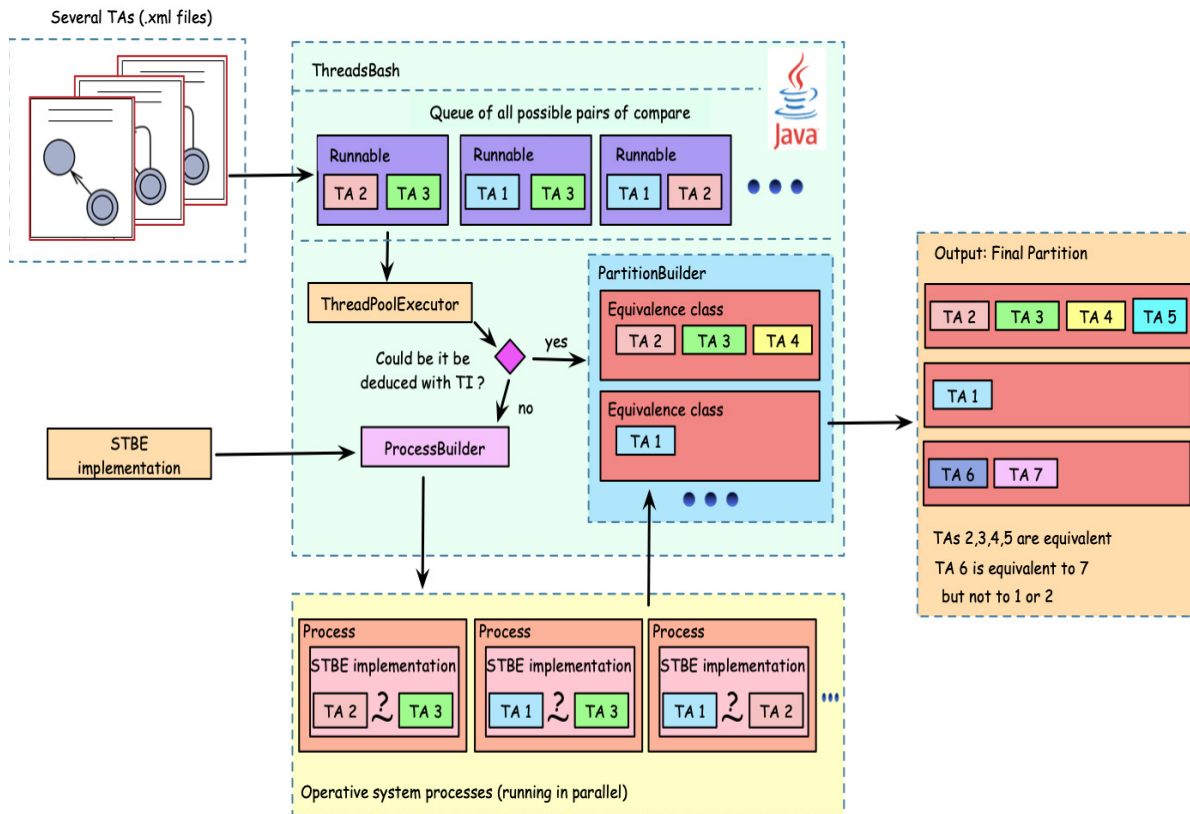


Figure 5: General software architecture of ThreadsBash

multiple concurrent threads, which in turn will be in charge of initiating the comparison, effectively starting multiple parallel OS processes that check bisimulation between different automata. The concurrency of this technique is bounded by the number of parallel processes a processor and OS might be able to carry. This parallel mechanism is the result of exploiting opportunity OP1.

ThreadsBash also has the TI feature enabled by default. This means that ThreadsBash can infer equivalence before it computes, depending on previous results. It does so by storing results as a partition. This partition can be built with incoming STBE results by performing the following checks before starting a comparison between two TAs A and B:

- 1) If  $A$  and  $B$  belong to the same equivalence class, then it can be inferred that  $A$  and  $B$  are equivalent.
- 2) If any member of the equivalence class of  $A$  was previously compared with any member of the equivalence class of  $B$ , and it resulted in they not being equivalent, then it can be inferred that  $A$  and  $B$  are not equivalent either.
- 3) If neither of these two cases holds, then the STBE must be carried out, but its result may contribute to constructing the partition further. If  $A$  and  $B$  turn out to be bisimilar, their equivalence classes may be merged. If they are not bisimilar, the result may also be stored for future inferences of step 2.

The details and pseudocode of this algorithm can be found in (21). This procedure takes advantage of OP2.

OP3 was capitalized on by conducting an inspection and refactoring of the MUTES source code, which is detailed in (21). Logging was muted, and the resulting MUTES v2 shows a remarkable improvement in performance. This is evidenced in the results of the experiments.

### i. Results of the experiment

The experiment designed for step 4 (section 3d) was conducted using 8 concurrent threads. To accomplish the promised parallelism, MUTES v2 and TimBrCheck were equipped with ThreadsBash, setting up approaches PM2 and PT, respectively. The results of this experiment are shown in Figure 6. For three TCS, mutant samples of 10 automata were generated, compared using one of the proposed alternatives, and the ACT was plotted in the y-axis. For every approach and system, each box plots the 25 ACTs produced by processing five different samples five times.

In general, the results are consistent in the sense that they do not vary much between tries or samples. The size of the TA seems to have an impact on ACT, as seen with the results of system CGC, which is orders of magnitude bigger than the other two. This validates the theoretical time complexity of the STBE operation that depends on the number of locations and transitions (21).

Approach SM had problems with the system CGC and its mutants. That was the only case where executions timed out, totaling 20 runs that exceeded the 15-minute limit. We believe that the first version of MUTES, which is the one used in approach SM, had performance problems that would not let it successfully finish in some cases. These problems were solved for MUTES v2, which was included in approach PM2. Even though timed-out runs were not included in the plotted data, the five runs of SM which successfully finished yielded a very high ACT, which further discards the approach.

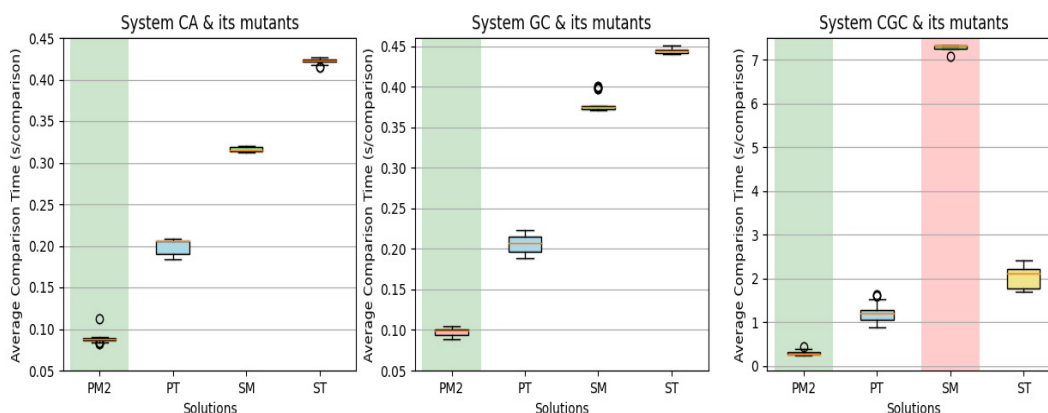


Figure 6. Average Comparison Time of four approaches to computing bisimilarity of three timed systems and its mutants. Lower is better.

Applying parallelism proved to be valuable. This is reflected by the halving of ACT between ST and its parallel counterpart PT and sequential SM, regarding PM2.

However, this experiment leaves a clear winner highlighted in green: Approach PM2, which is the proposed solution backed by our work. Even when TimBrCheck is augmented with parallel processing in approach PT, the results of PM2 are less than a half of the ACT of every other approach.

Given the multiple factors of the computing platform and execution environment that may introduce big variations in the measurements, we decided to evaluate the robustness and scalability of the proposed solution by executing additional tests. This is detailed in (21), where 45 experiments were run on three different computers, two different operating systems and inputs that vary between 8 to almost 2000 TA, using different configurations and numbers of parallel processors. In those experiments, the results presented remain true: At worst, PM2 is twice as fast, and at best, six times faster than the other alternatives. With the addition of these results, we are confident that the improvements can be applied to larger scenarios, because the parallelization technique and problem at hand are suitable to scale in concurrency: The STBE processes are completely independent, and therefore no race condition, deadlocks or resource starvation is foreseen. In fact, this property could be further exploited with other alternatives, such as GPU parallelization or distributed computing, alternatives that were considered in (21) but remain as future work.

One thing to investigate may be the ZHG mechanism that the authors of TimBrCheck propose, which they say is necessary to have a correct algorithm (9). This would imply differences in precision between TimBrCheck and MUTES that may have an impact on the choice of implementation of STBE to be used in MBMT. In this experiment, in particular, the output partitions were the same between the four approaches. Nevertheless, we believe this poses a threat to validity which we are currently looking into.

## Conclusions

This article introduces a tool that helps to solve EMP when using UPPAAL models for MBMT of TCS. ThreadsBash is a product that can augment available STBE implementations with parallelism to determine equivalences between large sets of mutants, and then the ones that are equivalent can be pruned. This can be done efficiently with ThreadsBash and MUTES v2, as seen in the results of the experiments, where timed bisimulation equivalences are decided two times faster than previously available solutions, accounting for environmental factors like configuration and concurrency and input properties like size and variety of TA. In fact, our solution is already being used in other scientific work. This makes the application of MBMT more viable for adoption in industries that rely on quality assurance of systems such as TCS.

## Acknowledgments

This article was a result of Joan S. Betancourt bachelor's thesis work, directed by professors Jesús Aranda and James Ortiz (21). Special thanks to the Erasmus+ multilateral agreement between the University of Namur and Universidad del Valle, which made the collaboration between the two institutions possible.





## References

- (1) Tretmans, J. Model Based Testing with Labeled Transition Systems. In: Formal Methods and Testing – An Outcome of the FORTEST Network. Berlin: Springer-Verlag, 2008. p. 1–38.
- (2) Zander, J, Schieferdecker, I. Model-Based Testing for Embedded Systems. CRC Press; 2017.
- (3) Budd, TA, Gopal, AS. "Program testing by specification mutation". Computer Languages, 1985.
- (4) Jia, Y, Harman, M. "An Analysis and Survey of the Development of Mutation Testing". IEEE Transactions on Software Engineering 2011; 37(5):649-678.
- (5) Papadakis, M, Kintis, M, Zhang, J, Yue, Traon, Y, Harman, M. Chapter Six - Mutation Testing Advances: An Analysis and Survey. In: Advances in Computers, Volume 112, 2019; 112:275-378.
- (6) Ortiz Vega, J, Perrouin, G, Amrani, M, Schobbens, PY. Model-Based Mutation Operators for Timed Systems: A Taxonomy and Research Agenda. In 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS) 2018 (pp. 325-332).
- (7) Guha, S. On Decidability of Prebisimulation for Timed Automata. In Computer Aided Verification, 2012 (pp. 444–461). Springer Berlin Heidelberg.
- (8) Leiserson, CE, Thompson, NC, Emer, JS, Kuszmaul, BC, Lampson, BW, Sanchez, D, Schardl, TB. "There's plenty of room at the Top: What will drive computer performance after Moore's law?". Science 2020; 368(6495):eaam9744.
- (9) Luthmann, L, Göttmann, H, Bacher, I, Lochau, M. Checking Timed Bisimulation with Bounded Zone-History Graphs – Technical Report. 2019. Available at: <https://arxiv.org/abs/1910.08992>
- (10) Ortiz Vega, J, Amrani, M, Schobbens, PY. Multi-timed Bisimulation for Distributed Timed Automata. In NASA Formal Methods, 2017 (pp. 52–67). Springer International Publishing.
- (11) Baier, C, Katoen, JP. Principles of Model Checking. The MIT Press; 2008. pp. 673-738
- (12) Čerāns, K. Decidability of bisimulation equivalences for parallel timer processes. In Computer Aided Verification, 1993 (pp. 302–315). Springer Berlin Heidelberg.
- (13) UPPAAL [Internet]. Uppsala Universitet. Available at: <https://uppaal.org/>
- (14) Weise, C, Lenzen, D. Efficient scaling-invariant checking of timed bisimulation. In STACS 97 1997 (pp. 177–188). Springer Berlin Heidelberg.
- (15) Concurrency Workbench of the New Century [Internet]. North Carolina State University; [cited Aug 13 2023]. Available at: <https://www3.cs.stonybrook.edu/~cwb/>



- (16) Andersen, J, Hansen, M, Andersen, N. CAAL [Internet]. Aalborg University; [cited Aug 13 2023]. Available at: [http://caal.cs.aau.dk/docs/CAAL2\\_EPG.pdf](http://caal.cs.aau.dk/docs/CAAL2_EPG.pdf)
- (17) Kanellakis P. Smolka S. "CCS expressions, finite state processes, and three problems of equivalence". Information and Computation, 1990.
- (18) R. Paige, R. Tarjan. "Three partition refinement algorithms". SIAM Journal on Computing 1987.
- (19) Martens J, Groote, JF, van den Haak, LB, Hijma, P, Wijs, A. "A linear parallel algorithm to compute bisimulation and relational coarsest partitions". CoRR 2021; abs/2105.11788.
- (20) ANTRL [Internet]. Available at: <https://www.antlr.org/>
- (21) Betancourt JS. A parallel algorithm to compute strong timed bisimulation [bachelor's thesis]. Cali: Universidad del Valle; 2023 [cited Aug 10 2023]. Available at: <https://drive.google.com/file/d/1zxDyAf-4qXV0tMry8MkbTThjzJKBUr5b/view?usp=sharing>
- (22) Lindahl, W. Formal design and analysis of a gear controller. In Tools and Algorithms for the Construction and Analysis of Systems, 1998 (pp. 281–297). Springer Berlin Heidelberg.
- (23) Jensen, H, Larsen, K, Skou, A. "Modelling and Analysis of a Collision Avoidance Protocol using SPIN and UPPAAL". BRICS Report Series 2002; 3.
- (24) Lindahl, M, Tettersson, P, Yi W. Formal design and analysis of a gear controller. In Tools and Algorithms for the Construction and Analysis of Systems, 1998. Springer Berlin Heidelberg.
- (25) Basile, D, Beek, M, Cordy, M, Legay, A. Tackling the Equivalent Mutant Problem in Real-Time Systems: The 12 Commandments of Model-Based Mutation Testing. In Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A 2020. Association for Computing Machinery.
- (26) Cuartas, J, Aranda, J, Cordy, M, Ortiz, J, Perrouin, G, Schobbens, PY. MUPPAAL: Reducing and Removing Equivalent and Duplicate Mutants in UPPAAL. In 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2023.
- (27) Oracle. Thread (Java Platform SE7). [Cited Aug 14 2023] Available at <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>
- (28) Basile D, Alessandro Fantechi, Rosadi I. Formal Analysis of the UNISIG Safety Application Intermediate Sub-layer. Lecture Notes in Computer Science. 2021 Jan 1;174–90.
- (29) Mercaldo, F, Martinelli, F, & Santone, A. Real-Time SCADA Attack Detection by Means of Formal Methods. In 2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) (pp. 231-236).

- (30) Guo, X, Lin, HH, Kenro, Yatake. An UPPAAL Framework for Model Checking Automotive Systems with FlexRay Protocol. Communications in computer and information science. 2014.
- (31) Pajic, M, Jiang, Z, Lee, I, Sokolsky, O, Mangharam, R. From Verification to Implementation: A Model Translation Tool and a Pacemaker Case Study. IEEE 18th Real Time and Embedded Technology and Applications Symposium 2012.
- (32) Cuartas, J, Cortés, D, Betancourt, J, Aranda, J, García, J, Valencia, A, Ortiz, J. Formal Verification of a Mechanical Ventilator Using UPPAAL. In Proceedings of the 9th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems 2023 (pp. 2–13). Association for Computing Machinery.
- (33) Lorber, F, Larsen, K, Nielsen, B. Model-Based Mutation Testing of Real-Time Systems via Model Checking. In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2018 (pp. 59–68).
- (34) Larsson, J. Automatic Test Generation and Mutation Analysis using UPPAAL SMC [Thesis]. Marinescu R, editor. [Mälardalen University, School of Innovation, Design and Engineering]; 2017.
- (35) Siavashi, J. Testing Web Services with Model-Based Mutation. In Software Technologies 2017 (pp. 45–67). Springer International Publishing.
- (36) Mathematical program solvers - IBM CPLEX [Internet]. Ibm.com. Available from: <https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer>

## 1. Table of abbreviations

Abbreviation	Meaning
ACT	Average Comparison Time
CGC, GC, CA	Subject systems
CLI	Command Line Interface
EMP	Equivalent Mutant Problem
MBMT	Model-Based Mutation Testing
MBT	Model-Based Testing
OS	Operative System
SMI, SMI-NR, TMI, TAD, CXL, CXS, CCN	Mutation operators
SM, ST, PM2, PT	Considered solutions
STBE	Strong Timed Bisimulation Equivalence
TA	Timed Automata
TCS	Time-Critical Systems
TI	Transitivity Inference
TTS	Timed Transition System
ZG	Zone Graph
ZHG	Zone-History Graph