*International Journal on Recent and Innovation Trends in Computing and Communication*
***ISSN: 2321-8169 Volume: 12 Issue: 2***
***Article Received:** 25 November2023 **Revised:** 12 December 2023 **Accepted:** 30 January 2024*

# Comparing Two Inclusion Techniques in Timed Automata

**Hafedh Mahmoud Zayani**
Department of Electrical Engineering, College of Engineering,
Northern Border University, (corresponding author)
Arar, Saudi Arabia,
hafedh.zayani@nbu.edu.sa

**Refka Ghodhbani, Taoufik Saidani**
Department of Computer Sciences Faculty of Computing and Information Technology,
Northern Border University,
Rafha, Saudi Arabia,
Refka.Ghodhbani@nbu.edu.sa, Taoufik.Saidan@nbu.edu.sa

**Abstract**— Verifying the correctness of real-time systems often involves checking language inclusion between timed automata. This problem determines if the language of a system implementation is a subset of the language specified by its design. While the general case is undecidable, recent advancements have proposed techniques for specific scenarios. This paper compares two such techniques: a zone-based semi-algorithm for non-Zeno runs and a time-bounded discretization approach. We analyze their strengths and weaknesses, highlighting cases where each method is advantageous. The comparison highlights the timed bounded discretized language approach's advantages in terms of guaranteed termination and lower memory usage.

**Keywords**-Timed Automata; Inclusion; Verification; Comparaison

## I. Introduction

Timed automata are a powerful formalism for modeling real-time systems, where behavior is governed by both discrete actions and continuous time passage. A crucial aspect of real-time system verification involves ensuring the implementation adheres to the specified behavior. This translates to the problem of language inclusion checking: determining if all legal execution sequences (language) of the implementation automaton are also valid sequences in the specification automaton.

The general language inclusion problem for timed automata is known to be undecidable [1], posing a significant challenge for verification. However, research efforts have focused on developing techniques for specific scenarios. This paper explores two recent advancements in language inclusion checking for timed automata.

Researchers have actively explored methods for language inclusion checking in timed automata, a crucial step in verifying if an implementation's behavior aligns with a specified property. To address the challenge of non-Zeno runs causing misleading results, Wang et al. [2] proposed a zone-based semi-algorithm that leverages zones (sets of states with time constraints) for efficient state space exploration and violation identification. This approach is further enhanced through simulation reduction. On the other hand, the general problem's undecidability is tackled by Ammar et al. [3] with a time-bounded verification framework. Their technique utilizes a novel discretization approach to represent timed words within a finite time window, achieving decidability for non-Zeno timed automata. Beyond language inclusion checking, research in timed automata verification encompasses broader property verification techniques. Alur et al. [4] introduced a framework for model checking timed automata using timed logic, enabling the specification and automated analysis of complex properties in timed systems. Additionally, Tripakis [5] introduced timed testing techniques, where designing test cases explores the timed behavior of a system to reveal inconsistencies.

The concept of language inclusion checking for timed automata has been extensively studied in [2, 4, 5, 6, 7]. The proposed technique in [2] proposes a zone-based semi-algorithm specifically designed to handle non-Zeno runs. On the other hand, the proposed technique in [3] introduces a time-bounded verification framework for the inclusion problem. This technique leverages a novel discretization approach to represent timed words within a bounded time interval, enabling decidability for non-Zeno timed automata.

This paper builds upon these existing works by comparing and contrasting the two inclusion checking techniques. We delve into the details of each approach, analyzing their strengths, limitations, and potential application areas. The goal is to provide a comprehensive understanding of these techniques and guide users in selecting the most suitable approach for their specific verification needs.

This paper is organized as follows. Section II lays the foundation by introducing the specific type of timed automata used in the analysis - the non-Zeno timed automata model. Section III and IV delve into the core of the paper of the reference [2] and [3]. Each section focuses on a specific approach for verifying language inclusion. After introducing the verification approaches, the paper applies both methods to the same example of non-Zeno timed automata. This allows for a direct comparison of their performance in Section V. The comparison focuses on two key aspects: space memory usage, strengths and weaknesses

## II. Notations

This section defines a timed automaton, a formal model used to represent real-time systems. Here's a breakdown of the key components:

- **Clocks** ($X$): A finite set representing the system's clocks. Each clock holds a non-negative rational value $\mathbb{Q}_+$.
- **Clock Constraints** (*C(X)*): The set of formulas expressing relationships between clocks and constants using comparison operators (<, <=, =, >=, >) and constants from positive rationales ($\mathbb{Q}_+$.). These formulas define conditions that must hold for the system to be in a specific state.
- **Clock Valuation** ($v$): A function that assigns a non-negative rational value to each clock.
- **Time Delay** ( $d$ ): A non-negative rational value representing the time passage.
- **Timed Automaton** $\boldsymbol{A} = (L, l_0, X, \Sigma, I, T)$ : A 7-tuple consisting of:
  - ***Locations*** ($L$): A finite set of states the system can be in.
  - ***Initial Location*** ($l_0$): The starting state of the system.
  - ***Clocks*** (X): As defined above. |X|=1
  - ***Actions*** ($\Sigma$): A finite set of events or actions the system can perform.
  - ***Location Invariants*** ($I$): A function that assigns a clock constraint to each location, restricting the allowed clock valuations for the system to be in that location.
  - ***Transitions*** ($T \subseteq L \times C(X) \times \Sigma \times 2^X \times L$): A finite set of transitions between locations. Each transition is a 5-tuple (*l, g, a, r, l'*) representing:
    - *Source Location* ($l$): The origin location of the transition.
    - *Guard* ($g$): A clock constraint that must be true for the transition to occur.
    - *Action* ($a$): The action associated with the transition.
    - *Reset Set* ($r$): A subset of clocks that are reset to zero when the transition is taken.
    - *Target Location* ($l'$): The destination location after the transition.
- **Non-Zenoness**: The definition emphasizes that the timed automaton is non-Zeno. This means there are no execution paths where an infinite number of actions occur within a finite amount of time.
- **States**: A state of a timed automaton is denoted by a pair (*l, v*) where:
  - $l \in L$: Represents the current location of the system.
  - $v \in \mathbb{Q}_+{}^{|X|}$: Represents the current clock valuation, assigning a non-negative rational value to each clock.
- **Transition Run**: A transition run, denoted by $\psi_p = l_0 \xrightarrow{e_0} l_1 \xrightarrow{e_1} l_2 \ldots l_i \xrightarrow{e_i} l_{i+1} \ldots l_{p-1} \xrightarrow{e_p} l_{p+1}$ , represents a sequence of transitions the automaton can take. It's a sequence of elements $\mathrm{e}_0, \mathrm{e}_1, \ldots, \mathrm{e}_i, \mathrm{e}_{i+1} \ldots, \mathrm{e}_p$ where:
  - Each $\mathrm{e}_i$ is a transition from the set $T$ of transitions.
  - The transition $\mathrm{e}_{i+1}$ follows $\mathrm{e}_i$ such that:
    - $\mathrm{e}_i = (\mathrm{l}_i, \mathrm{a}_i, \mathrm{g}_i, \mathrm{r}_i, \mathrm{l'}_i)$ and
    - $\mathrm{e}_{i+1} = (\mathrm{l}_{i+1}, \mathrm{a}_{i+1}, \mathrm{g}_{i+1}, \mathrm{r}_{i+1}, \mathrm{l'}_{i+1})$
    - The target location ($\mathrm{l'}_i$) of $\mathrm{e}_i$ is the source location ($\mathrm{l}_{i+1}$) of $\mathrm{e}_{i+1}$. This ensures a valid sequence of transitions.
    - This condition holds for all *i* between *0* and *p-1*.

This formal definition provides a foundation for understanding and analyzing the behavior of real-time systems using non-Zeno timed automata.

## III. Verification of Inclusion Problem using a Semi-Algoritm

The paper [2] contributes to the field of timed automata verification by addressing language inclusion checking while considering non-zenoness. The authors propose a zone-based semi-algorithm to address language inclusion checking with non-zenoness. This proposed method offers a practical solution even though it may not provide a guaranteed answer in all scenarios. This approach based on the following steps, as shown in Fig. 1.

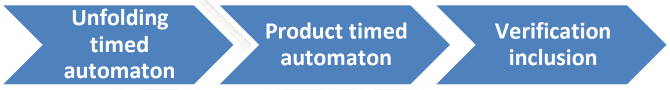

Figure 1. Steps of the timed bounded verification of inclusion in [1].

### A. *Unfolding of a timed automaton*

The first step of the zone-based semi-algorithm for language inclusion checking with non-zenoness involves unfolding the timed automaton. This process creates a tree structure that captures all possible executions of the automaton, while keeping track of timing constraints. Each node represents a possible state along an execution path in the original automaton. The labels on the nodes connect these states back to the original automaton and encode the timing information using the reset clocks. The unfolding step as described as follows:

- **New Clock Resets**: At each level of the tree, the clocks of the original automaton are reset. This ensures that all timing constraints are considered afresh at each step.
- **Node Labeling**: Each node ($n$) in the unfolded automaton B1 is labeled with a pair (*l, z*).
- **Locality (*l*)**: This identifies the specific location in the original automaton A that the current node in the unfolded automaton B1 simulates.
- **Clock Encoding (z):** This describes how the clocks in B are represented using the clocks in the unfolded automaton B1. If z(x) = $z_i$, it implies that the clock value x in B is currently equivalent to the value of clock $z_i$ in B1. This allows for tracking timing constraints across different levels of the tree.

**Example 1:**

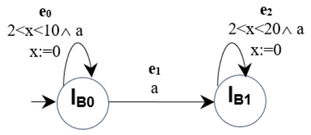

Figure 2. Timed automaton B

Let $\boldsymbol{B} = (L_B, l_{B0}, X, \Sigma, I_B, T_B)$ be a timed automaton, represented in Fig. 2 where $X = \{x\}$ and $\Sigma = \{a\}$ . This automaton describes the possible locations and transitions. The unfolding process takes the timed automaton B and creates a new automaton, typically shown in Fig. 3, which is a tree-like structure. This unfolded automaton captures all the possible execution paths of the original automaton, considering timing constraints.

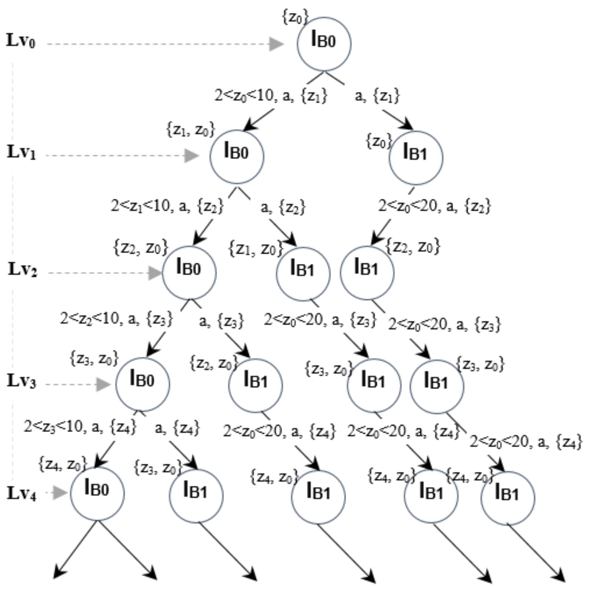

Figure 3. Unfolding of the timed automaton B

## B. *Product-timed automaton*

The second step involves creating a new timed automaton called the product automaton. This product automaton combines the original timed automaton A with the unfolded timed automaton B1. It as a merged structure that captures both the original system's behavior and all potential execution paths revealed through unfolding. The product automaton, denoted by P, is based on the following elements:

- **States**: The product automaton's states are formed by pairing a location from the timed automaton A with a node from the unfolded automaton B1. This combined state essentially tracks both the current position in the original system and the specific point along a possible execution path in the unfolded structure.
- **Transitions**: The product automaton only allows the transition if the corresponding node in the unfolded automaton permits it. This ensures the transition adheres to the local state and timing constraints at that specific point in the potential execution path.
- **Clocks**: The product automaton gathers all the clocks from both the timed automaton A and the unfolded automaton B1. This allows for a comprehensive view of how time constraints influence the combined behavior.

The state of the product automaton is represented by a triplet $(s_a, X_b, \partial)$ where $s_a$ identifies the specific location currently active in the timed automaton A, $X_b$ refers to the particular node from the unfolded automaton B1 and $\partial$ represents the combined clocks constraints from the timed automaton A and any constraints arising from the specific node in the unfolded automaton B1.

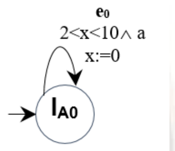

Figure 4. Timed automaton A

**Example 2:**

Let $\boldsymbol{A} = (L_A, l_{A0}, X, \Sigma, I_A, T_A)$ be a timed automaton, shown in Fig. 4 and B1 be an unfolding automaton shown in Fig. 3. The timed automaton A is combined with the unfolded automaton B1 to create the product automaton shown in Fig. 5.

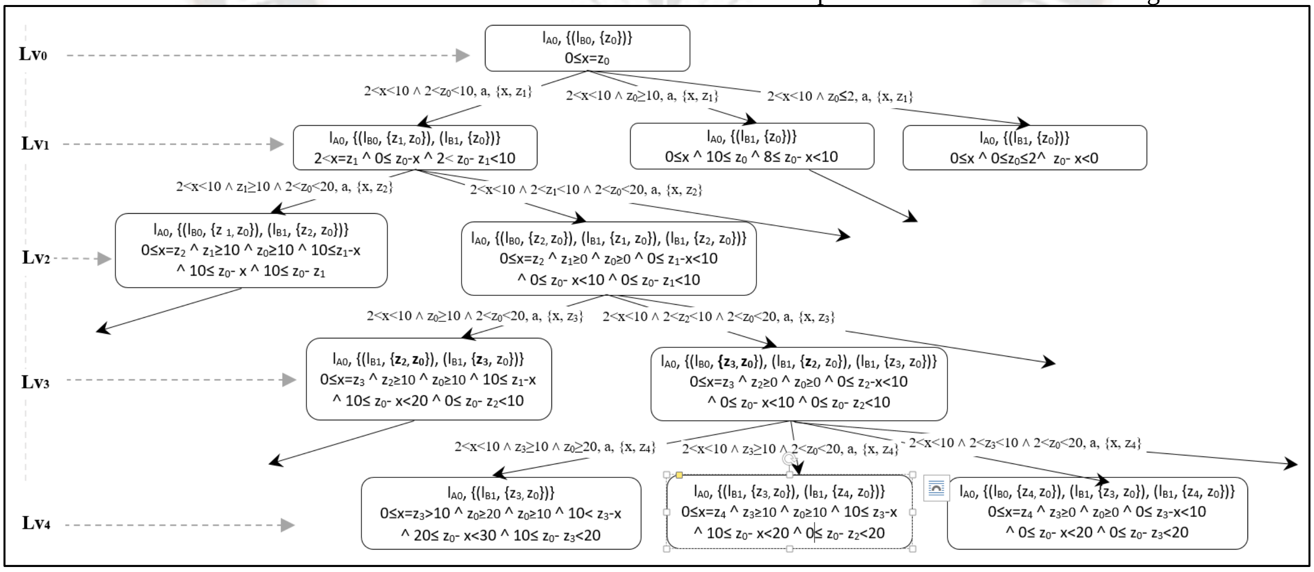
Lv0
Lv1
Lv2
Lv3
Lv4
lA0, {(lB0, {z0})}
0≤x=z0
2<x<10 ∧ 2<z0<10, a, {x, z1}
2<x<10 ∧ z0≥10, a, {x, z1}
2<x<10 ∧ z0≤2, a, {x, z1}

Figure 5. Product automaton of A and B1

Fig. 5 shows three active clocks: $z_0$, $z_2$ and $z_3$ in level 3. The authors propose to reuse $z_1$ because that is not active in level 3 and renaming $z_3$ to $z_1$. This optimization aims to reduce the number of active clocks in the product automaton while maintaining the same behavior. Then, Fig. 6 depicts the resulting tree structure after renaming the clock.

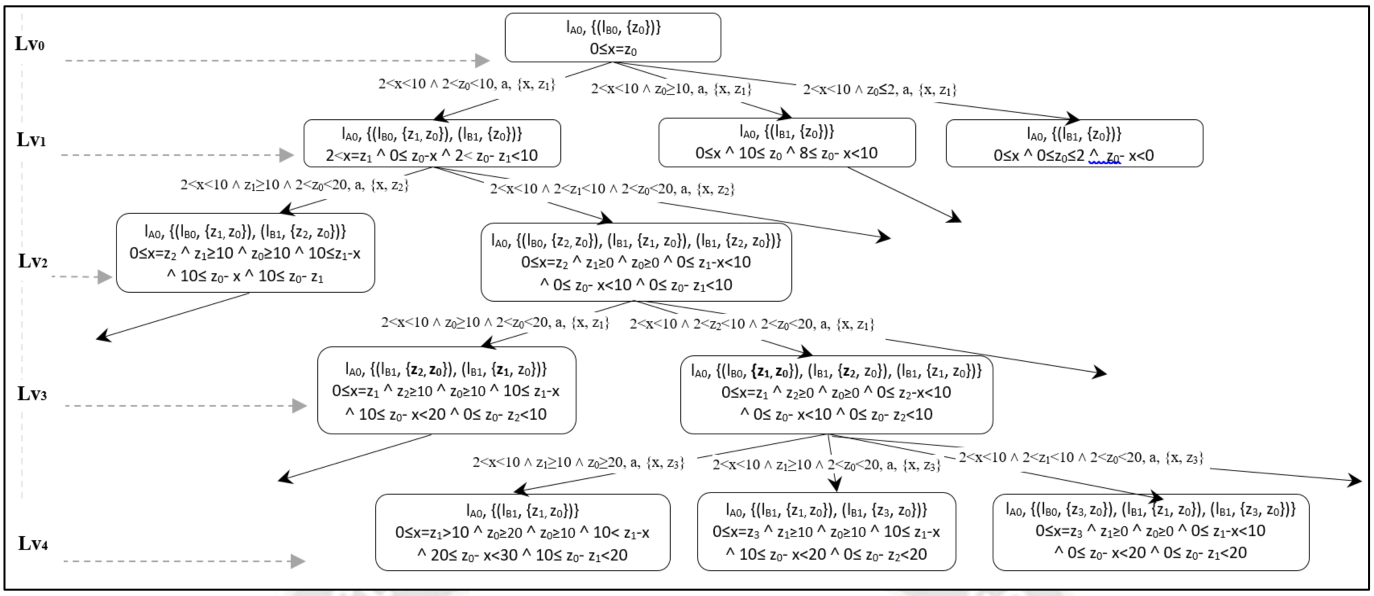
Lv0
lA0, {(lB0, {z0})}
0≤x=z0
2<x<10 ∧ 2<z0<10, a, {x, z1}
2<x<10 ∧ z0≥10, a, {x, z1}
2<x<10 ∧ z0≤2, a, {x, z1}
Lv1
lA0, {(lB0, {z1, z0}), (lB1, {z0})}
2<x=z1 ^ 0≤ z0-x ^ 2< z0- z1<10
lA0, {(lB1, {z0})}
0≤x ^ 10≤ z0 ^ 8≤ z0- x<10
lA0, {(lB1, {z0})}
0≤x ^ 0≤z0≤2 ^ z0- x<0
2<x<10 ∧ z1≥10 ∧ 2<z0<20, a, {x, z2}
2<x<10 ∧ 2<z1<10 ∧ 2<z0<20, a, {x, z2}
Lv2
lA0, {(lB0, {z1, z0}), (lB1, {z2, z0})}
0≤x=z2 ^ z1≥10 ^ z0≥10 ^ 10≤z1-x
^ 10≤ z0- x ^ 10≤ z0- z1
lA0, {(lB0, {z2, z0}), (lB1, {z1, z0}), (lB1, {z2, z0})}
0≤x=z2 ^ z1≥0 ^ z0≥0 ^ 0≤ z1-x<10
^ 0≤ z0- x<10 ^ 0≤ z0- z1<10
2<x<10 ∧ z0≥10 ∧ 2<z0<20, a, {x, z1}
2<x<10 ∧ 2<z2<10 ∧ 2<z0<20, a, {x, z1}
Lv3
lA0, {(lB1, {z2, z0}), (lB1, {z1, z0})}
0≤x=z1 ^ z2≥10 ^ z0≥10 ^ 10≤ z1-x
^ 10≤ z0- x<20 ^ 0≤ z0- z2<10
lA0, {(lB0, {z1, z0}), (lB1, {z2, z0}), (lB1, {z1, z0})}
0≤x=z1 ^ z2≥0 ^ z0≥0 ^ 0≤ z2-x<10
^ 0≤ z0- x<10 ^ 0≤ z0- z2<10
2<x<10 ∧ z1≥10 ∧ z0≥20, a, {x, z3}
2<x<10 ∧ z1≥10 ∧ 2<z0<20, a, {x, z3}
2<x<10 ∧ 2<z1<10 ∧ 2<z0<20, a, {x, z3}
Lv4
lA0, {(lB1, {z1, z0})}
0≤x=z1>10 ^ z0≥20 ^ z0≥10 ^ 10< z1-x
^ 20≤ z0- x<30 ^ 10≤ z0- z1<20
lA0, {(lB1, {z1, z0}), (lB1, {z3, z0})}
0≤x=z3 ^ z1≥10 ^ z0≥10 ^ 10≤ z1-x
^ 10≤ z0- x<20 ^ 0≤ z0- z2<20
lA0, {(lB0, {z3, z0}), (lB1, {z1, z0}), (lB1, {z3, z0})}
0≤x=z3 ^ z1≥0 ^ z0≥0 ^ 0≤ z1-x<10
^ 0≤ z0- x<20 ^ 0≤ z0- z1<20

Figure 6. Product automaton with minimizing clocks in level 3

## C. *Verification*

The final step of the semi-algorithm aims to determine whether the language of timed words allowed by the timed automaton A is entirely contained within the language allowed by another timed automaton B. In other words, it verifies if all the timed behaviors possible in A are also valid behaviors within the timed automaton B. In this paper, the verification process involves checking the product automaton P for a specific state: Is there a state in P where $X_b$ is empty?

If the semi-algorithm explores all reachable states in P and never encounters an empty nodes $X_b$, it strengthens the case for language inclusion. If it finds a state in P with an empty $X_b$, it indicates a violation of language inclusion.

**Example 3:**
Assuming Fig. 6 depicts the product automaton for our verification process, we can analyze it to check language inclusion. We note that the tree structure represents an infinite tree. In this case, we need to make an assumption about the number of clocks involved. We assume a maximum of 3 clocks for this analysis. Thus, we conclude that language inclusion is verified. In simpler terms, the timed automaton A is entirely contained within the language allowed by timed automaton B.

## IV. Verification of Inclusion Problem using a Discretization Framework

The work in [3] proposes a novel approach to verifying language inclusion for timed automata. It moves beyond traditional discretization techniques that combine timed words into simplified representations. Instead, it introduces the concept of a "timed bounded discretized language." This language consists of discrete timed words, each capturing an action along with its minimum and maximum possible execution times. This allows the verification process to leverage this richer information compared to using just the actions themselves. The following section details the steps involved in this verification using the discretized framework.

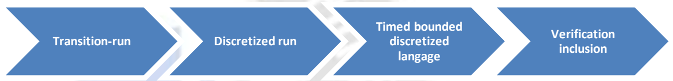

Figure 1: Steps of the timed bounded verification of inclusion in [3]

## A. *Transition-run*

The first step involves determining all possible transition-runs for both timed automata being compared. A transition run is a sequence of transitions and locations that the automaton can take. However, these transition-runs could be infinite due to ever-increasing time delays. To address this challenge, the work in [3] proposes a novel approach. They suggest introducing a bound on the time execution for the last transition in a given transition-run. By introducing such bounds, the authors ensure that the set of possible transition-runs becomes finite, enabling further analysis within the verification process.

**Example 4:**
Let A and B be two timed automaton shown in Fig. 4 and Fig.2 respectively and α=5 be a constant value. The set of transition-runs of A and B is introduced in Table I.

Transition-runs generated by A Transition-runs generated by B

TABLE I. Transition-runs of Automata A and B

| *Transition-runs generated by A* | *Transition-runs generated by B* |
|---|---|
| $l_{A0} \xrightarrow{e_0} l_{A0}$ | $l_{B0} \xrightarrow{e_0} l_{B0}$ |
| $l_{A0} \xrightarrow{e_0} l_{A0} \xrightarrow{e_0} l_{A0}$ | $l_{B0} \xrightarrow{e_0} l_{B0} \xrightarrow{e_0} l_{B0}$ |
| | $l_{B0} \xrightarrow{e_0} l_{B0} \xrightarrow{e_0} l_{B0} \xrightarrow{e_1} l_{B1}$ |
| | $l_{B0} \xrightarrow{e_0} l_{B0} \xrightarrow{e_1} l_{B1}$ |
| | $l_{B0} \xrightarrow{e_0} l_{B0} \xrightarrow{e_1} l_{B1} \xrightarrow{e_2} l_{B1}$ |
| | $l_{B0} \xrightarrow{e_1} l_{B1}$ |
| | $l_{B0} \xrightarrow{e_1} l_{B1} \xrightarrow{e_2} l_{B1}$ |
| | $l_{B0} \xrightarrow{e_1} l_{B1} \xrightarrow{e_2} l_{B1} \xrightarrow{e_2} l_{B1}$ |

## *B. Discretized runs*

Building upon the identified transition runs from step 1, the authors in [3] introduce two key concepts: Minimum Execution Duration (Dmin) and Maximum Execution Duration (Dmax). Dmin represents the shortest possible time it takes to reach a specific transition within a given transition run. Dmax represents the longest possible time it takes to reach that same transition.

By analyzing each transition run, the authors construct a new type of sequence called a discretized run. This discretized run captures a series of elements, each represented as (action, [Dmin, Dmax]) where action refers to the specific action performed during the transition and [Dmin, Dmax] is a pair of values representing the minimum and maximum execution times to reach the transition associated with that action. Note that if the constraints of clocks is strict, then the interval is opened and if the constraints of clocks is large, then the interval is closed.

**Example 5:**

According to the results shown in Table 1, we determine the set of discretized runs of A and B presented in Table II.

TABLE II. DISCRETIZED RUNS OF AUTOMATA A AND B

| ***Discretized runs generated by A*** | ***Discretized runs generated by B*** |
|---|---|
| $l_{A0} \xrightarrow{(a,]2,10[)} l_{A0}$ | $l_{B0} \xrightarrow{(a,]2,10[)} l_{B0}$ |
| $l_{A0} \xrightarrow{(a,]2,10[)} l_{A0} \xrightarrow{(a,]2,10[)} l_{A0}$ | $l_{B0} \xrightarrow{(a,]2,10[)} l_{B0} \xrightarrow{(a,]2,10[)} l_{B0}$ |
|  | $l_{B0} \xrightarrow{(a,]2,10[)} l_{B0} \xrightarrow{(a,]2,10[)} l_{B0} \xrightarrow{(a,]0,\infty[)} l_{B1}$ |
|  | $l_{B0} \xrightarrow{(a,]2,10[)} l_{B0} \xrightarrow{(a,]0,\infty[)} l_{B1}$ |
|  | $l_{B0} \xrightarrow{(a,]2,10[)} l_{B0} \xrightarrow{(a,]0,\infty[)} l_{B1} \xrightarrow{(a,]2,20[)} l_{B1}$ |
|  | $l_{B0} \xrightarrow{(a,]0,\infty[)} l_{B1}$ |
|  | $l_{B0} \xrightarrow{(a,]0,\infty[)} l_{B1} \xrightarrow{(a,]2,20[)} l_{B1}$ |
|  | $l_{B0} \xrightarrow{(a,]0,\infty[)} l_{B1} \xrightarrow{(a,]2,20[)} l_{B1} \xrightarrow{(a,]2,20[)} l_{B1}$ |

## *C. Timed bounded discretized language*

This step focuses on converting the discretized runs from step 2 into a format suitable for the timed bounded discretized language. Each discretized run is transformed into a discrete timed word (action, [Tmin, Tmax]) where [Tmin, Tmax] represents the minimum and maximum execution times to reach the transition associated with the action.

Finally, all the obtained discrete timed words generated from each discretized run are gathered together to form the timed bounded discretized language. This language becomes the foundation for the verification process, allowing the comparison of behaviors between the two timed automata while considering both actions and their possible execution time ranges.

**Example 6:**

According to the results shown in Table 2, we determine the timed bounded discretized language of A and B as follow:

DTL(A,3)={(a,]2,10[), (a,]2,10[) (a,]4,20[)};

DTL(B,3)={(a,]2,10[), (a,]2,10[) (a,]4,20[), (a,]2,10[) (a,]4,20[) (a,]4,20+∞[), (a,]2,10[) (a,]2,10+∞[), (a,]2,10[) (a,]2,10+∞[) (a,]4,30+∞[), (a,]0,+∞[) (a,]2,20+∞[), (a,]0,+∞[) (a,]2,20+∞[) (a,]4,40+∞[)}

## *D. Verification*

With the timed bounded discretized language (DTLs) constructed for both automata (A and B), the actual verification of language inclusion can now take place. This step involves checking each element (discrete timed word) within the DTL of automaton A (denoted as DTL(A)) against the elements in the DTL of automaton B (DTL(B)). Every discrete timed word in DTL(A) must have a corresponding element (another discrete timed word) within DTL(B). The corresponding elements in both DTLs should share the same actions. Additionally, each interval (minimum and maximum execution time range) for an action in the discrete timed word generated from A should be entirely contained within the corresponding interval for the same action in the discrete timed word generated from B.

According to the results of timed bounded discretized language of A and B, we obtain that DTL(A, α) is included in DTL(B, α) (for each discrete timed word in DTL(A, 5), there exists an equivalent discrete timed word in DTL(B, 5)).

## V. COMPARISON BETWEEN THESE METHODS

This section focuses on how the previously discussed methods ([2] – semi-algorithm and [3] - Timed Bounded Discretized Language) compare for bounded verification of language inclusion between timed automata A and B.

### *A. Finiteness*

The authors of method [2] acknowledge that their approach can potentially lead to infinite exploration, which could prevent verification from ever terminating. In contrast, the authors of method [3] have proven that their approach using timed bounded discretized language is guaranteed to terminate (finite).

### *B. Bounding Techniques*

The key difference between the methods lies in their approach to bounding exploration. Method in [2] relies on a bound on the number of clocks generated during the verification process. Method in [3] employs a bound on the execution time for transitions within the automata.

### *C. Complexity and Memory Usage:*

Applying both methods to automata A and B, we observe that method [2] (semi-algorithm) is generally more complex than method [3] (timed bounded discretized language).

This increased complexity is also reflected in the memory usage. Table III demonstrates that method [2] requires more memory space for the verification of inclusion compared to method [3].

TABLE III. SPACE MEMORY

|  | ***Method in [2]*** | ***Method in [3]*** |
|---|---|---|
| Formula | $2^{\|L_B\|} * \gamma^{\|X\|+1} (2b)^{\gamma^2} * \gamma! * \|L_A\|$ | $\|DTL(A,5)\| * \|DTL(A,5)\| = 3 * k^2 * \|T'_A\|^k * \|T'_B\|^k$ |
| Calcul | $2^2 * 3^2 * (2*3)^{3*3} * 3! * \|1\| =$ 2 176 782 336 | $3 * 3^2 * \|1\|^3 * \|2\|^3 = 216$ |

Where b is the maximum number clocks, k is the maximum length of the discrete timed word and |T'| represents the maximum number of out-degrees of a location.

## VI. CONCLUSION

This paper investigated the verification of language inclusion for timed automata, focusing on the challenges posed by non-Zeno behavior. We explored two verification approaches: the zone-based semi-algorithm [2] and the timed bounded discretized language method [3].

Our analysis revealed key differences between these approaches. While the zone-based semi-algorithm is a well-established technique, it can lead to infinite exploration. The timed bounded discretized language method offers a promising alternative by incorporating execution time information, leading to a guaranteed termination process and more comprehensive verification.

The comparison of these methods on specific timed automata highlighted the advantages of the timed bounded discretized language approach in terms of guaranteed termination, lower memory usage, and more informative verification results.

Building upon the insights gained from this comparison, future work could involve exploring the potential benefits of combining elements from both approaches to create a hybrid verification technique that leverages the strengths of each. Furthermore, the development of software tools based on these verification approaches can facilitate practical application in the design and verification of real-time systems.

## REFERENCES

[1] Rajeev Alur and David Dill, A theory of timed automata, Theor. Comput. Sci. (1994), pp. 183--235.
[2] Y. Wang, M. Sun, L. Sun, and J. Sun. "Language Inclusion Checking of Timed Automata with Non-Zenoness." In IEEE Transactions on Software Engineering, vol. 43, no. 1, pp. 142-159, 2017.
[3] I. Ammar, Y. El Touati, J. Mullins, and M. Yeddes. "Timed Bounded Verification of Inclusion Based on Timed Bounded Discretized Language." In International Journal of Foundations of Computer Science, vol. 32, no. 01 (2021), pp. 1-23, 2021.
[4] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Hooman. "Model-checking for Timed Automata." In Theoretical Computer Science, vol. 290, no. 1, pp. 251-273, 2003.
[5] S. Tripakis. "Formal Testing for Timed Systems: A Survey." In Real-Time Systems Symposium (RTSS'99), pp. 166-175, IEEE, 1999.
[6] Ting Wang, Jun Sun, Yang Liu, Xinyu Wang and Shanping Li, Are timed automata bad for a specification language? Language inclusion checking for timed automata, Tools and Algorithms for the Construction and Analysis of Systems, Vol. 8413 (International Conference TACAS, 2014), pp. 310--325.
[7] Ikhlass Ammar, Yamen El Touati et Moez Yeddes. «Verification of bounded inclusion problem for Timed automata with diagonal constraints». International Conference on ELectrical, Computer, and Energy Technologies (ICECET)-IEEE Explore to appear 20-22 july 2022.