Model Checking Continuous-time Duration Calculus Against Bounded Behaviour of Timed Automata

ABSTRACT

Extended Linear Duration Invariants (ELDIs), an important subset of Duration Calculus, extends well-studied Linear Duration Invariants with logical connectives and the chop modality. It is known that the model checking problem of ELDIs is undecidable with both the standard continuous-time and discrete-time semantics [12, 13], but it turns out to be decidable if only bounded execution fragments of timed automata are concerned in the context of the discrete-time semantics [38]. In this paper, we further prove that this problem is still decidable in the continuous-time semantics, although it is well-known that model-checking Duration Calculus with continuous-time semantics is much more complicated than with discrete-time semantics. This is achieved by reduction to the validity of Quantified Linear Real Arithmetic (QLRA). Some examples are provided to illustrate the efficiency of our approach.

KEYWORDS

Model Checking, Duration Calculus, Extended Linear Duration Invariants, Timed Automata, Quantified Linear Real Arithmetic.

1 INTRODUCTION

Duration Calculus is an Interval Temporal Logic, which was first proposed by Zhou, Hoare and Ravn [19]. It extends Interval Temporal Logic [16] with the notion of *duration*, the integral of state expression over the reference interval. It is designed for specifying and reasoning about real-time and embedded systems at a high abstract level [14], and has been successfully and widely applied in practice [25, 35].

Although the powerful expressiveness of DC is desirable for requirement specification and analysis, it is a burden for automatic verification as satusfiability checking, validity checking or model checing of DC is undecidable [36], unless the use of chop (the only modality in DC), and/or negation, and/or the considered models are severely constrained [10, 11, 17, 25, 26, 30, 32, 36, 37].

In [37], a subset of DC formulas of the form $b \leq \ell \leq e \Rightarrow \sum_{s\in S} c_s \int s \leq M$, called Linear Duration Invariants (LDIs), was identified, in which a given interval $[t, t']$, $\int s$ stands for the accumulated time for the presence of state s over $[t, t']$, and ℓ for the length of the interval, i.e., $t' - t$. So, an LDI formula says that if the length of the interval satisfies the constraint $b \leq \ell \leq e$, then over the time interval the durations of the state expressions should satisfy the constraint $\sum_{s\in S} c_s \int s \leq M$. LDIs is very expressive. For example, in the gas burner example [35], the safety requirement that "the proportion of leak time is not more than one-twentieth of the elapsed time for any time interval at least one minute", can be simply specified as $\ell \geq 60 \Rightarrow 20 \int \mathit{Leak} \leq \ell$.

In [37], it was proved that the model-checking problem of LDIs against real-time automata is decidable. Real-time automata are a specific kind of timed automata in which reset is not allowed. Following this work, in [4, 22, 23], the authors investigated the model-checking of LDIs against more expressive models such as timed automata [1] and hybrid automata [18]. Some more efficient algorithms for mode-checking LDIs against timed automata based on graph search were proposed in [32, 33], and the results were further extended to the networks of timed automata [34].

An interesting problem is whether it is possible to find a larger subset of DC whose model-checking problem is still decidable. It is a natural solution to investigate the extension of LDIs with Boolean connectives and the chop modality, i.e., ELDIs. Unfortunately, the satisfiability and validity of ELDIs both are undecidable in the discrete-time and continuous-time (dense-time) settings according to the results in [36]. In [12, 13], Fränzle and Hansen further proved that the model-checking problem of ELDIs against finite state machines turns out to be undecidable also both in the discrete time and continuous time settings. Therefore, they proposed an approximation semantics for ELDIs, called doubly situation based semantics, and showed its model-checking is decidable in the discrete time setting with the complexity of cubic in the number of states of the model and linear in the size of the formula. However, further observation indicates that such

HSCC'2018, Porto, Portugal

DOI: XX.XXX/XXX_X

approximation semantics is too coarse to be useful in practice [12]. So, they refined the semantics to another approximation semantics called counting semantics and reduced the model-checking problem of ELDIs to Presburger Arithmetic with the complexity of 3-fold exponential [13]. In addition, according to their approach, one can only prove/disprove those formulas that can be approximated to be **true/false** over the given model represented by a Kripke structure, and cannot draw any conclusion in other cases. Therefore, the low efficiency and approximation semantics hinder the application of their approach. Motivated by their work, as an alternative, in [38] Zu *et al.* proved that model-checking ELDIs against bounded execution fragments of timed automata in the standard discrete-time semantics is decidable by providing an efficient model-checking algorithm with the complexity of singly exponential in the size of formulas and quadratic in the number of states of the considered model.

In this paper, we investigate the model-checking problem of ELDIs against bounded execution fragments of timed automata in the standard continuous-time semantics. We prove that it is still decidable. The basic idea is that for a given timed automaton $\mathcal{A}$ and an ELDI formula Φ, we first find the set Θ of all execution fragments of $\mathcal{A}$ whose length is in between the lower and upper bounds from the zone graph of $\mathcal{A}$, and the number of such paths is finite. Then, for each path in Θ, we construct a Quantified Linear Real Arithmetic (QLRA) formula by taking Φ into account. Finally, we exploit REDLOG [7], a computer algebra tool based on quantifier elimination [31], to solve the resulting QLRA formula. Although the complexity of REDLOG is double exponential in the number of variables in the worst case, REDLOG can handle QLRA formulas very efficiently in practice, as *virtual substitution* [8] can be applied to formulas which only contain polynomials with degree no more than 4. Our experiments indicate the efficiency of our approach.

The rest of the paper is organized as follows. Section 2 recalls some basic notions of timed automata, zones and ELDIs. Section 3 explains how to find all execution fragments with the bounded length for a given timed automaton from its zone graph. The algorithm of constructing a QLRA formula according to a given ELDI formula and an execution fragment is presented in section 4. Section 5 is devoted to solve the resulting QLRA formulas by quantifier elimination and the complexity analysis. Section 6 presents the implementation and experiments, and finally section 7 draws a conclusion.

2 PRELIMINARIES

In this section, we first review timed automata (TA) as a modeling language for real-time systems and Zone graph as the symbolic representation of states of TA, then recall ELDIs, a subset of DC, as a specification logic for real-time systems. For convenience, we fix a set of propositions $\mathcal{P}$ throughout this paper.

2.1 Timed automata

A timed automaton (TA) [1] is a finite-state automaton equipped with a set of clocks, and each location is equipped with a set of propositions from $\mathcal{P}$ that hold at the location. Let X represent the set of clocks and $\Delta(X)$ be the set of *clock constraints* on X, which are conjunctions of the formulas of the form $x \leq c$ or $c \leq x$, where $x \in X$ and $c \in \mathbb{N}$. Additionally, we assume that all TA are strongly *non-Zeno* [1], i.e., there is a non-zero constant $\epsilon \in \mathbb{R}_+$ such that in every control cycle at least ϵ units of time is taken.

Definition 2.1 (Timed Automaton). A TA is a tuple $\mathcal{A} = (L, X, E, \Sigma, l_0, \Lambda, I)$, where L is a finite set of locations, X is a finite set of clocks, $E \subseteq L \times \Sigma \times \Delta(X) \times 2^X \times L$ is a transition relation, Σ is a set of actions, $l_0 \in L$ is the initial location, Λ is a mapping that assigns a subset of $\mathcal{P}$ to each location l indicating all propositions in $\Lambda(l)$ hold in l, and I is a mapping that assigns each location $l \in L$ with a clock constraint $I(l) \in \Delta(X)$, called *invariant*. Furthermore, an invariant in TA must be downward closed.

A *clock valuation* is a function $\mathbf{v} : X \to \mathbb{R}_+$. We denote the set of all clock valuations by $\mathcal{H}$. Hence a state of a TA $\mathcal{A}$ is a pair $(l, \mathbf{v}) \in L \times \mathcal{H}$ consisting of a location and a clock valuation. Every subset $\lambda \subseteq X$ induces a reset function $\mathit{Reset}_\lambda : \mathcal{H} \to \mathcal{H}$ defined by

$$\mathit{Reset}_\lambda \mathbf{v}(x) = \begin{cases} 0, & \textbf{if } x \in \lambda \\ \mathbf{v}(x), & \textbf{if } x \notin \lambda \end{cases}$$

We use $\mathbf{1}$ to denote the unit vector(1,...,1) and $\mathbf{0}$ for zero vector. There are two kinds of *steps* of a TA: *discrete step* and *time-delay step*. A discrete step is of the form $(l, \mathbf{v}) \xrightarrow{a} (l', \mathbf{v}')$, if there exists $(l, a, g, \lambda, l') \in E$ such that $\mathbf{v}$ satisfies g and $\mathbf{v}' = \mathit{Reset}_\lambda(\mathbf{v})$, where $a \in \Sigma$. A time-delay step is of the form $(l, \mathbf{v}) \xrightarrow{t} (l, \mathbf{v} + t\mathbf{1})$, such that for any $t' \in [0, t]$ $v + t'\mathbf{1}$ satisfies $I(l)$, where $t \in \mathbb{R}_+$.

Definition 2.2 (Run and Behaviour). Let $\mathcal{A}$ be a TA,

(1) the run r of $\mathcal{A}$ is an infinite sequence of the form

$$r \ : \ (l_0, \mathbf{v}_0) \xrightarrow[\delta_0]{a_0} (l_1, \mathbf{v}_1) \xrightarrow[\delta_1]{a_1} (l_2, \mathbf{v}_2) \xrightarrow[\delta_2]{a_2} \cdots$$

where $(l_0, \mathbf{v}_0)$ is the initial state, and δ_i is the time $\mathcal{A}$ staying in the location l_i. If $l_i = l_{i+1}$, the step is a time-delay step and $\mathbf{v}_{i+1} = \mathbf{v} + \delta_i \mathbf{1}$; otherwise, the step is a discrete step and $\mathbf{v}_{i+1} = \mathit{Reset}_\lambda(\mathbf{v_i})$, for $i \geq 0$.

(2) a behaviour β corresponding to the run, is the infinite sequce of timed locations

$$\beta \ : \ (l_0, t_0)(l_1, t_1) \cdots (l_k, t_k) \cdots$$

that satisfies following conditions: (1) $t_0 = 0$; (2) for any $T \in \mathbb{R}^+$, there is some $i \geq 0$ such that $t_i \geq T$; (3) t_i is the instant that $\mathcal{A}$ enters to l_i, which implies $\delta_i = t_{i+1} - t_i$, and $\mathcal{A}$ stays in l_i for δ_i time units.

A *zone* is a clock constraint [3]. For a location, a zone is the maximal set of clock valuations satisfying the constraint. In the zone-graph, zones are used to denote symbolic states. Zones and their representations based on Difference Bounded Matrices (DBMs) [2] are the standard data structures which have been implemented in several verification tools for TA, e.g., UPPAAL [21]. Operations over zones are well defined in [3]. Note that zone provides a more efficient representation of symbolic states of TA than *region graph* [1], therefore, model-checking algorithms for TA based on zone is more efficient than those based on region graph.

2.2 Extended linear duration invariants

ELDIs with the set $\mathcal{P}$ of state variables consists of three syntactic categories, which are state expressions S, linear duration formulas (LDFs) $\mathcal{D}$, and ELDI formulas ϕ. The BNFs for them are given as follows:

$$\begin{aligned} S &::= 0 \mid P \mid \neg S \mid S_1 \vee S_2 \\ \mathcal{D} &::= \sum_{i\in\Omega} c_i \textstyle\int S_i \leq c \\ \phi &::= \mathcal{D} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 ; \phi_2 \end{aligned}$$

where $P \in \mathcal{P}$ stands for a state variable, interpreted as a Boolean function over time, c_is and c are real numbers, and Ω is a finite set of indices.

As the convention of DC, ℓ is defined as $\int 1$, denoting the length of the reference interval. The Boolean value **true**, denoted by $\top$, is defined by $\ell \geq 0$, falling in ELDIs. Obviously, each ELDI formula can be represented by the form $b \leq \ell \leq e \Rightarrow \phi$, where b is either a positive real or 0, e is either a positive real or ∞, and ϕ is defined as above. In this paper, we only focus on the case when e is bounded, and will represent an ELDI of this form by $\Phi, \Psi, \cdots$, possibly with superscript and subscript in the sequel.

Definition 2.3 (Interpretation $\mathcal{I}_\beta$ of ELDIs). Given a TA $\mathcal{A}$ and one of its behaviours β, define an interpretation $\mathcal{I}_\beta$ of ELDIs as follows:

state expressions: given a time point $t \in \mathbb{R}^+ \cup \{0\}$
$\mathcal{I}_\beta(0)(t) = 0$ and $\mathcal{I}_\beta(1)(t) = 1$

$$\mathcal{I}_\beta(P)(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \wedge P \in l_i \\ & \quad \text{for some } i > 0 \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{I}_\beta(\neg S)(t) = 1 - \mathcal{I}_\beta(S)(t)$
$\mathcal{I}_\beta(S_1 \vee S_2)(t) = \max\{\mathcal{I}_\beta(S_1)(t), \mathcal{I}_\beta(S_2)(t)\}$

duration: given an interval $[t_1, t_2]$, where $t_1, t_2 \in \mathbb{R}^+ \cup \{0\}$ and $t_1 \leq t_2$, $\int S$ is interpreted by $\mathcal{I}_\beta(\int S)([t_1, t_2]) = \int_{t_1}^{t_2} \mathcal{I}_\beta(S)(t)dt$.

formulas: given an interval $[t_1, t_2]$, an ELDI formula ϕ is interpreted by
$\mathcal{I}_\beta, [t_1, t_2] \models \sum_{i\in\Omega} c_i \int S_i \leq c$ iff $\sum_{i\in\Omega} c_i \mathcal{I}_\beta(\int S_i)([t_1, t_2]) \leq c$;
$\mathcal{I}_\beta, [t_1, t_2] \models \neg\phi$ iff $\mathcal{I}_\beta, [t_1, t_2] \not\models \phi$;
$\mathcal{I}_\beta, [t_1, t_2] \models \phi_1 \vee \phi_2$ iff $\mathcal{I}_\beta, [t_1, t_2] \models \phi_1$ or $\mathcal{I}_\beta, [t_1, t_2] \models \phi_2$;
$\mathcal{I}_\beta, [t_1, t_2] \models \phi_1 ; \phi_2$ iff $\mathcal{I}_\beta, [t_1, t] \models \phi_1$ and $\mathcal{I}_\beta, [t, t_2] \models \phi_2$ for some $t \in [t_1, t_2]$.

2.3 Quantified linear real arithmetic

Quantified linear real arithmetic (QLRA) is a theory of first order logic, with the specific signature $\langle \mathbb{R}, 0, +, =, < \rangle$, i.e., in which all terms are linear. Thus, formulas of QLRA are defined according to the following syntax:

$$\phi \quad ::= \quad c_0 + c_1 x_1 + \cdots + c_n x_n \rhd 0 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \forall x.\phi$$

where $c_0, c_1, \ldots, c_n \in \mathbb{R}, \rhd \in \{=, <\}$. QLRA is interpreted in the standard way. Other notations of first-order logic and real arithmetic are defined as usual, e.g., $\phi \vee \psi \hat{=} \neg(\neg\phi \wedge \neg\psi)$, $\exists x.\phi \hat{=} \neg\forall x.\neg\phi$, $t \leq 5 \hat{=} t < 5 \vee t = 5$, etc.

2.4 A motivating example

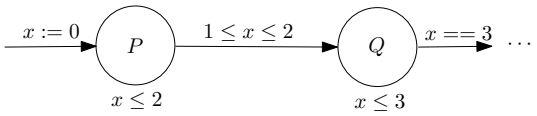


Figure 1: A motivating example

Consider a simple incomplete TA as in Fig. 1. When the automaton enters the location P, it resets the clock x. The invariant of the location P is $x \leq 2$ which means that the automaton can stay in P for two time units at most. There is a transition from the location P to the location Q and the guard is $1 \leq x \leq 2$. The invariant at the location Q is $x \leq 3$, and the transition left from Q has a guard $x = 3$. So, the total time the automaton can stay at P and Q is three time units. Consider ELDI formula

$$3 \leq \ell \leq 3 \Rightarrow 2(\textstyle\int P + \int Q) \geq 3 \,; 2(\int P + \int Q) \geq 3$$

on the time interval $[0, 3]$.

Obviously, the formula is unsatisfiable on the interval with the discrete semantics, but it becomes satisfiable in continuous semantics.

3 FINDING ALL POSSIBLE EXECUTION FRAGMENTS WITH BOUNDED LENGTH

In this section, we consider given a TA $\mathcal{A}$ and an interval $[b, e]$ with $b, e \in \mathbb{R}^+ \cup \{0\}$ and $b \leq e$ how to find all execution fragments of $\mathcal{A}$ whose length are in between b and e.

As discussed in [3], *ZoneG*($\mathcal{A}$) is a transition graph derived from $\mathcal{A}$, can be represented by $\langle Z, z_0, \mapsto \rangle$, where

- Every $z \in Z$ stands for a zone, which is a pair $(l, \Delta(l))$, where $l \in \mathcal{A}.L$, and $\Delta(l)$ is a timing constraint on $\Delta(\mathcal{A}.X)$, denoting a symbolic state of $\mathcal{A}$, i.e., a set of clock valuations satisfying $\Delta(l)$.
- $z_0 = (l_0, \wedge_{x \in \mathcal{A}.X} x = 0)$, is the initial zone.
- $\mapsto = \uparrow \cup \cup_{a \in \Sigma} \xrightarrow{g,a,\lambda}$, where
  - $(l_1, \Delta(l_1)) \uparrow (l_2, \Delta(l_2))$ iff $l_1 = l_2 \wedge \Delta(l_2) = \{u + d \in I(l_1) \mid u \in \Delta(l_1) \wedge d \in \mathbb{R}^+\}$. $\uparrow$ stands for a delay transition.
  - $(l_1, \Delta(l_1)) \xrightarrow{g,a,\lambda} (l_2, \Delta(l_2))$ iff $(l_1, g, a, \lambda, l_2) \in \mathcal{A}.E \wedge \Delta(l_2) = \{u[\lambda] \mid u \in \Delta(l_1) \wedge g(u)\}$. $\xrightarrow{g,a,\lambda}$ stands for a possible discrete transition over the zone graph derived from $\mathcal{A}$.

Given a transition $\tau \in \mapsto$, $\tau.preZ$ and $\tau.postZ$ respectively denote the pre- and post-zone of the transition, while $\tau.g$, $\tau.a$ and $\tau.\lambda$ denote its guard, action and the set of reset clocks, respectively, if τ is a discrete transition. Given a zone z, we use $z.\tau$ stands for the set of transitions outgoing from z, and $Post^a(z)$ for the set of zones which can be reached from z via a discrete transition, i.e., $\{z' \mid (z, g, a, \lambda, z') \in \mapsto$ for some $g \in \Delta(X), a \in \Sigma,$ and $\lambda \subseteq X\}$.

Given a TA $\mathcal{A}$, the procedure for finding all possible execution fragments whose length is in the given interval $[b, e]$ is implemented by Algorithm 1. The basic idea of Algorithm 1 is as follows:

- According to the descriptions of zone in [3, 28, 29], we first construct a zone graph of $\mathcal{A}$ with k-normalization, denoted by *ZoneG*($\mathcal{A}$) the resulting zone graph. As $\mathcal{A}$ does not have time difference constraints as guard, thus *ZoneG*($\mathcal{A}$) is *sound* and *complete* w.r.t. the standard operational semantics of $\mathcal{A}$ as discussed in the previous section and its transition relation is finite [28, 29].
- Starting from each symbolic state in *ZoneG*($\mathcal{A}$), we find all execution fragments whose length in between b and e using depth first search. With an implicit extra clock t added to the DBMs, we can easily determine whether the length of the current execution fragment ρ can reach the interval $[b, e]$ through checking whether $currentZ \wedge (b \leq t \leq e)$ holds. *STK* is a stack to store the zones to be explored. The algorithm pushes a zone z into *STK* as long as the length of the

Algorithm 1: PEF(*ZoneG*($\mathcal{A}$), b, e) /* Finding all possible execution fragments satisfying the length bound */

input : *ZoneG*($\mathcal{A}$), the zone graph of $\mathcal{A}$, $[b, e]$, the bound interval
output : Θ, the set of possible execution fragments whose length is within $[b, e]$

```
1  begin
2      Θ := ∅ ;
3      foreach z ∈ Z do
4          ρ := ε ;
5          STK.push(z) ;
           /* STK is a stack, stores all zones
              to be explored                    */
6          while STK ≠ ∅ do
7              currentZ := STK.pop;
8              if ↑∈ currentZ.τ then
9                  currentZ := currentZ↑ ;
                   /* t is an additional clock added
                      to the DBMs, reseted at the
                      beginning of ρ             */
10             if currentZ ∧ (b ≤ t < e) ≠ ∅ then
11                 ρ := ρ ∘ ⟨currentZ⟩; Θ := Θ ∪ {ρ};
12                 if Post^a(currentZ) ≠ ∅ then
13                     foreach z' ∈ Post^a(currentZ) do
14                         STK.push(z');
15                 else
16                     ρ := remove(ρ, currentZ);
                       /* remove currentZ from ρ
                          */
17                     while Post^a(Lastzone(ρ)) have been
                       popped out do
18                         ρ := remove(ρ, Lastzone(ρ));
19             else if currentZ ∧ (t < b) ≠ ∅ then
20                 ρ := ρ ∘ ⟨currentZ⟩;
21                 if Post^a(currentZ) ≠ ∅ then
22                     foreach z' ∈ Post^a(currentZ) do
23                         STK.push(z');
24                 else
25                     ρ := remove(ρ, currentZ);
26                     while Post^a(Lastzone(ρ)) have been
                       popped out do
27                         ρ := remove(ρ, Lastzone(ρ));
28             else if currentZ ∧ (t ≥ e) ≠ ∅ then
29                 ρ := ρ ∘ ⟨currentZ⟩; Θ := Θ ∪ {ρ};
30                 ρ := remove(ρ, currentZ) ;
31                 while Post^a(Lastzone(ρ)) have been poped
                   out do
32                     ρ := remove(ρ, Lastzone(ρ));
33     return Θ
```

derived execution fragment by appending Z to the end of the considered execution fragment is no more than the given upper bound e, otherwise, the head of *STK* will be popped. Suppose the number of the locations of $\mathcal{A}$ is N, then the number of such execution fragments is at most $N^{1+N*(1+\lceil \frac{e}{\epsilon} \rceil)}$, where ϵ is the least dwelling time in each control cycle of $\mathcal{A}$.

THEOREM 3.1 (CORRECTNESS OF ALGORITHM 1). *For any TA $\mathcal{A}$ and $[b, e]$, Algorithm 1 is correct, i.e.,*

Termination *the algorithm terminates;*
Soundness *if $\rho \in \Theta$ then ρ is a real execution fragment of zones in ZoneG($\mathcal{A}$) with length in $[b, e]$; and*
Completeness *if ρ is a real execution fragment of zones in ZoneG($\mathcal{A}$) with length in $[b, e]$, then $\rho \in \Theta$.*

PROOF. For **termination**, to guarantee the termination of Algorithm 1, we only need to prove the **while** loop terminates as it is obvious that the outermost **for** loop and the three innermost **for** loops terminate. For a given zone z, suppose there is an execution fragment ρ starting from z whose length is within $[b, e]$ which has $N * (1 + \lceil \frac{e}{\epsilon} \rceil)$ transitions, where N is the number of zones. According to König Lemma [9, 20], there is a zone z' with more than $1 + \lceil \frac{e}{\epsilon} \rceil$ occurrences in ρ, which implies there are $\lceil \frac{e}{\epsilon} \rceil$ control cycles in ρ at least. Hence, the execution time of ρ is more than e from the assumption that each control cycle has ϵ dwelling time at least. It follows that any execution fragment starting from z has $N*(1 + \lceil \frac{e}{\epsilon} \rceil)$ transitions at most if its length is within $[b, e]$, and therefore, all such execution fragments can be found in $N^{N*(1+\lceil \frac{e}{\epsilon} \rceil)}$ iterations at most. After finding out all such execution fragments, no zone can be pushed in *STK* any more, but at least one zone is popped out from *STK* in each iteration (see lines 10-32). This implies that the **while** loop must terminate.

For **soundness**, suppose $\rho \in \Theta$. Obviously, ρ is an execution fragment of *ZoneG*($\mathcal{A}$) by Algorithm 1. Additionally, from lines 10-32, it follows that ρ's length is within $[b, e]$, because on one hand, ρ's length cannot be less than b as $\rho \notin \Theta$ otherwise; on the other hand, ρ's length cannot be greater than e, as the last zone in ρ cannot be appended otherwise according to lines 28-32.

For **completeness**, suppose $\rho = (l_1, \Delta(l_1)), ..., (l_n, \Delta(l_n))$ is a real execution fragment of *ZoneG*($\mathcal{A}$) whose length is within $[b, e]$. Thus, we can construct a $\rho' \in \Theta$ as follows: Let ρ' starts from $(l_1, \Delta(l_1))$. Obviously, it is doable by line 3. Because ρ is a real execution fragment of *ZoneG*($\mathcal{A}$), the second zone $(l_2, \Delta(l_2))$ in ρ must be a successor of $(l_1, \Delta(l_1))$, and the length of $(l_1, \Delta(l_1)), (l_2, \Delta(l_2))$ is less than e, so ρ' can be extended by appending $(l_2, \Delta(l_2))$ at the end from lines 10-27. We can repeat the above procedure until $(l_n, \Delta(l_n))$ is appended to the end of ρ'. Clearly, from lines 10-14, $\rho' \in \Theta$. □

4 REDUCTION TO QRLA

In this section, we present a translation from a given possible execution fragment whose length is within the given interval and an ELDI formula into a QLRA formula equivalently in the sense that the execution fragment satisfies the ELDI formula iff the resulting QLRA formula is valid.

Note that for any execution fragment generated by Algorithm 1 which is a sequence of zones, we can remove these zones without dwelling time, because $(\phi; \ell = 0) \Leftrightarrow (\ell = 0; \phi) \Leftrightarrow \phi$ always holds in DC. So, in what follows, we only consider such refined execution fragments, and denote by $z_1, z_2, \ldots, z_k$. Moreover, for each zone z_i, we introduce a variable δ_i to indicate the real duration the automaton dwells on. Thus, we will denote the refined execution fragment as $(z_1, \delta_1)(z_2, \delta_2), \ldots, (z_k, \delta_k)$.

Given the ELDI formula $(b \leq \ell \leq e \Rightarrow \phi)$ and a refined execution fragment $\rho = (z_1, \delta_1)(z_2, \delta_2), \ldots, (z_k, \delta_k)$, the encoded QLRA formula is of the form $L(\rho) \Rightarrow L(\phi)$, where $L(\rho)$ is a QLRA formula translated from ρ, which entails $b \leq \sum_{i=1}^{k} \delta_i \leq e$, and $L(\phi)$ is obtained from ϕ. We will explain the details of the translation later.

Algorithm 2: *LP*($\rho, \mathbf{x}_0$)

input : ρ, a refined execution fragment $(z_1, \delta_1), \ldots, (z_k, \delta_k)$; $\mathbf{x}_0$, the initial value of clocks at the beginning of ρ.
output : the timing constraint derived from the execution fragment.

1 **begin**
2 $z_1' = z_1[x_{01} + \delta_1/x_1, \cdots, x_{0m} + \delta_1/x_m]$;
3 **foreach** $i \in \{2, \ldots, k\}$ **do**
4 **foreach** $j \in \{1, \ldots, m\}$ **do**
5 **if** $x_j \in a_i.\lambda$ **then**
 /* a_i is the transition from z_{i-1} to z_i */
6 $e_j := \delta_i$;
 /* x_j should be reset */
7 **else**
8 $e_j := z_{i-1}.x_j + \delta_i$;
 /* $z_{i-1}.x_j$ stands for the value of clock x_j at the previous zone */
9 $z_i' = z_i[e_1/x_1, \cdots, e_m/x_m]$;
10 $\Gamma = \wedge_{i=1}^{k}(z_i' \wedge \delta_i \geq 0) \wedge b \leq \sum_{i=1}^{k} \delta_i \leq e$;
11 **return** Γ ;

Deriving timing constraint from a refined execution fragment: Given an execution fragment $\rho = (z_1, \delta_1), (z_2, \delta_2), \ldots, (z_k, \delta_k)$ and initial value $\mathbf{x}_0 = (x_{01}, \cdots, x_{0m})$ (suppose $X = \{x_1, \ldots, x_m\}$), Algorithm 2 derives a QLRA formula to stand for the timing constraint on the dwelling times δ_is

and the initial value $\mathbf{x}_0$ derived from ρ. Essentially, the constraint on the dwelling times δ_is is derived by considering the following three aspects:

- each δ_i should be nonnegative;
- their sum should be in the bound interval on the length of considered execution fragments, i.e., $b \leq \sum_{i=1}^{k} \delta_i \leq e$;
- the constraint derived from the corresponding zone by taking the initial values of clocks into account.

Encoding ELDI formula: Given an execution fragment $\rho = \langle (z_1, \delta_1), \ldots, (z_n, \delta_n) \rangle$ and an ELDI formula ϕ, the encoding procedure is done by the structure of ϕ as follows:

- If ϕ is an atomic formula of the form $b \leq \ell \leq e \Rightarrow \mathcal{D}$, we mainly focus on how to encode $\mathcal{D}$. Suppose $\mathcal{D}$ contains d duration expressions $\int S_1, \ldots, \int S_d$. We use e_{ij} to denote the duration that S_i holds at z_j, for $i = 1, \ldots, d$ and $j = 1, \ldots, n$. Clearly, if z_j satisfies S_i, then e_{ij} is δ_j, otherwise 0. Thus, the total duration of S_i holding on ρ should be $\sum_{j=1}^{n} e_{ij}$. Hence, we just need to replace each S_i with $\sum_{j=1}^{n} e_{ij}$ in $\mathcal{D}$, therefore, the translated QLRA formula is

$$LP(\rho, \mathbf{0}) \Rightarrow (\mathcal{D}[\sum_{i=1}^{n} e_{i1}/\textstyle\int S_1, \cdots, \sum_{i=1}^{n} e_{id}/\textstyle\int S_d]),$$

  where $LP(\rho, \mathbf{0})$ stands for the QLRA formula translated from ρ by applying Algorithm 2 with initial clock values $\mathbf{0}$, and $\phi[e_1/e_2]$ stands for replacing each occurrence of e_2 by e_1 in ϕ.
- When $\phi = \neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$, it is easy to obtain by revoking the procedure recursively.
- If $\phi = \phi_1; \phi_2$, then we consider the $n + 2$ cases where the chop point is taken respectively before z_1, at z_1, …, z_n, after z_n. Subsequently, we recursively recall the translation procedure to the resulted corresponding subproblems.
- Finally, quantify the resulted formula with the respective quantifications to the corresponding introduced fresh variables (line 26).

The above procedure is implemented by Algorithm 3, and the returned formula is a closed QLRA formula.

THEOREM 4.1 (CORRECTNESS OF ALGORITHM 3). *Algorithm 3 is correct, i.e.,*

Termination *the algorithm terminates;*
Soundness *if $\rho \models \phi$ then $LF(\phi, \rho, \mathbf{x}_0)$ is satisfiable (valid);*
Completeness *if $LF(\phi, \rho, \mathbf{x}_0)$ is satisfiable (valid), then $\rho \models \phi$.*

PROOF. Regarding **termination**, it can be simply done by induction on the structure of ϕ. Regarding **soundness** and **completeness**, we still proceed by induction on the structure of ϕ as follows:

- The basic case, i.e., ϕ is a formula of the form

$$b \leq \ell \leq e \Rightarrow \mathcal{D}, \text{ where } D = \sum_{i \in \Omega} c_i \textstyle\int S_i \leq c$$

  Then $\mathcal{I}_\rho, [\rho.b, \rho.e] \models b \leq \ell \leq e \Rightarrow \sum_{i\in\Omega} c_i \int S_i \leq c$ iff $b \leq \rho.e - \rho.b \leq e \Rightarrow \sum_{i\in\Omega} c_i \mathcal{I}_\rho(\int S_i)([\rho.b, \rho.e]) \leq c$, where $\rho.b$ and $\rho.e$ stands for the starting and ending points of ρ. Obviously,

$$\forall \delta_1, \cdots, \delta_n, \mathcal{Q}. \left(\begin{array}{l} LP(\rho, \mathbf{x}_0) \\ \Rightarrow \mathcal{D}[\sum_{i=1}^{n} e_{i1}/\int S_1, \cdots, \sum_{i=1}^{n} e_{id}/\int S_d] \end{array} \right) (1)$$

  is unsatisfiable implies $\rho \not\models \phi$ (soundness), and that (1) holds implies $\rho \models \phi$ (completeness) according to Algorithm 2 and Algorithm 3.
- $\phi = \neg\phi_1$
  For soundness, suppose $LF(\rho, \neg\phi_1)$ is unsatisfiable, i.e.,$\neg LF(\phi_1, \rho, \mathbf{x}_0)$ is unsatisfiable by Algorithm 3, which implies $LF(\phi_1, \rho, \mathbf{x}_0)$ is valid. By the induction hypothesis, we have $\rho \models \phi_1$. Therefore, $\rho \not\models \neg\phi_1$.

  For completeness, suppose $LF(\neg\phi_1, \rho, \mathbf{x}_0)$ is valid, which derives $LF(\phi_1, \rho, \mathbf{x}_0)$ is unsatisfiable by Algorithm 3. By the induction hypothesis, we have $\rho \not\models \phi_1$. Hence, $\rho \models \neg\phi_1$.
- $\phi = \phi_1 \vee \phi_2$
  For soundness, suppose $LF(\phi_1 \vee \phi_2, \rho, \mathbf{x}_0)$ is unsatisfiable, i.e., $LF(\phi_1, \rho, \mathbf{x}_0) \vee LF(\phi_2, \rho, \mathbf{x}_0)$ is unsatisfiable by Algorithm 3, which implies $LF(\phi_1, \rho, \mathbf{x}_0)$ and $LF(\phi_2, \rho, \mathbf{x}_0)$ both are unsatisfiable. By the induction hypothesis, it follows that $\rho \not\models \phi_1$ and $\rho \not\models \phi_2$. Therefore, $\rho \not\models \phi_1 \vee \phi_2$.

  For completeness, suppose $LF(\phi_1 \vee \phi_2, \rho, \mathbf{x}_0)$ is valid, which implies $\neg LF(\phi_1, \rho, \mathbf{x}_0)$ is unsatisfiable or $\neg LF(\phi_2, \rho, \mathbf{x}_0)$ is unsatisfiable by Algorithm 3. By the induction hypothesis, we have either $\rho \not\models \neg\phi_1$ or $\rho \not\models \phi_2$, hence $\rho \models \phi_1 \vee \phi_2$.
- $\phi = \phi_1; \phi_2$
  It is clear that $\rho \models \phi_1; \phi_2$ iff that ρ can be split into two parts ρ_1 and ρ_2 such that $\rho = \rho_1 \circ \rho_2$, and $\rho_1 \models \phi_1$ and $\rho_2 \models \phi_2$, iff there exists $0 \leq i \leq n$ such that zone z_i can be split two parts, i.e., $\langle z_i, \delta_{i1} \rangle$ and $\langle z_i, \delta_{i2} \rangle$ with $\rho_1 = \langle z_1, \delta_1 \rangle \circ \cdots \langle z_i, \delta_{i1} \rangle \models \phi_1$ and $\rho_2 = \langle z_i, \delta_{i2} \rangle \circ \cdots \langle z_n, \delta_n \rangle \models \phi_1$ with $\delta_i = \delta_{i1} + \delta_{i2}$, where δ_{i1} and δ_{i2} are fresh variables, which is exactly equivalent to

$$\forall \delta_1, \cdots, \forall \delta_n, \mathcal{Q}. \left(\begin{array}{ll} & LF(\phi_1, \langle (z_1, \delta_1), \cdots, (z_i, \delta_{i1}) \rangle, \mathbf{x}_0) \\ \wedge & LF(\langle (\phi_2, z_i, \delta_{i2}), \cdots, (z_n, \delta_n) \rangle, \\ & \qquad \mathbf{x}_0 + \sum_{j=1}^{i-1} \delta_j + \delta_{i1}) \\ \wedge & \delta_i = \delta_{i1} + \delta_{i2} \end{array} \right)$$

by Algorithm 3 and the induction hypothesis.

□

Algorithm 3: $LF(\phi, \rho, \mathbf{x}_0)$

input : $\rho = \langle(\underline{z_1}, \underline{\delta_1}), (\underline{z_2}, \underline{\delta_2}), \ldots, (\underline{z_n}, \underline{\delta_n})\rangle$, the underlined execution fragment;
$\mathbf{x}_0$, the initial value of clocks at the beginning of ρ;
ϕ, the considered ELDI formula.
output : Γ, the derived QLRA formula.

begin
  $\mathcal{Q} := \varepsilon$;
  /* $\mathcal{Q}$ records existential quantifications over introduced fresh variables */
  case $\phi := b \leq \ell \leq e \Rightarrow \mathcal{D}$
    foreach $i \in \{1, \ldots, n\}$ **do**
      foreach $j \in \{1, \ldots, d\}$ **do**
        if $z_i \models S_j$ **then**
          $e_{ij} := \delta_i$;
        else
          $e_{ij} := 0$
    $\Gamma := LP(\rho, \mathbf{x}_0) \Rightarrow (\mathcal{D}[\sum_{i=1}^{n} e_{i1}/\int S_1, \cdots, \sum_{i=1}^{n} e_{id}/\int S_d])$;
  case $\phi := \neg\phi_1$
    $\Gamma := \neg LF(\phi_1, \rho, \mathbf{x}_0)$;
  case $\phi := \phi_1 \wedge \phi_2$
    $\Gamma := LF(\phi_1, \rho, \mathbf{x}_0) \wedge LF(\phi_2, \rho, \mathbf{x}_0)$;
  case $\phi := \phi_1 \vee \phi_2$
    $\Gamma := LF(\phi_1, \rho, \mathbf{x}_0) \vee LF(\phi_2, \rho, \mathbf{x}_0)$;
  case $\phi := \phi_1; \phi_2$
    $\Gamma :=$

$$\bigvee_{i=0}^{n+1} \left(\begin{array}{ll} & LF(\phi_1, \langle(z_1, \delta_1), \cdots, (z_i, \delta_{i1})\rangle, \mathbf{x}_0) \\ \wedge & LF(\phi_2, \langle(z_i, \delta_{i2}), \cdots, (z_n, \delta_n)\rangle, \\ & \qquad \mathbf{x}_0 + \sum_{j=1}^{i-1} \delta_j + \delta_{i1}) \\ \wedge & \delta_i = \delta_{i1} + \delta_{i2} \end{array} \right)$$

    ;
    $\mathcal{Q} := \mathcal{Q}, \exists\delta_{11}, \exists\delta_{12}, \cdots, \exists\delta_{n1}, \exists\delta_{n2}$;
    /* $\delta_{11}, \delta_{12}, \ldots, \delta_{n1}, \delta_{n2}$ are fresh variables */
  return $\forall\delta_1, \cdots, \forall\delta_n, \mathcal{Q}.\Gamma$;

5 SOLVING DERIVED QLRA FORMULAS AND COMPLEXITY ANALYSIS

In this section, we further discuss how to solve the resulted QLRA formulas and the complexity of our approach.

5.1 Solving derived QLRA formulas

From Theorem 3.1 and Theorem 4.1, given a TA $\mathcal{A}$, an interval $[b, e]$ to bound the length of the execution fragments of $\mathcal{A}$, and an ELDI formula Φ, model-checking Φ is satisfied by all execution fragments of $\mathcal{A}$ whose length is within $[b, e]$ is reduced to whether a QLRA formula is valid, i.e.,

THEOREM 5.1. *Given a timed automaton $\mathcal{A}$ and an ELDI formula Φ, $\mathcal{A}, [b, e] \models \Phi$ iff $\wedge_{\rho \in PEF(ZoneG(\mathcal{A}))} LF(\Phi, \rho, \mathbf{0})$ is valid.*

PROOF. It is straightforward by Theorem 3.1 and Theorem 4.1. □

According to Tarsk's result, the satisfiability and validity of QLRA both are decidable [31], as QLRA admits the property of quantifier elimination (QE). So, an immediate result of Theorem 5.1 is that

COROLLARY 5.2. *Given a timed automaton $\mathcal{A}$ and an ELDI formula Φ, $\mathcal{A}, [b, e] \models \Phi$ is decidable.*

Tarski's original QE algorithm for real arithmetics is non-elementary [31]. But in 1970s, Collins invented a new algorithm for QE based on *cylindrical algebra decomposition* (CAD) [6], which is double exponential in the number of variables. CAD has been implemented in many computer algebra tools such as REDLOG [7], QEPCAD [5], and so on. Particularly, all formulas of QLRA are linear, therefore the QE of QLRA can be achieved more efficiently by *virtual substitution* due to Weinspfenning [8], which has been implemented in REDLOG, although the worst case is still double exponential. Additionally, according to Grigor'ev's result [15], a more efficient algorithm on QE with double exponential on the number of alternations of quantifiers could be possible, while in our case, the alternation of quantifiers in the resulted QLRA formula is at most two.

5.2 Complexity

As discussed above, the model checking of ELDIs against bounded behaviours of timed automata consists of three procedures:

- The first one is *PEF* to find out all execution fragments whose length is within the bounded interval $[b, e]$ for a given TA $\mathcal{A}$. The number of such execution fragments is at most $N^{1+N*(1+\lceil\frac{e}{\epsilon}\rceil)}$ as we discussed before, where N is the number of the zones generated from $\mathcal{A}$ and ϵ is the least dwelling time in each cycle of $\mathcal{A}$. This step can be done in $O(N^{N*(1+\lceil\frac{e}{\epsilon}\rceil)})$.
- The second one is *LF*, which translate whether a given execution fragment ρ satisfies a considered ELDI formula Φ into a QLRA formula. *LF* itself can be done in the linear of the size of Φ, but the size of the generated QLRA formula could be $O((N(1 + \lceil\frac{e}{\epsilon}\rceil))^d * |\Phi|)$ in the worst case, where d is the number of nested chops. In addition, we need to introduce $d * M$

fresh variables and the corresponding quantifications, where M is the number of zones in a bounded observation interval, $N * (1 + \lceil \frac{e}{\epsilon} \rceil)$ at most.

- The last one is QE tool, here we adopt REDLOG, whose complexity is double exponential in the number of variables, i.e., $O(2^{2^{dN*(1+\lceil \frac{e}{\epsilon} \rceil)}})$ to check whether each of such execution fragments satisfies the considered ELDI Φ.
- So, the total complexity to check whether $\mathcal{A}, [b, e] \models \Phi$ is

$$O(N^{N*(1+\lceil \frac{e}{\epsilon} \rceil)} * 2^{2^{dN*(1+\lceil \frac{e}{\epsilon} \rceil)}}). \quad (2)$$

Moreover, it is well-known that N, the number of zones of *ZoneG*($\mathcal{A}$), is exponential in n, the number of the locations of $\mathcal{A}$ [3], so the complexity of our approach is 3-fold exponential in the size of $\mathcal{A}$ and 2-fold exponential in the number of nested chops in Φ.

Although the theoretical complexity of our approach is quite high as analyzed above, in practice, the worst cases happen with quite low possibility. We believe that REDLOG can handle QLRA formulas in polynomial time in their sizes in most cases. The below experiments will justify this.

6 IMPLEMENTATION AND EXPERIMENTS

Based on the DBM library of UPPAAL, we develop a prototypical tool for our approach on Linux with two input files: the first is a .xml file to represent the timed automaton under consideration in UPPAAL format, and the other contains the ELDI formula to be verified. Our tool outputs a text file, which represents the generated QLRA formula as the input of REDLOG (Reduce), the QE tool we used. REDLOG will return *true* or *false* to the problem whether the ELDI formula is satisfied by the timed automaton on all bounded observation intervals. Besides, our tool also provides the function to check whether a given bounded execution fragment satisfies the ELDI formula. Therefore, on the one hand, the size of the

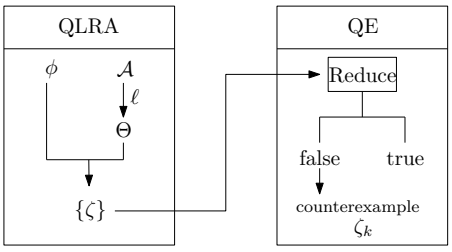


Figure 2: The overall structure

generated QLRA formula could be decreased dramatically, which can scale up our approach very much; on the other hand, it can tell which execution fragment is a counterexample as well, when model-checking the problem is done by enumerating all possible execution paths. The overall structure of our tool is depicted as Fig. 2.

The following case studies are used to illustrate the efficiency of our approach in practice, although it has a quite high theoretical complexity. The experiments are conducted on a laptop with Inter Core i3-5005U at 2.0GHz and 4GB DDR3L-1600MHz RAM.

Example 6.1. In this example, we consider the anomalous behaviour of priority-driven systems given in [24].

	r	d	$[e^-, e^+]$
J_1	0	10	5
J_2	0	10	$[2, 6]$
J_3	4	15	8
J_4	0	20	10

Figure 3: The description of jobs

The simple system contains four independent jobs, which are scheduled on two identical processors P_1 and P_2 in a priority-driven manner. P_1 and P_2 maintain a common priority order of jobs $J_1 > J_2 > J_3 > J_4$. These jobs may be preempted, but never be migrated, which means that once it begins to be executed on a processor, the job has to be executed on that processor until completion. The release times (r), deadlines (d), and execution times of the jobs are listed in Fig. 3. J_1, J_2, J_4 are released at 0, while J_3 is released at 4. The execution times of J_1, J_3, J_4 are fixed, while J_2's is varied in $[2, 6]$. In addition, J_1 is required to be executed on P_1, J_2 is required to be executed on P_2, while J_3 and J_4 can be executed on either of P_1 and P_2.

The property which should be satisfied by P_2 on $[0, 20]$ can be represented by the following ELDI formula:

$$20 \leq \ell \leq 20 \Rightarrow [(2 \leq \int run_{J_2} \leq 6 \wedge \int run_{J_2} - \int 1 = 0)$$
$$; ((\int run_{J_3} = 0 \vee \int run_{J_3} = 8) \wedge (\int run_{J_4} = 0 \vee \int run_{J_4} = 10) \wedge$$
$$0 < \int run_{J_3} + \int run_{J_4} \leq 18)].$$

The tool verifies the formula in 2.4 seconds, and returns *false*. This implies that some job cannot catch the deadline on the observation interval $[0, 20]$, a counterexample is provided in Fig. 4.

Example 6.2. Now consider an example of final testing of integrated circuits, which is a simplified version of a real-world problem reported in [27]. In this simplified version, all jobs are clustered into two product types A and B. Both of A and B jobs are processed in two stages, i.e., `testing` and `burn_in`. A and B jobs are grouped into 5 and 2 lots respectively. In `testing` stage, all A and B lots are processed serially on several parallel machines, and it takes 3

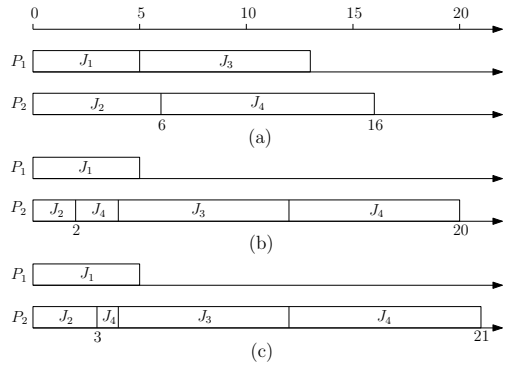
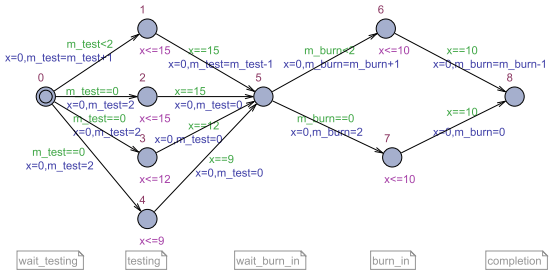
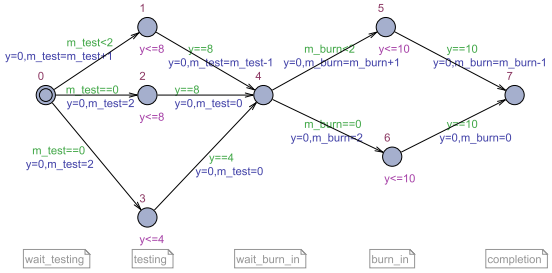


Figure 4: A counterexample



(a)



(b)

Figure 5: (a) *PA*: TA for product A; (b) *PB*: TA for product B.

time units to handle each A lot, and 4 time units to handle each B lot by one machine. In `burn_in` stage, all A and B lots are processed in a batch manner on several parallel machines, and it takes 10 time units to finish a batch by one machine. The maximum batch size for a machine is 5. For simplicity, here we assume there are two machines on each stage. So, we introduce two integer variables $m\_test \in [0, 2]$ and $m\_burn \in [0, 2]$ to stand for how many machines have been used at `testing` and `burn_in` stages, respectively. At `testing`, any machine cannot be allocated to B product (dually A product) unless all A jobs (dually B jobs) have been finished in case it has been allocated to A product (dually B product). Additionally, any type product can switch from `testing` to `burn_in` immediately whenever all its lots have been processed at `testing`. Thus, A and B products can be modelled by TA *PA* and *PB* in Fig. 5.(a) and Fig. 5.(b) correspondingly.

PA have nine locations $0, 1, 2, 3, 4, 5, 6, 7, 8$. 0 stands for waiting for testing, 1 for one machine allocated to test A product, $2, 3, 4$ for both of the two machines allocated to test A product, but with different scheduling policies. In 2, one machine has 5 lots and the other has no jobs; in 3, one machine has 4 lots and the other has 1 lot; in 4, one machine has 3 lots and the other has 2 lots. 5 for waiting for burning, $6, 7$ for A lots under burn respectively with one machine and two the machines, 8 for completion. *PB* can be understood similarly. Note that there are only two scheduling policies in *PB* at `testing` when both of the two machines are allocated to it. x and y are two clock variables. Fig. 6 is the product of *PA* and *PB*. The timing constraints on the products A and B are given as follows: The deadlines for A and B are 30 and 36, respectively. Now, the question is wether we can find a feasible schedule to meet these requirements within 36 to 40 time units, i.e., we need to check whether the following

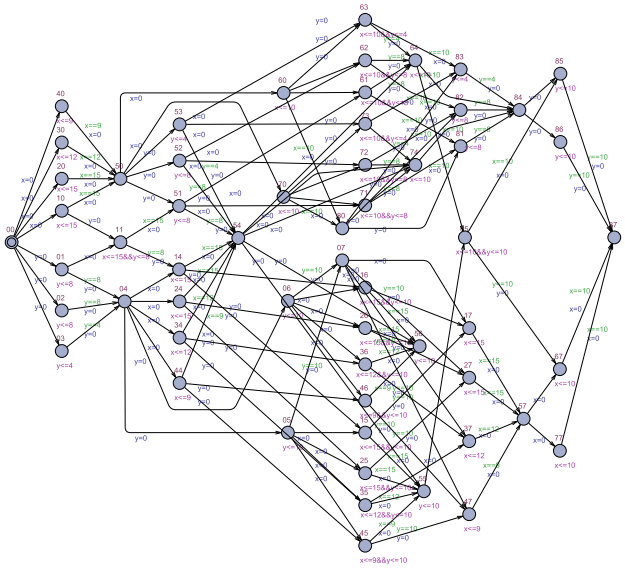


Figure 6: *PA* $\times$ *PB*, the product of PA and PB

formula holds.

$$36 \le \ell \le 40 \Rightarrow$$

$$\left(\begin{array}{l} \int(PA.wait\_testing, *) + \int(PA.testing, *) + \\ \int(PA.wait\_burn\_in, *) + \int(PA.burn\_in, *) \le 30 \\ \wedge [9 \le \int(PA.testing, *) \le 15; \int(PA.burn\_in, *) = 10] \end{array}\right)$$

$$\wedge \left(\begin{array}{l} \int(*, PB.wait\_testing) + \int(*, PB.testing) + \\ \int(*, PB.wait\_burn\_in) + \int(*, PB.burn\_in) \le 36 \\ \wedge [4 \le \int(*, PB.testing) \le 8; \int(*, PB.burn\_in) = 10] \end{array}\right)$$

where $(PA.l, *)$ stands for a state expression, which means that in $PA \times PB$, the component PA stays in location l at time t if $(PA.l, *)(t) = 1$, $(*, PB.l)$ can be understood symmetrically.

By checking the above property with our tool, which takes 83.9 seconds totally, we find several feasible schedules by solving the resulting QLRA formula. For example, in Fig. 6, the path $00 \to 10 \to 11 \to 14 \to 54 \to 55 \to 65 \to 67 \to 87$ with chop points at 54 is a feasible schedule.

7 CONCLUSION

In this paper, we investigate the model-checking of continuous-time extended linear duration invariants against bounded execution fragments of timed automata. This is achieved through the following steps: firstly, we compute all execution fragments of a given timed automaton $\mathcal{A}$ whose length is within the given bounded interval according to $\mathcal{A}$'s zone graph; secondly, we encode whether each execution fragment obtained in the first step satisfies the given ELDI formula ϕ into a QLRA formula; finally, we invoke REDLOG, a computer algebra tool, to solve the resulting QLRA formula. The complexity of our approach is 3-fold exponential in the number of $\mathcal{A}$'s locations in the worst case. But in practice, it is very efficient as in REDLOG, *virtual substitution* can be applied to QLRA formulas.

We have implemented a prototypical tool and some case studies are provided to illustrate our approach, which can be found at https://github.com/Leslieaj/VCELDI.

REFERENCES

[1] R. Alur and D. L. Dill. 1994. A theory of timed automata. *TCS* 126(2) (1994), 183–235.

[2] R. E. Bellman. 1957. *Dynamic Programming*. Princeton University Press.

[3] J. Bengtsson and Y. Wang. 2004. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. 87–124.

[4] V. A. Braberman and D. V. Huang. 1998. On checking timed automata for linear duration invariants. In *RTSS 1998*. 264 – 273.

[5] C. W. Brown. 2003. QEPCAD B: A Program for Computing with Semi-algebraic Sets Using CADs. *ACM SIGSAM Bulletin* 37, 4 (2003), 97–108.

[6] G. Collins. 1975. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *2nd GI Conference on Automata Theory and Formal Languages*. 134–183.

[7] A. Dolzmann, A. Seidl, and T. Sturm. 2006. *Redlog User Manual* (Edition 3.1, for Redlog Version 3.06 (Reduce 3.8) ed.).

[8] A. Dolzmann, T. Sturm, and V. Weispfenning. 1998. A New Approach for Automatic Theorem Proving in Real Geometry. *J. of Automated Reasoning* 21, 3 (1998), 357–380.

[9] M. Franchella. 1997. On the origins of Dénes König's infinity lemma. *Archive for History of Exact Sciences* 51, 1 (1997), 3–27.

[10] M. Fränzle. 2004. Model-checking dense-time Duration Calculus. *Formal Aspects of Computing* 16, 2 (2004), 121–139.

[11] M. Fränzle and M. R. Hansen. 2007. Deciding an Interval Logic with Accumulated Durations. In *TACAS 2007*. 201–215.

[12] M. Fränzle and M. R. Hansen. 2008. Efficient model checking for duration calculus based on branching-time approximations. In *SEFM 2008*. 63–72.

[13] M. Fränzle and M. R. Hansen. 2009. Efficient model checking for duration calculus. *International Journal of Software and Informatics* 3, 2-3 (2009), 171–196.

[14] V. Goranko, A. Montanari, and G. Sciavicco. 2004. A road map of interval temporal logics and duration calculi. *J. of Applied Non-Classical Logics* 14, 1-2 (2004), 9–54.

[15] D. Y. Grigor'ev. 1988. The complexity of deciding Tarski algebra. *J. of Symb. Comp.* 5, 1-2 (1988), 65–108.

[16] J. Y. Halpern, Z. Manna, and B. C. Moszkowski. 1983. A Hardware Semantics Based on Temporal Intervals. In *ICALP 1983*. 278–291.

[17] M. R. Hansen. 1994. Model-checking Discrete Duration Calculus. *Formal Aspects of Computing* 6, 1 (1994), 826–845.

[18] T. A. Henzinger. 1996. The theory of hybrid automata. In *LICS 1996*. 278–292.

[19] C. Zhou. C. A. R. Hoare and A. P. Ravn. 1991. A calculus of durations. *Inf. Proc. Let.* 40, 5 (1991), 269–276.

[20] D. König. 1927. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math.* 3, 2-3 (1927), 121–130.

[21] K. G. Larsen, P. Pettersson, and Y. Wang. 1997. Uppaal in a nutshell. *STTT* 1, 1 (1997), 134–152.

[22] X. Li. and D. V. Huang. 1996. Checking linear duration invariants by linear programming. In *ASIAN 1996*. 321–332.

[23] X. Li, D. V. Huang, and T. Zheng. 1997. Checking hybrid automata for linear duration invariants. In *ASIAN 1997*. 166–180.

[24] J. W. S. Liu. 2000. *Real-Time Systems*. Prentice Hall.

[25] R. Meyer, J. Faber, J. Hoenicke, and A. Rybalchenko. 2008. Model checking Duration Calculus: a practical approach. *Formal Aspects of Computing* 20, 4 (2008), 481–505.

[26] P. K. Pandya. 2001. Specifying and deciding quantified discrete-time duration calculus formulae using DCVALID. In *RT-TOOLS 2001*.

[27] W. L. Pearn, S. H. Chung, A. Y. Chen, and M. H. Yang. 2004. A case study on the multistage IC final testing scheduling problem with reentry. *International Journal of Production Economics* 88, 3 (2004), 257 – 267.

[28] P. Pettersson. 1999. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice.* PhD thesis. Uppsala University.

[29] T. G. Rokicki. 1993. *Representing and Modeling Digital Circuits*. PhD thesis. Stanford University.

[30] B. Sharma, P. K. Pandya, and S. Chakraborty. 2005. Bounded Validity Checking of Interval Duration Logic. In *TACAS 2005*. 301–316.

[31] A. Tarski. 1951. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley.

[32] P. H. Thai and D. V. Hung. 2004. Verifying linear duration constraints of timed automata. In *ICTAC 2004*. 295–309.

[33] M. Zhang, D. V. Hung, and N. Zhan. 2008. Verification of linear duration invariants by model checking CTL properties. In *ICTAC 2008*. 395–409.

[34] M. Zhang, Z. Liu, and N. Zhan. 2009. Model checking linear duration invariants of networks of automata. In *FSEN 2009*. 244–259.

[35] C. Zhou and M. R. Hansen. 2004. *Duration Calculus: A Formal Approach to Real-Time Systems*. Springer.

[36] C. Zhou, M. R. Hansen, and P. Sestoft. 1993. Decidability and undecidability results for duration calculus. In *STACS 1993*. 58–68.

[37] C. Zhou, J. Zhang, L. Yang, and X. Li. 1994. Linear duration invariants. In *FTRTFT 1994*. 86–109.

[38] Q. Zu, M. Zhang, J. Zhu, and N. Zhan. 2013. Bounded model-checking of discrete duration calculus. In *HSCC 2013*. 213–222.