SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2018, Vol. 61 052105:1–052105:17
https://doi.org/10.1007/s11432-017-9133-4

Anti-chain based algorithms for timed/probabilistic refinement checking

Ting WANG[1], Tieming CHEN[1*] , Yang LIU[2] & Ye WANG[3]

[1]*College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou* 310000*, China;*
[2]*School of Computer Engineering, Nanyang Technological University, Singapore* 639798*, Singapore;*
[3]*School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou* 310000*, China*

Received 22 May 2017/Accepted 9 June 2017/Published online 6 December 2017

Abstract Refinement checking answers the question on whether an implementation model is a refinement of a specification model, which is of great value for system verification. Some refinement relationships, e.g., trace refinement and failures/divergence refinement, have been recognized for different verification purposes. In general, refinement checking algorithms often rely on subset construction, which incurs in the state space explosion problem. Recently the anti-chain based approach has been suggested for trace refinement checking, and the results show a significant improvement. In this paper, we investigate the problems of applying the anti-chain approach to timed refinement checking (a timed implementation vs. a timed or untimed specification) and probabilistic refinement checking (a probabilistic implementation vs. a non-probabilistic specification), and show that the state space can be reduced considerably by employing the anti-chain approach. All the algorithms have been integrated into the model checking tool PAT, and the experiments have been conducted to show the efficiency of the application of anti-chains.

Keywords model checking, refinement, anti-chain, timed automata, markov decision process

Citation Wang T, Chen T M, Liu Y, et al. Anti-chain based algorithms for timed/probabilistic refinement checking. Sci China Inf Sci, 2018, 61(5): 052105, https://doi.org/10.1007/s11432-017-9133-4

1 Introduction

Formal methods take advantage of a broad variety of mathematical theories to carry out the specification and verification of software and hardware systems in computer science [1–3], and have been applied to many real-life problems [4, 5]. Model checking is a powerful formal approach to verify the properties of finite state models, which in a nutshell solves the problem described as follows: given a system model and a property, automatically and exhaustively check if the model satisfies the property. One particularly popular setting for model checking is that the property is given as a temporal logic formula (e.g., LTL or CTL) which is to be expressed in a different language from the system modeling language. Another technique is often referred to as refinement checking [6], which has been traditionally adopted for the verification of CSP. The notion of refinement is a particularly useful concept in many engineering activities. Refinement checking examines if there exists a refinement relationship between the implementation model and the specification model both of which are in the same language. Suppose that the specification satisfies a certain property of the system, we can infer that the implementation also satisfies that property if there is a property preserving refinement relationship between them.

* Corresponding author (email: tmchen@zjut.edu.cn)

The success of the verification tool FDR for CSP [1] evidences the usefulness of refinement checking. A variety of refinement relationships have been well recognized to represent properties with different purposes. For instance, trace refinement can be used to verify safety, while failures/divergence refinement can be used to verify not only safety but also some classes of liveness properties [7]. However, there is not much work on the refinement checking for timed and probabilistic systems.

The work in [8] has shown that trace refinement relationship can be extended to the verification of probabilistic systems. A probabilistic implementation model (i.e., a Markov decision process) exhibits a trace with a certain probability. Therefore, given a non-probabilistic specification model (a finite state automaton) that captures desired system behaviours, we can calculate the probability that the implementation model correctly executes the behaviours (i.e., the traces) of the specification model. Intuitively, it can be seen as the probability that the implementation model behaves well according to the specification model. In this work, this is referred to as probabilistic refinement checking.

Naturally, similar extensions can also be applied to timed systems. That is, given a timed implementation model (with the semantics of timed automata) and an untimed specification model (a finite state automaton), we check whether the implementation exhibits the traces of the specification under some timed conditions. It is useful since we can verify whether a timed system can execute the same behaviours represented by the specification model. Moreover, a more complicated class of refinement checking for timed systems is considered in this work, which is inspired by the work in [9–11]. That is, the specification is also a timed model, and thus a timed trace refinement relationship is established between the implementation model and the specification model, which enables us to verify timed properties. To the best of our knowledge, there has not been any complete study of refinement checking approach for timed systems.

For the aforementioned refinement checking with an untimed/non-probabilistic specification, the standard (and most popular) approach is to first determinize the specification that is usually a NFA (nondeterministic finite-state automaton) using the subset construction, and then check for reachability in the synchronous product of the implementation and specification. In the worst case, the subset construction could construct a DFA which is exponentially larger than the original NFA, which leads to the state space explosion. In order to alleviate this problem, Wulf et al. [12] suggested the anti-chain method which can be used in trace refinement checking. It was shown that this approach outperformed the standard algorithm. The original definition of anti-chain is a subset of a partially ordered set such that any two elements in the subset are incomparable. Anti-chain allows us to store only a 'maximal' subset of states in the product. As a result, the complete subset construction can be avoided and we do not need to construct the whole product. It will be shown in Sections 2–4 that all the refinement checking with an untimed (and non-probabilistic) specification is based on subset construction.

For timed refinement checking with a timed specification, the specification can also be determinized. The determinization procedure for timed automata has been proposed in [10], which is much more complicated than for NFAs since timed systems transit from one state to another with timed conditions. The determinization requires the transitions labeled with clock constraints to be mutually exclusive, and it can be seen as a subset construction with time. The method of [10] is impractical since it is based on region abstraction which is known to be very inefficient [13]. Our previous work [14] supplies a construction based on a zone abstraction which is a more succinct representation for the verification [13]. However, the construction is complicated, i.e., the specification is transformed into an infinite timed tree before determinization. Therefore, in this work we improve the construction and use a straightforward determinization without transforming the specification.

It has been shown that not all timed automata can be determinized since in some cases the clocks are added infinitely often which leads to an infinite state space. To avoid this problem, the work in [11] limits the timed automata to have only one clock and proves that the language inclusion problem for one-clock timed automata is decidable. It should be noted that anti-chain is the key for proving the decidability of this problem. However, in this work we prefer a method which can be applied to arbitrary timed automata. Then it is worthwhile to study the role of anti-chain in this settings.

For this work, we study the timed refinement and probabilistic refinement checking. The contributions

are summarized as follows. Firstly, we first define timed refinement checking with an untimed specification, and further show that anti-chain improves the performance. Secondly, for the timed refinement checking with a timed specification, we use an algorithm with a straightforward determinization which is an enhancement of our previous work and show that anti-chain can not only contribute to the termination of the algorithm, but also reduce the state space significantly. Thirdly, we first apply anti-chain to probabilistic refinement checking with a non-probabilistic specification, and show improvements in some cases. Lastly, we implement all the algorithms in the tool PAT [15].

Organization. Section 2 reviews the foundations of anti-chain based refinement checking. Section 3 shows the anti-chain algorithms for the timed refinement checking. Section 4 illustrates that anti-chain can also contribute to probabilistic refinement checking. Section 5 reports the experimental results. Section 6 gives the related work. Lastly, Section 7 concludes the paper.

2 Foundations of anti-chain based refinement checking

Let Σ be a set of observable events, τ be a hidden event, and $\text{Act} = \Sigma \cup \{\tau\}$. A Labeled Transition System (LTS) is a tuple $\mathcal{L} = (S, \text{Init}, \text{Act}, T)$, where S is a set of locations, $\text{Init} \subseteq S$ is a set of initial locations, and $T \subseteq S \times \text{Act} \times S$ is a transition relation.

We define $s \xrightarrow{e} s'$ if $(s, e, s') \in T$ and $e \in \Sigma$. We define $s \overset{\tau}{\rightsquigarrow} s'$ if $\mathcal{L}$ contains a sequence $\langle s_0, s_1, \ldots, s_n \rangle$ where $(s_i, \tau, s_{i+1}) \in T$ for all $0 \leqslant i < n$ and $s = s_0$ and $s' = s_n$. We write $s_1 \overset{e}{\rightsquigarrow} s_2$ if $s_1 \overset{\tau}{\rightsquigarrow} s'$ and $s' \xrightarrow{e} s''$ and $s'' \overset{\tau}{\rightsquigarrow} s_2$. If there exists a sequence $\langle s_0, s_1, \ldots, s_n \rangle$ where $s_i \overset{e_i}{\rightsquigarrow} s_{i+1}$ for all $0 \leqslant i < n$ and $s_0 \in \text{Init}$, the finite sequence of events $\langle e_0, e_1, \ldots, e_{n-1} \rangle$ is a trace in $\mathcal{L}$. The set of all the traces in $\mathcal{L}$ is denoted by $\text{traces}(\mathcal{L})$.

Definition 1 (Trace refinement)**.** Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two LTSs. $\mathcal{L}_1$ refines $\mathcal{L}_2$ in trace semantics iff $\text{traces}(\mathcal{L}_1) \subseteq \text{traces}(\mathcal{L}_2)$.

According to the subset construction for trace refinement checking [1], the LTS as the specification can be reconstructed as a trace-equivalent (to the original LTS) and deterministic one without τ events. The determinized LTS of $\mathcal{L}$ is denoted by $\det(\mathcal{L}) = (S', \text{Init}', \text{Act}', T')$, such that $S' \subseteq 2^S$, $\text{Init}' = \{s' \in S \mid \exists i \in \text{Init}.\ i \overset{\tau}{\rightsquigarrow} s'\}$, $\text{Act}' = \text{Act} \setminus \{\tau\}$, and T' is the transition relation such that $(N_d, e, N'_d) \in T_d$ iff $N'_d = \{s' \in S \mid \exists s \in N_d.\ s \overset{e}{\rightsquigarrow} s'\}$.

Definition 2 (Synchronous product of LTSs)**.** Let $\mathcal{L}_1 = (S_1, \text{Init}_1, \text{Act}_1, T_1)$, $\mathcal{L}_2 = (S_2, \text{Init}_2, \text{Act}_2, T_2)$ be two LTSs. Given $\det(\mathcal{L}_2) = (S'_2, \text{Init}'_2, \text{Act}'_2, T'_2)$, the synchronous product of $\mathcal{L}_1$ and $\det(\mathcal{L}_2)$, written as $\mathcal{L}_1 \times \det(\mathcal{L}_2)$, is the LTS $\mathcal{L} = (S, \text{Init}, \text{Act}, T)$ where $S = S_1 \times S'_2$; $\text{Init} = \{(i_1, i_2) \mid i_1 \in \text{Init}_1 \wedge i_2 = \text{Init}'_2\}$; $\text{Act} = \text{Act}_1 \cup \text{Act}'_2$; and T is the transition relation such that for all (s, N) with $s \in S_1$ and $N \in S'_2$,

- if $(s, \tau, s') \in T_1$, then $((s, N), \tau, (s', N)) \in T$;
- if $(s, e, s') \in T_1$ and $(N, e, N') \in T'_2$, then $((s, N), e, (s', N')) \in T$.

The trace refinement checking tries to find a target state (s, N) in $\mathcal{L}_1 \times \det(\mathcal{L}_2)$ such that N is the empty set, which is named as a TR-witness state in this paper (TR: trace refinement).

Given $s \in S$ and $s' \in S$, we say that s simulates s', denoted $s' \prec s$, iff for any $s' \overset{e}{\rightsquigarrow} s'_1$, there exists $s \overset{e}{\rightsquigarrow} s_1$ and $s'_1 \prec s_1$. The anti-chain based approach actually infers a "simulation" relation in $\mathcal{L}_1 \times \det(\mathcal{L}_2)$. Given any two product states (s_1, N_1) and (s_2, N_2) of $\mathcal{L}_1 \times \det(\mathcal{L}_2)$, let $(s_2, N_2) \preceq (s_1, N_1)$ denote $s_1 = s_2$ and for any $n_1 \in N_1$, there is $n_2 \in N_2$ where $n_1 \prec n_2$. If $(s_2, N_2) \preceq (s_1, N_1)$ and $((s_1, N_1), e, (s'_1, N'_1))$ is a transition in the product, then there exists (s'_2, N'_2) such that $((s_2, N_2), e, (s'_2, N'_2))$ is also a transition in the product and $(s'_2, N'_2) \preceq (s'_1, N'_1)$. As a result, we have the following lemma.

Lemma 1 ([16])**.** Let (s_1, N_1) and (s_2, N_2) be two product states in $\mathcal{L}_1 \times \det(\mathcal{L}_2)$. If $(s_2, N_2) \preceq (s_1, N_1)$, then a TR-witness state is reachable from (s_2, N_2) implies a TR-witness state is reachable from (s_1, N_1).

As a result, we can skip (s_2, N_2) if (s_1, N_1) has been explored. Only the "maximal" states like (s_1, N_1) are stored, which constitute the Anti-chain subset.

3 Timed refinement checking

3.1 Background: timed automata and zone abstraction

In this paper, timed safety automata are the focus [17] (simply referred to as timed automata), which are often used to model timed systems in practice. Suppose that there is a clock set C, let $\Phi(C)$ be the set of clock constraints. A clock constraint is inductively defined as $\delta := \text{true} \mid x \sim d \mid \delta_1 \wedge \delta_2$ where $\sim \in \{=, \leqslant, \geqslant, <, >\}$; x is a clock in C and d is an integer. $\Phi_{\leqslant,<}(C)$ is obtained with $\sim \in \{\leqslant, <\}$.

Definition 3 (Timed automata)**.** A Timed Automaton (TA) is denoted by a tuple $\mathcal{A} = (S, \text{Init}, \Sigma, C, L, T)$, such that S is a set of locations, Init $\subseteq S$ is a set of initial locations, C is a finite set of clocks, Σ is a set of visible events, $L : S \to \Phi_{\leqslant,<}(C)$ is a function that maps an invariant to each location, and $T \subseteq S \times \Sigma \times \Phi(C) \times 2^C \times S$ is a transition relation.

The transition $(s_1, e, \delta, Y, s_2) \in T$ is a jump from s_1 to s_2 where the clock constraint δ and the location invariant $L(s_2)$ are satisfied. After that, the clocks in Y are set to zero. A timed automaton $\mathcal{A}$ is deterministic iff Init contains only one location and for any two transitions $(s_0, e_0, \delta_0, Y_0, s_0') \in T$ and $(s_1, e_1, \delta_1, Y_1, s_1') \in T$, if $s_0 = s_1$ and $e_0 = e_1$, then δ_0 and δ_1 are mutually exclusive.

A clock valuation is a map $v : C \to R^+$. Let $d \in R^+$, for any $t \in C$, $v + d$ represents the clock valuation v' where $v'(t) = v(t) + d$. For a clock set $Y \subseteq C$, $[Y \mapsto 0]v$ represents the v' where $v'(t) = v(t)$ for any $t \in C \wedge t \notin Y$, and $v'(y) = 0$ for any $y \in Y$. Let $C = 0$ be the clock valuation where each clock $t \in C$ equals to 0.

A concrete configuration in $\mathcal{A}$ is a pair denoted by (s, v) such that $s \in S$, v is a clock valuation and $v \models L(s)$. Given a timed event (d, e) (where $d \in R^+$ is a duration and e is an event), a concrete transition of $\mathcal{A}$ is denoted by $((s, v), (d, e), (s', v'))$, where a transition $(s, e, \delta, Y, s') \in T$ exists; $v + d \models \delta$; $v + d \models L(s)$; $[Y \mapsto 0](v + d) = v'$; and $v' \models L(s')$. Given a concrete configuration (s, v) and a timed event (d, e), we define $\text{post}((s, v), (d, e), \mathcal{A}) = \{(s', v') \mid ((s, v), (d, e), (s', v'))$ is a concrete transition of $\mathcal{A}\}$. For a set of concrete configurations X, we define $\text{post}(\text{Conf}, (d, e), \mathcal{A}) = \{(s', v') \mid \exists (s, v) \in X.\ ((s, v), (d, e), (s', v'))$ is a concrete transition of $\mathcal{A}\}$.

A run of $\mathcal{A}$ is a finite sequence $\langle (s_0, v_0), (d_1, e_1), (s_1, v_1), (d_2, e_2), \ldots (s_n, v_n)\rangle$ where $((s_i, v_i), (d_{i+1}, e_{i+1}), (s_{i+1}, v_{i+1}))$ is a concrete transition of $\mathcal{A}$ for any $0 \leqslant i < n$. Then we can obtain a timed trace which is a sequence of timed events: $\langle (d_1, e_1), (d_2, e_2), \ldots, (d_n, e_n)\rangle$. We define $\text{tmtraces}(\mathcal{A}, (s, v))$ to be the set of timed traces obtained from all runs starting with (s, v). We also define $\text{tmtraces}(\mathcal{A})$ as the timed traces obtained from any run of $\mathcal{A}$ starting from $\{(s, C = 0) \mid s \in \text{Init}\}$. If two timed automata have the same set of timed traces, they are equivalent. Given the above sequence $\langle (s_0, v_0), (d_1, e_1), (s_1, v_1), (d_2, e_2), \ldots (s_n, v_n)\rangle$, we define the finite sequence of events $\langle e_1, e_2, \ldots, e_n\rangle$ as an untimed trace of $\mathcal{A}$. The set of all the untimed traces of $\mathcal{A}$ is denoted by $\text{traces}_{\text{TA}}(\mathcal{A})$.

Zone abstraction is an effective technique for model checking Timed Automata [13]. Given a clock constraint δ, let $\delta^{\uparrow}$ denote the zone (which is also a clock constraint) reached by delaying by an arbitrary amount of time. We write $v \in \delta$ iff the clock valuation v is evaluated to be true with δ. For a set $Y \subseteq C$, $[Y \mapsto 0]\delta$ denotes the zone by setting the clocks in Y to 0; and let $\delta[Y]$ denote the projection of δ on Y. The result of zone abstraction is a zone graph, the definition of which is shown as below.

Definition 4 (Zone graph)**.** Given a timed automaton $\mathcal{A} = (S, \text{Init}, \Sigma, C, L, T)$, the zone graph of $\mathcal{A}$, denoted by $\text{ZG}(\mathcal{A})$, is the tuple $(S_z, \text{Init}_z, \Sigma, T_z)$ such that

- $S_z = \{(s, \delta) \mid s \in S \wedge \delta$ is a clock constraint$\}$;
- $\text{Init}_z = \{(\text{init}, (C = 0)^{\uparrow} \wedge L(\text{init})) \mid \text{init} \in \text{Init}\}$ is a set of initial nodes;
- $T_z \subseteq S_z \times \Sigma \times S_z$ is a transition relation such that for all $((s_1, \delta_1), e, (s_2, \delta_2)) \in T_z$ iff $(s_1, e, \delta_T, Y_T, s_2) \in T$, $\delta_1 \wedge \delta_T$ is not empty, $[Y_T \mapsto 0](\delta_1 \wedge \delta_T) \wedge L(s_2)$ is not empty and $\delta_2 = \mathcal{N}(([Y_T \mapsto 0](\delta_1 \wedge \delta_T))^{\uparrow} \wedge L(s_2))$.

Note that $\mathcal{N}$ [1)] is a zone normalization function, which is to make sure that the total number of zones is

1) For any clock $c \in C$, let ceil(c) be the clock ceiling obtained from $\mathcal{A}$. $\mathcal{N}(\delta)$ can be computed by $(y \in C)$: (1) removing all constraints of the form $c < m$, $c \leqslant m$, $c - y < m$, $c - y \leqslant m$ where $m >$ ceil(c); (2) replacing all constraints of the form $c > m$, $c \geqslant m$, $c - y > m$, $c - y \geqslant m$ where $m >$ ceil(c) with $c >$ ceil(c) and $c - y >$ ceil(c) respectively. Therefore, the zones that may contain arbitrarily large constants can be transformed to a unique representation whose constants are bounded by clock ceilings.

finite [18,19]. An abstract run of $\mathrm{ZG}(\mathcal{A})$ is a finite sequence: $\pi_z = \langle (s_0, \delta_0), e_0, (s_1, \delta_1), e_1, (s_2, \delta_2), e_2, \ldots, e_{n-1}, (s_n, \delta_n)\rangle$ such that $(s_0, \delta_0) \in \mathrm{Init}_z$ and $((s_i, \delta_i), e_i, (s_{i+1}, \delta_{i+1})) \in T_z$ for all $0 \leqslant i < n$. A concrete run $\langle (s_0, v_0), (d_0, e_0), (s_1, v_1), (d_0, e_0), \ldots, (d_{n-1}, e_{n-1}), (s_n, v_n)\rangle$ of $\mathcal{A}$ is an instance of π_z if $v_i \in \delta_i$ for all $0 \leqslant i \leqslant n$. Zone graph preserves reachability and linear properties [20]. We can obtain a finite sequence of events $\langle e_0, e_2, \ldots, e_{n-1}\rangle$ from π_z. All these untimed sequences of $\mathrm{ZG}(\mathcal{A})$ are denoted by $\mathrm{traces}_{\mathrm{ZG}}(\mathcal{A})$. It is easy to show that $\mathrm{traces}_{\mathrm{ZG}}(\mathcal{A})$ equals to $\mathrm{traces}_{\mathrm{TA}}(\mathcal{A})$.

3.2 Trace refinement checking for timed automata

3.2.1 *Definitions of trace refinement*

We give the definition of the refinement checking between a timed automaton and an LTS as below.

Definition 5 (TA-Trace refinement)**.** Let $\mathcal{A}$ be a timed automaton, $\mathcal{L}$ be an LTS. $\mathcal{A}$ refines $\mathcal{L}$ in trace semantics iff $\mathrm{traces}_{\mathrm{TA}}(\mathcal{A}) \subseteq \mathrm{traces}(\mathcal{L})$.

From the definition, intuitively it is to decide whether $\mathcal{A}$ can exhibit a trace of $\mathcal{L}$ under the timed constraints. Since $\mathrm{traces}_{\mathrm{ZG}}(\mathcal{A})$ equals to $\mathrm{traces}_{\mathrm{TA}}(\mathcal{A})$, we can reduce this problem to decide whether $\mathrm{traces}_{\mathrm{ZG}}(\mathcal{A}) \subseteq \mathrm{traces}(\mathcal{L})$, and build the synchronous product of a timed automaton and an LTS with the zone abstraction.

Definition 6 (Synchronous product of TA and LTS)**.** Given a timed automaton $\mathcal{A} = (S_a, \mathrm{Init}_a, \Sigma_a, C_a, L_a, T_a)$ and an LTS $\mathcal{L} = (S_l, \mathrm{Init}_l, \mathrm{Act}_l, T_l)$, the determinized LTS of $\mathcal{L}$ is denoted by $\det(\mathcal{L}) = (S'_l, \mathrm{Init}'_l, \mathrm{Act}'_l, T'_l)$. The synchronous product of $\mathcal{A}$ and $\det(\mathcal{L})$ is a zone graph $\mathrm{ZG}_{\mathrm{AL}} = (S, \mathrm{Init}, \Sigma_a, T)$ as follows.

- An element in S is an abstract configuration in the form of (s, δ, W) such that s is a location in S_a, δ is a clock constraint on C_a and W is a state in S'_l.
- $\mathrm{Init} = \{(\mathrm{init}_a, (C_a = 0)^{\uparrow} \wedge L_a(\mathrm{init}_a), \mathrm{Init}'_l) \mid \mathrm{init}_a \in \mathrm{Init}_a\}$.
- $T : S \times \Sigma \times S$ is a transition relation such that $((s_1, \delta_1, W_1), e, (s_2, \delta_2, W_2)) \in T$ iff the following conditions are satisfied: (1) $(s_1, e, \delta_a, Y_a, s_2) \in T_a$, $\delta_1 \wedge \delta_a$ is not empty, $[Y_a \mapsto 0](\delta_1 \wedge \delta_a)^{\uparrow} \wedge L(s_2)$ is not empty and $\delta_2 = \mathcal{N}([Y_a \mapsto 0](\delta_1 \wedge \delta_a)^{\uparrow} \wedge L(s_2))$; (2) $(W_1, e, W_2) \in T'_l$.

We denote the successors of an abstract configuration ps in $\mathrm{ZG}_{\mathrm{AL}}$ as $\mathrm{post}(\mathrm{ps}, \mathrm{ZG}_{\mathrm{AL}})$.

Theorem 1. Given a timed automaton $\mathcal{A}$ and an LTS $\mathcal{L}$, $\mathrm{traces}_{\mathrm{ZG}}(\mathcal{A}) \subseteq \mathrm{traces}(\mathcal{L})$ iff there is no reachable state $(s, \delta, \emptyset)$ in $\mathrm{ZG}_{\mathrm{AL}}$.

The abstract configuration $(s, \delta, \emptyset)$ is called a TATR-witness state (TATR: timed automata trace refinement).

3.2.2 *Algorithm based on anti-chains for trace refinement checking*

Next, we show how to apply two kinds of anti-chains to this problem to reduce the state space. First, the lower-upper bounds simulation reduction method from [21] is used. Two functions Lw and Up are defined as follows. Given a clock x and a state st in $\mathcal{A}$, a depth-first-search is used to gather all the transitions that are reachable from st until meeting a transition which resets x. Then, $\mathrm{Lw}(\mathrm{st}, x)$ is set to be the maximal constant c where a constraint $x > c$ or $x \geqslant c$ exists in these transitions; $\mathrm{Up}(\mathrm{st}, x)$ is set to be the maximal constant c where a constraint $x < c$ or $x \leqslant c$ exists in these transitions. If such a c does not exist, $\mathrm{Lw}(\mathrm{st}, x)$ or $\mathrm{Up}(\mathrm{st}, x)$ is set to be $-\infty$.

With Lw and Up, a relation between two zones is defined. Given two clock valuations v and v' in (s, δ, W), $v \prec_{\mathrm{LU}} v'$ represents that if for any clock x, either $v'(x) = v(x)$ or $\mathrm{Lw}(s, x) < v'(x) < v(x)$ or $\mathrm{Up}(s, x) < v(x) < v'(x)$. Then, for two zones δ and δ', $\delta \prec_{\mathrm{LU}} \delta'$ represents that for any $v \models \delta$, there exists $v' \models \delta'$ where $v \prec_{\mathrm{LU}} v'$. If (s, δ, W) is searched, it can be replaced by (s, δ', W) if $\delta \prec_{\mathrm{LU}} \delta'$ where the zone δ is expanded to δ', and the reachability analysis is still preserved. This is an anti-chain since only the "maximal" elements are kept. For a state $\mathrm{ps} = (s, \delta, W)$, we use $\mathrm{LU}(\mathrm{ps})$ to denote the above method.

Another kind of anti-chain is based on the direct comparison of two zones. Given two clock constraints δ and δ' on the same set of clocks, we define $\delta \subseteq \delta'$ iff $v \models \delta$ implies $v \models \delta'$. We use $(s, \delta, W) \lesssim (s, \delta', W')$ to denote $\delta \subseteq \delta'$ and $W' \subseteq W$.

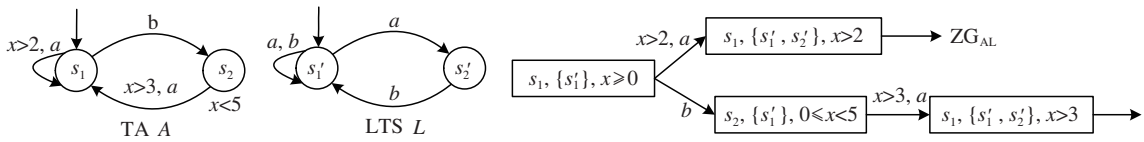

Figure 1 Synchronous product of a TA and an LTS.

Lemma 2. Let (s, δ, W) and (s, δ', W') be two abstract configurations in $\mathrm{ZG_{AL}}$. If $(s, \delta, W) \lesssim (s, \delta', W')$, then a TATR-witness state is reachable from (s, δ, W) implies a TATR-witness state is reachable from (s, δ', W').

The anti-chain based trace refinement checking algorithm for timed automata, denoted by Algorithm 1, constructs $\mathrm{ZG_{AL}}$ on-the-fly with reachability analysis using anti-chain. Lines 5 and 12 ensure that anti is an anti-chain. The following theorem states that the algorithm always produces correct results.

Theorem 2. Algorithm 1 returns true iff $\mathrm{traces_{ZG}}(\mathcal{A}) \subseteq \mathrm{traces}(\mathcal{L})$.

Algorithm 1 Anti-chain based trace refinement checking for timed automata

```
1:  Let working := Init;
2:  Let anti := ∅;
3:  while working ≠ ∅ do
4:      remove ps := (s, δ, W) from working;
5:      remove all ps' ∈ anti  s.t. ps' ≲ ps;
6:      add ps into anti;
7:      if W = ∅ then
8:          return false;
9:      end if
10:     for all (s', δ', W') ∈ post(ps, ZG_AL) do
11:         ps'' := LU((s', δ', W'));
12:         if ∄ ps' ∈ anti  s.t. ps'' ≲ ps' then
13:             add ps'' into working;
14:         end if
15:     end for
16: end while
17: return true;
```

Example 1. We illustrate a simple example in Figure 1 to show how anti-chain works. In the figure, the timed automaton A has a location invariant $x < 5$ on s_2, which means that the system must leave s_2 before the clock is greater than or equal to 5. $\mathrm{ZG_{AL}}$ shows the product of the timed automaton and the LTS. Let $\mathrm{ps}_0 = (s_1, \{s_1'\}, x \geqslant 0)$, $\mathrm{ps}_1 = (s_1, \{s_1', s_2'\}, x > 2)$ and $\mathrm{ps}_2 = (s_1, \{s_1', s_2'\}, x > 3)$. It is not necessary to search from ps_1 and ps_2 because $\mathrm{ps}_1 \lesssim \mathrm{ps}_0$ and $\mathrm{ps}_2 \lesssim \mathrm{ps}_0$.

3.3 Timed trace refinement checking between timed automata

According to our definition of timed traces of timed automata, a timed automaton can be transformed to an equivalent one without location invariants [14], which can simplify the following presentation. The idea is to move the location invariants to the incoming and outgoing transitions for each location. Given a timed automaton $\mathcal{A} = (S, \mathrm{Init}, \Sigma, C, L, T)$ and any $s \in S$ (1) if (s, e, δ, Y, s') is a transition from s, conjunct δ with $L(s)$; (2) if (s', e, δ, Y, s) is a transition to s, conjunct δ with $L(s)[C \backslash Y]$.

The set of timed traces of the resultant timed automaton equals to the set of timed traces of original $\mathcal{A}$. In the following, we fix two timed automata $\mathcal{P} = (S_p, \mathrm{Init}_p, \Sigma_p, C_p, L_p, T_p)$ and $\mathcal{Q} = (S_q, \mathrm{Init}_q, \Sigma_q, C_q, L_q, T_q)$ without location invariants.

3.3.1 *Definitions of timed trace refinement*

Definition 7 (Timed trace refinement)**.** $\mathcal{P}$ refines $\mathcal{Q}$ in timed trace semantics iff $\mathrm{tmtraces}(\mathcal{P}) \subseteq \mathrm{tmtraces}(\mathcal{Q})$.

Given $\mathcal{P}$ and $\mathcal{Q}$, the problem of checking $\mathrm{tmtraces}(\mathcal{P}) \subseteq \mathrm{tmtraces}(\mathcal{Q})$ can be converted to a problem that explores a product of $\mathcal{P}$ and $\mathcal{Q}$, and ensures that any timed trace that $\mathcal{P}$ accepts can be accepted by $\mathcal{Q}$. This procedure requires a synchronous product of $\mathcal{P}$ and a determinization of $\mathcal{Q}$. In the following, we formally define this product in concrete semantics.

Definition 8 (Concrete synchronous product of timed automata)**.** The concrete synchronous product, denoted by $\mathcal{P} \otimes \mathcal{Q}$, is a tuple $(S, \text{Init}, \Sigma, T)$ where

- the set S contains product configurations, each of which is in the form $((s_p, v_p), X)$ where (s_p, v_p) is a concrete configuration of $\mathcal{P}$, and X is a set of concrete configurations of $\mathcal{Q}$;
- Init is a set of initial product configurations, each of which is in the form $((s_p, C_p = 0), X_0)$ where $s_p \in \text{Init}_p$ and $X_0 = \{(s_q, C_q = 0) \mid s_q \in \text{Init}_q\}$;
- the transition relation T is the smallest relation such that for any product configuration $((s_p, v_p), X)$, $(((s_p, v_p), X), (d, e), (s', \text{post}(X, (d, e), \mathcal{Q}))) \in T$ for any $s' \in \text{post}((s_p, v_p), (d, e), \mathcal{P})$. Recall that the function post is defined in Subsection 3.1.

Given a product configuration $((s_p, v_p), X)$, notice that if (d, e) is enabled at (s_p, v_p) but not at any concrete configuration in X, then $\text{post}(X, (d, e), \mathcal{Q})$ is empty. Thus we have the following theorem.

Theorem 3. $\text{tmtraces}(\mathcal{P}) \subseteq \text{tmtraces}(\mathcal{Q})$ iff there is not a reachable state $((s_p, v_p), \emptyset)$ in $\mathcal{P} \otimes \mathcal{Q}$.

According to the above theorem, we can reduce the language inclusion problem to a reachability problem of finding a state $((s_p, v_p), \emptyset)$ in $\mathcal{P} \otimes \mathcal{Q}$. We also remark that given a configuration $((s_p, v_p), X)$ in $\mathcal{P} \otimes \mathcal{Q}$, for all $((s_p, v_p), X')$, if $X' \subseteq X$, then a witness state (in the form of $((s, v), \emptyset)$) is reachable from $((s_p, v_p), X)$ implies that a witness state is also reachable from $((s_p, v_p), X')$.

The remaining problem is that $\mathcal{P} \otimes \mathcal{Q}$ is an infinite-state system which must be reduced before the checking is feasible, which is focused in Subsection 3.3.2.

3.3.2 *Synchronous product with zone abstraction*

The method for generating the synchronous product of timed automata with zone abstraction has been proposed in [14], where the specification $\mathcal{Q}$ is determinized. However this procedure is complicated, i.e., the specification $\mathcal{Q}$ is unfolded to an infinite timed tree. In the following, we represent it in a simplified way which yields an equivalent synchronous product to [14].

Given a clock c, we define $c_+ = \{c_0, c_1, c_2, \ldots\}$ as an infinite set of clocks, which adds subscripts to the original clock c (it will be used for the construction of the synchronous product, where any clock in c_+ can represent the clock c). For any clock $c \in C_q$, define a function $C_{q\oplus}(c)$ that maps the clock c to a unique clock c_x from c_+ (written as $C_{q\oplus}(c) = c_x$). We use $C_{q\oplus}$ to represent that every clock $c \in C_q$ is mapped to a unique clock c_x from c_+. We use $C^0_{q\oplus}$ to denote that every clock $c \in C_q$ is mapped to c_0. For example, given $x, y \in C_q$, $C_{q\oplus}(x) = x_2$ and $C_{q\oplus}(y) = y_3$ are feasible.

The synchronous product is a zone graph $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q}) = (S, \text{Init}, \Sigma, T)$. A state in S is an abstract configuration of the form (s_p, X_q, δ) such that $s_p \in S_p$; X_q is a set of states, each of which is of the form $(s_q, C_{q\oplus})$ where $s_q \in S_q$; and δ is a clock constraint. For any $(s_q, C_{q\oplus}) \in X_q$, $C_{q\oplus}$ represents a set of clocks that are active for the location s_q, which will be further explained together with the construction of T. For the set X_q, we write $\text{Act}(X_q)$ to denote the set of all active clocks, i.e., $\{t \mid \exists(((s_q, C_{q\oplus}) \in X_q) \wedge (c \in C_q)).\ t = C_{q\oplus}(c)\}$. δ constraints all the clocks in $\text{Act}(X_q) \cup C_p$. The Init of the zone graph is defined as $\{(s_p, X_q, ((\text{Act}(X_q) \cup C_p) = 0)^{\uparrow}) \mid s_p \in \text{Init}_p \wedge X_q = \{(s_q, C^0_{q\oplus}) \mid s_q \in \text{Init}_q\}\}$. Σ equals to Σ_p.

Next, we define T by illustrating how to generate successors of a given abstract configuration (s_p, X_q, δ).

Step 1. The set of transitions, denoted by $\text{Tr}(e, X_q)$, is defined as follows. For any state $(s_q, C_{q\oplus}) \in X_q$ and any transition (s_q, e, g_q, Y, s'_q) in T_q which starts with the location s_q and is labeled with event e, a transformed transition $((s_q, C_{q\oplus}), e, g'_q, Y', (s'_q, C'_{q\oplus}))$ is added into $\text{Tr}(e, X_q)$: g_q and g'_q are the same except that for any clock c appearing in g_q, the corresponding clock in g'_q is $C_{q\oplus}(c)$; Y and Y' are the same except that for any clock c appearing in Y, the corresponding clock in Y' is also $C_{q\oplus}(c)$; for any clock $c \in C_q$, if $C_{q\oplus}(c) \notin Y'$, then $C'_{q\oplus}(c) = C_{q\oplus}(c)$, otherwise $C'_{q\oplus}(c) = R$ whose value will be decided later in Step 3. Intuitively, if a clock is not reset on the transition, the clock is still active for the successors.

Step 2. Notice that for the purpose of determinization, the clock guards of transitions in $\text{Tr}(e, X_q)$ must be mutually exclusive. We define a set of clock constraints $\text{Ex}(e, X_q)$ such that each element in $\text{Ex}(e, X_q)$ is a clock constraint, which conjuncts either the transition guard or the negation of the guard for each transition in $\text{Tr}(e, X_q)$ [2)]. Thus, the elements in $\text{Ex}(e, X_q)$ are mutually exclusive by definition.

2) The negation of a clock constraint (a zone) may not be convex, and it can be represented as a set of clock constraints.

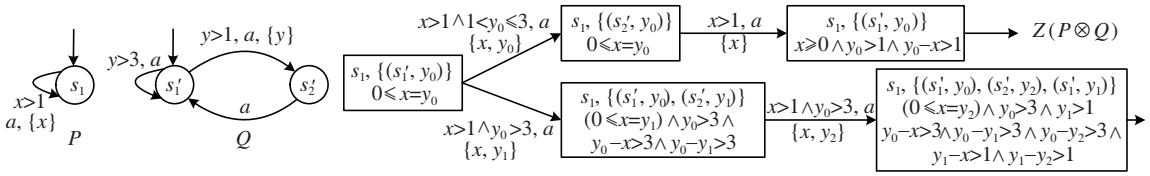

Figure 2 Synchronous product of two timed automata.

For an element in $\mathrm{Ex}(e, X_q)$, if the guard on a transition is not negated, we say that the transition is enabled.

Step 3. Given (s_p, X_q, δ) and an outgoing transition (s_p, e, g_p, Y_p, s'_p) from (s_p, X_q, δ), for each $g \in \mathrm{Ex}(e, X_q)$ we generate a successor (s'_p, X'_q, δ') such that (1) for any state $(s_q, C_{q\oplus}) \in X_q$ and any transition $((s_q, C_{q\oplus}), e, g_q, Y_q, (s'_q, C'_{q\oplus})) \in \mathrm{Tr}(e, X_q)$, if $\delta \wedge g_p \wedge g \wedge g_q$ is not false which means that the transition is enabled, then $(s'_q, C'_{q\oplus}) \in X'_q$; (2) we define two sets of clocks Reset and Active such that for any $(s'_q, C'_{q\oplus}) \in X'_q$ and any clock $c \in C_q$, if $C'_{q\oplus}(c) \neq R$ and $C'_{q\oplus}(c) \notin$ Active, then it is added to Active; if $C'_{q\oplus}(c) = R$ and $c \notin$ Reset, then c is added to Reset; (3) for any $c \in$ Reset, we choose a clock c_x in c_+ such that $c_x \notin$ Active, and for any $(s'_q, C'_{q\oplus}) \in X'_q$, if $C'_{q\oplus}(c) = R$, then $C'_{q\oplus}(c)$ is set to c_x; (4) let $\delta'' = (\delta \wedge g_p \wedge g)[\text{Active}]$, and then $\delta' = \mathcal{N}(([Y_p \mapsto 0](\delta'' \wedge (\text{Reset} = 0))^{\uparrow})$. Notice that with (3), all the active clocks in $\mathrm{Act}(X'_q)$ are decided.

Given a configuration (s_p, X_q, δ) and $v \in \delta$, the concrete configuration can also be written as (s_p, X_q, v). We denote the successors of an abstract configuration ps in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ as $\mathrm{post}(\mathrm{ps}, \mathcal{Z}(\mathcal{P} \otimes \mathcal{Q}))$. Notice that one of the constraints in $\mathrm{Ex}(e, X_q)$ conjuncts the negations of all clock guards of transitions in $\mathrm{Tr}(e, X_q)$, which is denoted by negCons. Assume that the successor generated by negCons is (s'_p, X'_q, δ') and δ' is not false. Obviously X'_q is empty since no transition is enabled. It is easy to see that $\mathcal{P}$ can perform e in some time point whereas $\mathcal{Q}$ cannot. Therefore there exists a timed trace in $\mathcal{P}$ but not in $\mathcal{Q}$.

Theorem 4. $\mathrm{tmtraces}(\mathcal{P}) \subseteq \mathrm{tmtraces}(\mathcal{Q})$ iff there is no reachable state $(s_p, \emptyset, \delta)$ in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ where δ is not false.

The abstract configuration $(s_p, \emptyset, \delta)$ is called a TTR-witness state (TTR: timed trace refinement). Given two abstract configurations (s_p, X_q, δ) and (s'_p, X'_q, δ') of $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$, because of different clock names, we cannot directly check the equivalence of them. For example, the two configurations $(s_1, \{(s'_1, y_0), (s'_2, y_1)\}, x = y_0 > 0 \wedge y_1 > 5)$ and $(s_1, \{(s'_1, y_1), (s'_2, y_0)\}, x = y_1 > 0 \wedge y_0 > 5)$ are actually equivalent, if we exchange y_0 and y_1 in the second configuration. Therefore, we do not need to care about the exact names of the clocks in $\mathrm{Act}(X_q)$ or $\mathrm{Act}(X'_q)$. The equivalence can be checked as follows. If $s_p = s'_p$ and there exists a bijection between X_q and X'_q such that for any $(s_q, C_{q\oplus}) \in X_q$, there is a unique $(s'_q, C'_{q\oplus}) \in X'_q$ with $s_q = s'_q$ (and vice versa). That is, we try to find a mapping from $\mathrm{Act}(X_q)$ to $\mathrm{Act}(X'_q)$ which is a bijective function bi: $\mathrm{Act}(X_q) \to \mathrm{Act}(X'_q)$. Given a constraint δ on the clocks in $\mathrm{Act}(X_q)$, we write $\mathrm{bi}(\delta)$ to denote the constraint obtained by renaming the clocks according to bi. If $\mathrm{bi}(\delta) = \delta'$, the two abstract configurations are equivalent.

Example 2. We illustrate an example in Figure 2 to see how to generate the product with zone abstraction. Let $\mathrm{ps}_0 = (s_1, \{(s'_1, y_0)\}, 0 \leqslant x = y_0)$ which is the initial configuration, ps_1 is one of the successors of ps_0 at the bottom, ps_2 is another successor of ps_0 on the top of ps_1, ps_3 is the successor of ps_1, and ps_4 is the successor of ps_2.

Step 1. For (s'_1, y_0) in ps_0, there are two transitions from s'_1 which are with the guards $y > 3$ and $y > 1$ respectively. Since the initial active clock for y is y_0, then two transitions $((s'_1, y_0), a, y_0 > 3, \emptyset, (s'_1, y_0))$ and $((s'_1, y_0), a, y_0 > 1, \{y_0\}, (s'_2, R))$ are added to $\mathrm{Tr}(a, (s'_1, y_0))$. The variable R in the latter transition will be decided in Step 3.

Step 2. In order to make the guards mutually exclusive, from $\mathrm{Tr}(a, (s'_1, y_0))$ we can get four guards $y_0 > 3 \wedge y_0 > 1$, $y_0 \leqslant 3 \wedge y_0 > 1$, $y_0 > 3 \wedge y_0 \leqslant 1$ and $y_0 \leqslant 3 \wedge y_0 \leqslant 1$, which constitute $\mathrm{Ex}(a, (s'_1, y_0))$. Two of them are feasible, i.e., $y_0 > 3$ and $1 < y_0 \leqslant 3$, which are shown on the two transitions from ps_0 in the figure. For $y_0 > 3$, both of the transitions from s'_1 are enabled. For $1 < y_0 \leqslant 3$, only the transition from s'_1 to s'_2 is enabled.

Step 3. Given the transition from ps_0 with $y_0 > 3$: for the self-transition of s'_1, clock y_0 is not reset, therefore (s'_1, y_0) can transit to (s'_1, y_0) in ps_1 where clock y_0 is still used; for the transition from s'_1 to s'_2, clock y_0 is reset but it is active in (s'_1, y_0), so we initiate a new clock y_1 in (s'_2, y_1) of ps_1 (notice that now $R = y_1$). For the transition from ps_0 with $1 < y_0 \leqslant 3$, only the transition from s'_1 to s'_2 is enabled, and clock y_0 can be reused in ps_2. The zone in ps_1 is calculated from the initial zone in ps_0 and the transition guard. Thus, we can construct the synchronous product step by step.

For the searching of TTR-witness states, it is easy to see that the negCons in $\mathrm{Ex}(a, (s'_1, y_0))$ of ps_0 is $y_0 \leqslant 1$. Then the conjunction of $0 \leqslant x = y_0$ (the zone in ps_0) and $x > 1$ and negCons is not true, thus a TTR-witness state cannot be generated for ps_0 (a TTR-witness state is not reachable and the refinement checking holds in this example).

Notice that the path $\langle \mathrm{ps}_0, \mathrm{ps}_1, \mathrm{ps}_3, \ldots \rangle$ is infinite because the clocks are added infinitely many times, and therefore the state space is infinite. It will be shown that with anti-chain, the algorithm can still terminate for this example in Subsection 3.3.3.

3.3.3 *Algorithm based on anti-chains for timed trace refinement checking*

For the refinement checking between timed automata, we can infer an anti-chain relation by comparing abstract configurations (s_p, X_q, δ) and (s'_p, X'_q, δ') of $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$. A clock mapping is defined for comparing X_q and X'_q (and also δ and δ') which may have different number of locations and different sets of clocks. A mapping from $\mathrm{Act}(X_q)$ to $\mathrm{Act}(X'_q)$ is an injective function $m : \mathrm{Act}(X_q) \to \mathrm{Act}(X'_q)$. We define $X_q \subseteq_m X'_q$ if there exists a mapping m such that for all $(s_q, C_{q\oplus}) \in X_q$, there exists $(s'_q, C'_{q\oplus}) \in X'_q$ such that $s_q = s'_q$ and for all $c \in C_q$, $m(C_{q\oplus}(c)) = C'_{q\oplus}(c)$. Note that there may be some clocks in $\mathrm{Act}(X'_q)$ that are not mapped to, and X'_q may contain more elements than X_q. We use maps_m to denote the set of clocks which are mapped to in $\mathrm{Act}(X'_q)$. Similarly, given a constraint δ on the clocks in $\mathrm{Act}(X_q)$, we write δ_m to denote the constraint obtained by renaming the clocks according to m. $\delta'[\mathrm{maps}_m]$ denotes the clock constraint obtained by projecting δ' onto clocks in maps_m. We write $\delta' \subseteq_m \delta$ if $\delta'[\mathrm{maps}_m] \subseteq \delta_m$, which means that the clock valuations which satisfy the constraint $\delta'[\mathrm{maps}_m]$ also satisfy δ after the clock renaming. We define $(s_p, X'_q, \delta') \preccurlyeq (s_p, X_q, \delta)$ iff there exists a mapping m such that $X_q \subseteq_m X'_q$ and $\delta' \subseteq_m \delta$. Then we get the following lemma.

Lemma 3. Let (s_p, X_q, δ) and (s_p, X'_q, δ') be two abstract configurations in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$. If $(s_p, X'_q, \delta') \preccurlyeq (s_p, X_q, \delta)$, then a TTR-witness state is reachable from (s_p, X'_q, δ') implies a TTR-witness state is reachable from (s_p, X_q, δ).

The anti-chain based timed trace refinement checking algorithm, denoted by Algorithm 2, constructs $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ on-the-fly with reachability analysis using anti-chain. Lines 5 and 11 ensure that anti is an anti-chain. The following theorem states that the algorithm always produces correct results.

Theorem 5. Algorithm 2 returns true iff $\mathrm{tmtraces}(\mathcal{P}) \subseteq \mathrm{tmtraces}(\mathcal{Q})$.

Algorithm 2 Anti-chain based timed trace refinement checking between timed automata

```
1:  Let working := Init;
2:  Let anti := ∅;
3:  while working ≠ ∅ do
4:      remove ps := (s_p, X_q, δ) from working;
5:      remove all ps' ∈ anti  s.t. ps' ≼ ps;
6:      add ps into anti;
7:      if X_q = ∅ then
8:          return false;
9:      end if
10:     for all (s'_p, X'_q, δ') ∈ post(ps, Z(P ⊗ Q))
        do
11:         if ∄ ps' ∈ anti  s.t. (s'_p, X'_q, δ') ≼ ps'
            then
12:             add (s'_p, X'_q, δ') into working;
13:         end if
14:     end for
15: end while
16: return true;
```

If $\mathcal{Q}$ is deterministic or can be determinized, Algorithm 2 terminates. However, Algorithm 2 does not always terminate since there exist some cases where $\mathcal{Q}$ is not determinizable and the clocks are added

infinitely, as shown in Figure 2. We show that the anti-chain can reduce the state space, as well as contributing to the termination of the algorithm with the following example.

Example 3. In Figure 2, we show that $\mathrm{ps}_1 \preccurlyeq \mathrm{ps}_2$ (ps_1 is denoted by (s_p, X'_q, δ') and ps_2 is denoted by (s_p, X_q, δ)), and as a result, we do not need to search from ps_1. We can find a clock mapping $m(y_0) = y_1$ for ps_2 which replaces y_0 with y_1 in ps_2. Then we have $\mathrm{ps}'_2 = (s_1, \{(s'_2, y_1)\}, 0 \leqslant x = y_1)$ which equals to ps_2 (δ is $0 \leqslant x = y_0$ and δ_m is $0 \leqslant x = y_1$). Since $\mathrm{maps}_m = \{y_1\}$, $\delta'[\mathrm{maps}_m]$ is $0 \leqslant x = y_1$. Compared with ps_1, it is easy to see that $\mathrm{ps}_1 \preccurlyeq \mathrm{ps}'_2$ and thus $\mathrm{ps}_1 \preccurlyeq \mathrm{ps}_2$. Although the path $\langle \mathrm{ps}_0, \mathrm{ps}_1, \mathrm{ps}_3, \ldots \rangle$ is infinite, it is pruned by the anti-chain. The infinite path $\langle \mathrm{ps}_0, \mathrm{ps}_2, \mathrm{ps}_4, \ldots \rangle$ can also be pruned similarly.

4 Probabilistic refinement checking

4.1 Probabilistic model checking

A Markov Decision Process (MDP) can capture system behaviours with probabilistic choices and non-determinism. The definition of MDP [22] is introduced as follows. For a set W, the map $\mu : W \to [0, 1]$ where $\Sigma_{w \in W} \mu(w) = 1$ represents a distribution. We use $\mathrm{Distr}(W)$ to denote the set of all distributions over W.

An MDP is denoted by a tuple $\mathcal{M} = (S, \mathrm{Ini}, A, P)$ such that S is a set of locations, Ini: $S \to [0, 1]$ is the initial distribution such that $\Sigma_{s \in S} \mu(s) = 1$, A is a set of visible events, and $P \subseteq S \times A \times \mathrm{Distr}(S)$ is the transition function.

There could be multiple events and distributions from a location in MDP. A transition in $\mathcal{M}$ is represented as (s, e, μ) such that μ is a distribution. $\pi = \langle s_0, e_0, \mu_0, s_1, e_1, \mu_1, s_2, e_2, \mu_2, \ldots \rangle$ represents an infinite path in $\mathcal{M}$ where $\mathrm{Ini}(s_0) > 0$, $(s_i, e_i, \mu_i) \in P$ and $\mu_i(s_{i+1}) > 0$ for any $i \geqslant 0$.

We use a scheduler to decide how to choose an event and a distribution in the MDP. A scheduler chooses in any state s one of the enabled events e and a distribution μ satisfying $(s, e, \mu) \in P$. We write $\beta : S \to A \times \mathrm{Distr}(S)$ to denote the scheduler. Given an MDP and a scheduler β, a Markov Chain can be obtained [2], written as $\mathcal{M}_\beta$. A Markov Chain can be seen as an MDP such that just one event (and one distribution) is enabled at each location. Notice that paths in Markov Chains are "maximal" (i.e., infinite) paths in the underlying digraph [2]. For a path $\pi_\beta = \langle s_0, e_0, \mu_0, s_1, e_1, \mu_1, s_2, e_2, \mu_2, \ldots \rangle$ in $\mathcal{M}$ scheduled by β, which is also in $\mathcal{M}_\beta$, $\mathrm{trace}(\pi)$ is defined as a sequence of events $\langle e_0, e_1, e_2, \ldots \rangle$. For the reachability analysis involved later, it is important to calculate the probability that a particular set of locations is reached (the article [2] has described and proved the method of computing reachability probabilities by infinite paths). Given a Markov Chain $\mathcal{M}_\beta$ and a set of target locations G_t, we use $\Pr(\mathcal{M}_\beta, s, G_t)$ to represent the accumulated probability of all paths from s to any location in G_t. Given a Markov Chain $\mathcal{M}_\beta$ and a trace tr, we use $\Pr(\mathcal{M}_\beta, s, \mathrm{tr})$ to denote the probability of running a trace tr starting from s: from all the paths π in $\mathcal{M}_\beta$ starting from s where $\mathrm{trace}(\pi) = \mathrm{tr}$, the accumulated probability can be obtained. For a set of traces Tr, we use $\Sigma_{\mathrm{tr} \in \mathrm{Tr}} \Pr(\mathcal{M}_\beta, s, \mathrm{tr})$ to denote the probability that $\mathcal{M}_\beta$ runs any trace in Tr starting from s.

Notice that the probability for reaching G_t or running a trace can be different using various schedulers. Thus maximal and minimal probabilities are measured. Derived from above, given an MDP $\mathcal{M}$ and a target location set G_t, the maximal probability of reaching G_t starting from s is $\Pr_{\max}(\mathcal{M}, s, G_t) = \sup_\beta \Pr(\mathcal{M}_\beta, s, G_t)$; the minimal is $\Pr_{\min}(\mathcal{M}, s, G_t) = \inf_\beta \Pr(\mathcal{M}_\beta, s, G_t)$. The maximal probability of running a trace set Tr by $\mathcal{M}$ from s is $\Pr_{\max}(\mathcal{M}, s, \mathrm{Tr}) = \sup_\beta(\Sigma_{\mathrm{tr} \in \mathrm{Tr}} \Pr(\mathcal{M}_\beta, s, \mathrm{tr}))$; the minimal probability is $\Pr_{\min}(\mathcal{M}_\beta, s, \mathrm{Tr}) = \inf_\beta(\Sigma_{\mathrm{tr} \in \mathrm{Tr}} \Pr(\mathcal{M}_\beta, s, \mathrm{tr}))$.

Definition 9 (Probabilistic refinement)**.** Given an MDP $\mathcal{M} = (S, \mathrm{Ini}, A, P)$ and an LTS $\mathcal{L}$. Let $S_I = \{s' \mid \exists s' \in S.\ \mathrm{Ini}(s') > 0\}$ be the set of initial locations distributed by Ini. The maximal probability that $\mathcal{M}$ refines $\mathcal{L}$ in trace semantics is $\mathrm{Prob}_{\max}(\mathcal{M}, \mathrm{Ini}, \mathrm{traces}(\mathcal{L})) = \Sigma_{s \in S_I} \mathrm{Ini}(s) \times \Pr_{\max}(\mathcal{M}, s, \mathrm{traces}(\mathcal{L}))$; the minimal probability is $\mathrm{Prob}_{\min}(\mathcal{M}, \mathrm{Ini}, \mathrm{traces}(\mathcal{L})) = \Sigma_{s \in S_I} \mathrm{Ini}(s) \times \Pr_{\min}(\mathcal{M}, s, \mathrm{traces}(\mathcal{L}))$.

However, we cannot directly calculate the maximal/minimal probability of $\mathcal{M}$ trace-refines $\mathcal{L}$ with above definition. From [8], one way is to (1) determine $\mathcal{L}$ using the standard powerset construction; (2)

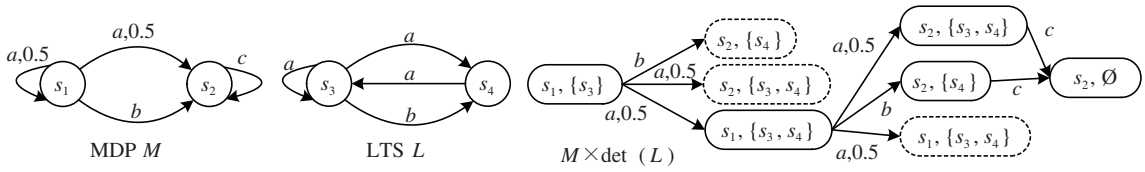

Figure 3 Synchronous product of an MDP and an LTS.

compute the synchronous product of $\mathcal{M}$ and determined $\mathcal{L}$; (3) reduce the problem to a probabilistic reachability one and calculate the probability of paths of the product.

Definition 10 (Synchronous product of MDP and LTS)**.** Given an MDP $\mathcal{M} = (S_m, \mathrm{Ini}_m, A_m, P_m)$, an LTS $\mathcal{L} = (S_l, \mathrm{Init}_l, \mathrm{Act}_l, T_l)$, and the determinized LTS $\det(\mathcal{L}) = (S'_l, \mathrm{Init}'_l, \mathrm{Act}'_l, T'_l)$. The synchronous product $\mathcal{M} \times \det(\mathcal{L})$, is denoted by an MDP which is a tuple (S, Ini, A, P) where $S = S_m \times S'_l$; $\mathrm{Ini} : S_m \times \{\mathrm{Init}'_l\} \to [0,1]$ is an initial distribution such that for any $s_m \in S_m$, $\mathrm{Ini}((s_m, \mathrm{Init}'_l)) = \mathrm{Ini}_m(s_m)$; $A = A_m$; and P is a transition function which satisfies that for all (s, N) with $s \in S_m$ and $N \in S'_l$,

- if $(s, \tau, \mu) \in P_m$, then $((s, N), \tau, \mu') \in P$ where for any $s' \in S_m$, $\mu'((s', N)) = \mu(s')$;
- if $(s, e, \mu) \in P_m$, $(N, e, N') \in T'_l$, then $((s, N), e, \mu') \in P$: for any $s' \in S_m$, $\mu'((s', N')) = \mu(s')$.

A state (s, N) in $\mathcal{M} \times \det(\mathcal{L})$ is called a PR-witness state (PR: Probabilistic Refinement) iff $N = \emptyset$. It means that if $(s, \emptyset)$ is reachable, then there exists a trace that $\mathcal{M}$ exhibits but $\mathcal{L}$ does not, which decreases the probability that $\mathcal{M}$ refines $\mathcal{L}$. Let G_t be the target set of all the PR-witness states in $\mathcal{M} \times \det(\mathcal{L})$, the problem can be reduced to a probabilistic reachability one by the following theorem [8].

Theorem 6. Given an MDP $\mathcal{M} = (S_m, \mathrm{Ini}_m, A_m, P_m)$, an LTS $\mathcal{L} = (S_l, \mathrm{Init}_l, \mathrm{Act}_l, T_l)$, and $\mathcal{M} \times \det(\mathcal{L}) = (S, \mathrm{Ini}, A, P)$. Let $S_I = \{s' \mid \exists s' \in S.\ \mathrm{Ini}(s') > 0\}$ be the set of initial locations distributed by Ini, and G_t be the set of all the PR-witness states in $\mathcal{M} \times \det(\mathcal{L})$:

- $\mathrm{Prob}_{\max}(\mathcal{M}, \mathrm{Ini}_m, \mathrm{traces}(\mathcal{L})) = 1 - \Sigma_{s \in S_I} \mathrm{Ini}(s) \times \mathrm{Pr}_{\min}(\mathcal{M} \times \det(\mathcal{L}), s, G_t)$;
- $\mathrm{Prob}_{\min}(\mathcal{M}, \mathrm{Ini}_m, \mathrm{traces}(\mathcal{L})) = 1 - \Sigma_{s \in S_I} \mathrm{Ini}(s) \times \mathrm{Pr}_{\max}(\mathcal{M} \times \det(\mathcal{L}), s, G_t)$.

We use value iterative approximation technique [2] which is more often applied for calculating the probability in the probabilistic verification.

Example 4. Figure 3 illustrates how the iterative calculation works. In the MDP, s_1 starts with two distributions, followed by two events a and b. The synchronous product $\mathcal{M} \times \det(\mathcal{L})$ is shown in the figure by following Definition 10. Let $\mathrm{ps}_0 = (s_1, \{s_3\})$, $\mathrm{ps}_1 = (s_1, \{s_3, s_4\})$, $\mathrm{ps}_2 = (s_2, \{s_4\})$, $\mathrm{ps}_3 = (s_2, \{s_3, s_4\})$ and $G_t = \{(s_2, \emptyset)\}$ which is the set of PR-witness states. We calculate the maximal probability of reaching G_t step by step. After k iterations, we denote ps_i^k as the maximal probability of ps_i. All the probabilities of the states in G_t equal to 1 in any iteration since they are always reachable from themselves. Initially, the probabilities of all the other states are set to 0. In each iteration we update the probabilities on the states according to the values of last iteration.

- Iteration 0. $\mathrm{ps}_0^0 = \mathrm{ps}_1^0 = \mathrm{ps}_2^0 = \mathrm{ps}_3^0 = 0$; $G_t = 1$.
- Iteration 1. $\mathrm{ps}_3^1 = 1 \times G_t = 1$; $\mathrm{ps}_2^1 = 1 \times G_t = 1$; $\mathrm{ps}_1^1 = \max\{0.5 \times \mathrm{ps}_1^0 + 0.5 \times \mathrm{ps}_3^0, \mathrm{ps}_2^0\} = 0$; $\mathrm{ps}_0^1 = \max\{0.5 \times \mathrm{ps}_1^0 + 0.5 \times \mathrm{ps}_3^0, \mathrm{ps}_2^0\} = 0$.
- Iteration 2. $\mathrm{ps}_1^2 = \max\{0.5 \times \mathrm{ps}_1^1 + 0.5 \times \mathrm{ps}_3^1, \mathrm{ps}_2^1\} = 1$; $\mathrm{ps}_0^2 = \max\{0.5 \times \mathrm{ps}_1^1 + 0.5 \times \mathrm{ps}_3^1, \mathrm{ps}_2^1\} = 1$.

Iteratively, the probability of the initial state ps_0 reaching G_t can be calculated, which is 100%. According to Theorem 6, the probability that M refines L is 0. Notice that the calculation of this example stops quickly. However, in many situations the iterations are infinite. Therefore, a user-defined threshold is often used to stop the calculation when the result is accurate enough.

4.2 Algorithm based on anti-chains for probabilistic refinement checking

Lemma 4. Given an MDP $\mathcal{M}$, an LTS $\mathcal{L}$. Let $\mathcal{D} = \mathcal{M} \times \det(\mathcal{L})$, and the set G_t containing all the PR-witness states of $\mathcal{D}$. For any product states (s_1, N_1) and (s_2, N_2) of $\mathcal{D}$ s.t. $(s_2, N_2) \preceq (s_1, N_1)$, $\mathrm{Pr}_{\max}(\mathcal{D}, (s_1, N_1), G_t) \geqslant \mathrm{Pr}_{\max}(\mathcal{D}, (s_2, N_2), G_t)$ and $\mathrm{Pr}_{\min}(\mathcal{D}, (s_1, N_1), G_t) \geqslant \mathrm{Pr}_{\min}(\mathcal{D}, (s_2, N_2), G_t)$.

Table 1 Tests on refinement checking of TA and LTS

System	Time			Visited states		
	Without AC (s)	AC (s)	Ratio	Without AC	AC	Ratio
FIS×6(2)	12.2	8.9	1.37	260.7 K	172.7 K	1.51
FIS×6(6)	22.9	6.3	3.63	497.0 K	100.2 K	4.96
FIS×8(1)	4.2	4.9	0.86	88.4 K	88.4 K	1.00
RW×6(3)	1.53	0.68	2.25	68.7 K	23.0 K	2.99
RW×8(3)	124.1	42.3	2.93	4.3 M	1.1 M	3.91
LYN×7(2)	4.5	3.5	1.29	165.0 K	130.0 K	1.27
LYN×8(2)	26.0	19.3	1.35	659.1 K	519.0 K	1.27
CSMA×7(1)	15.6	16.9	0.92	146.2 K	146.2 K	1.00

Recall that $(s_2, N_2) \preceq (s_1, N_1)$ has been defined in Section 2. For the probability calculation, the state space cannot be reduced directly. However, we can use the above lemma to speed up the iterative calculation. We define a set ac for each product state (s, N) such that $(s, N).\text{ac} = \{(s', N') \mid (s', N') \in S \wedge (s, N) \preceq (s', N')\}$ where S is the set of all the product states in $\mathcal{D}$. Then we apply the iterative calculation in $\mathcal{D}$. During the calculation, if the probability of state (s, N) is updated to p, for any product state in $(s, N).\text{ac}$, if the probability is less than p, then it can be set to p directly, which reduces the calculation.

Example 5. Given the same example in Figure 3. We have $\text{ps}_3.\text{ac} = \{\text{ps}_2\}$ and $\text{ps}_1.\text{ac} = \{\text{ps}_0\}$. As a result, in iteration 1, if ps_3^1 has been updated to 1, then ps_2^1 can be set to 1 directly without calculation (although the original calculation which is $1 \times G_t$ looks trivial); similarly, in iteration 2, if ps_1^2 has been calculated and updated to 1, then ps_0^2 can also be set to 1 directly (the original calculation is not trivial). In this example, there are only a few states, so the speedup is not obvious. For a system with a large size, this method can speedup the calculations potentially.

5 Evaluation

In this section, the performance of the algorithms is evaluated using some real-life systems, which have been embedded in PAT tool [15]. All the experiments are conducted using a PC (Intel (R) Core (TM) 3.40 GHz i7-2600 CPU with 8.0 GB of RAM) [3)].

5.1 Evaluation for timed systems

In this part, we model and verify timed benchmark systems with our algorithms, and evaluate the performance. The benchmark systems include Fischer's mutual exclusion protocol (FIS) [23], Lynch-Shavit's mutual exclusion protocol (LYN) [24], railway control system (RW) [25], fiber distributed data interface (FDDI) [26], and CSMA/CD protocol (CSMA) [27].

Table 1 shows the tests on refinement checking of timed automata and LTS with anti-chain (AC) and without anti-chain (Without AC) with several cases. All the systems and the properties to be verified are a set of processes, e.g., FIS × 6(2) indicates that there are 6 processes in the system and 2 processes in the property. Different properties are built for the benchmarks using LTSs, e.g., the refinement checking between the system and the property for FIS can be expressed as that whether a process or several processes can enter the critical section after they initiate the request. From the ratios of the verification time and visited states, we can see that the anti-chain based algorithm improves the performance in most of the cases. For some cases (e.g., FIS × 8(1)), the ratio of visited states is 1 which means that the state space cannot be reduced by anti-chain at all, because (1) the properties are deterministic; (2) the comparisons between zones (see the ani-chain relation in Lemma 2) and the Lw/Up functions cannot contribute to reduce the state space. From these tests, we can infer that the anti-chain can often provide better performance in the cases with non-deterministic property models. However, the effect of

3) All the models can be found in: http://pat.scse.ntu.edu.sg/antichain/index.htm.

Table 2 Tests on refinement checking between TA

System	Time			Visited states		
	Without AC (s)	AC (s)	Ratio	Without AC	AC	Ratio
FIS×6(1)	1.4	0.7	2.00	7.1 K	4.4 K	1.61
FIS×7(1)	8.9	4.6	1.93	31.3 K	20.0 K	1.57
FIS×8(1)	57.9	29.7	1.95	138.7 K	91.6 K	1.51
RW×6(6)	8.9	7.0	1.27	27.9 K	23.3 K	1.20
RW×7(1)	18.5	12.9	1.43	126.9 K	99.5 K	1.28
LYN×5(2)	48.8	2.3	21.2	63.6 K	8.1 K	7.85
LYN×6(1)	69.5	4.0	17.4	120.9 K	16.8 K	7.20
FDDI×7(7)	36.1	8.0	4.51	8.1 K	1.2 K	6.75
CSMA×5(1)	3.4	0.2	17.0	2.5 K	0.9 K	2.78

Table 3 Tests on refinement checking of MDP and LTS

System	States	Time			States during iterations		
		Without AC (s)	AC (s)	Ratio	Without AC	AC	Ratio
CSI(3)	46 K	2.7	2.2	1.23	4.2 M	3.0 M	1.40
CSI(4)	87 K	16.0	12.0	1.33	18.6 M	11.7 M	1.59
CSI(5)	117 K	123.9	80.8	1.53	130.7 M	76.3 M	1.71
CSI(6)	231 K	271.2	182.6	1.49	272.1 M	160.7 M	1.69
CSI(7)	343 K	511.1	340.3	1.50	515.2 M	298.8 M	1.72
MRS(3)	8 K	2.5	2.8	0.89	4.9 M	4.8 M	1.02
MRS(4)	59 K	47.1	49.3	0.96	43.4 M	43.3 M	1.00

the comparisons between zones and the Lw/Up functions depend on the system models since they can be built arbitrarily.

Table 2 shows the tests on refinement checking between timed automata, which enables the verification of timed properties expressed by timed automata. The systems and timed properties are also represented by a set of processes. The timed properties contain different number of clocks. For example, FIS $\times$ 6(1) indicates that there are 6 processes in the system and one clock (also one process) in the timed property. The refinement checking between the system and the timed property for FIS can be expressed as that whether a process or several processes can enter the critical section after they initiate the request within the required time. The timed properties are modeled using a set of commonly used patterns [28], and all the patterns are deterministic or can be determinized. Therefore, the algorithm can terminate for all these test cases. From the ratios of the verification time and visited states, it is obvious that the anti-chain based algorithm can often reduce the state space and improve the performance.

5.2 Evaluation for probabilistic systems

Now we evaluate the anti-chain based algorithm for refinement checking between MDP and LTS, which is sometimes appropriate for verifying complex properties. The experiments are performed on two probabilistic systems, i.e., a concurrent stack implementation (CSI) [29] and a multi-valued register simulation system (MRS) [30]. For example, the MRS system is from distributed computing modeling that how multiple readers and writers probabilistically initiate the registration of the resources. The property is described by a composition of LTSs which specifies correct reading and writing behaviours, and the refinement checking algorithm between the system and the property calculates the probability that the MRS system can register the resources correctly. In Table 3, the number in the column "System" shows the CSI and MRS and their sizes; the column "States" shows the total number of states in the product of MDP and LTS. The column "States during iterations" shows that during the iterative calculation process, how many states are involved in (the probability of a state may be calculated multiple times during the iterations). In the case of CSI, the total number of states during iterations can be reduced by the anti-chain based algorithm, and the verification time is shortened. We also show that in the case of

MRS, the states involved in the iterations can only be reduced slightly, thus the overhead of anti-chain operations decreases the benefit contributed by anti-chain. For this algorithm, if the property model is deterministic and without any simulation relation (see the relations $\prec$, $\preceq$ in Section 2 and Lemma 4), the anti-chain algorithm cannot work at all; if the degree of non-determinization is "low" and the simulation relations are rare, the anti-chain algorithm cannot improve the performance a lot, e.g., the case MRS; otherwise, the anti-chain algorithm can contribute to performance improvement.

6 Related work

In this part, we investigate the work related to anti-chain. We will give a brief description for them, as well as the relevance to our work. Wulf et al. [12] first suggested the anti-chain algorithm for the universality and language inclusion of non-deterministic finite automata (NFA) on finite words, and also showed how anti-chain could be used to solve the emptiness problem for alternating finite automata. Their results show that the anti-chain based approach largely increases the performance compared to the standard methods. In their following work [31, 32], it also shows that the anti-chain based method can play an important role for non-deterministic Büchi automata on infinite words in the universality and language inclusion problems, as well as the emptiness problem for alternating Büchi automata. Later Abdulla et al. [16] enhanced the anti-chain based method by using simulation relations on NFA with finite words. It proves that the simulation relation can help to reduce a larger portion of the state space. Motivated by the previous work, we incorporate the idea of anti-chain in the problems of refinement checking of timed systems and probabilistic systems in this paper. We also remark the work based on anti-chain [16,33,34] for solving other problems which are remotely related to our work. Anti-chain has played an important role in raising the efficiency, according to their experimental results.

There are some important articles for the language inclusion problem of timed automata where both the implementation and specification are timed automata, and some of them are related to anti-chain. If we use timed automata as a specification language, the language inclusion checking may not terminate, since the timed automata are not decidable in general [35]. Naturally, in order to avoid this problem, some articles identify determinizable subclasses of timed automata with reduced expressiveness. For example, strongly non-Zeno timed automata [10], timed automata with integer resets [36], timed automata with one clock [11] and event-clock timed automata [37] have been proved to be determinizable. In particular, for the language inclusion problem of one-clock timed automata which are decidable [11], the anti-chain is used to ensure the termination of the algorithm. Although our method is not decidable, it can be applied to arbitrary timed automata, and for many commonly used timed patterns, the algorithm can terminate. Meanwhile, in our method, the anti-chain can also contribute to the termination of the algorithm.

Most of the previous work is based on region abstraction which is inefficient in practice. Compared to region graph, zone graph is a more succinct representation for the verification and it is often used in existing tools like Uppaal [38] and Kronos [39]. The zone based approach has been applied to the universality problem of one-clock timed automata [9], where anti-chain is used for proving the decidability. Our previous work [14] proposed a zone based construction for the language inclusion checking. In this work, we simplify the construction of the state space by getting rid of infinite timed trees in [14], and further show that anti-chain contributes to the termination of the algorithm, as well as the state space reduction.

7 Conclusion and future work

In summary, we study two kinds of refinement checking in this work, that is, timed refinement checking and probabilistic refinement checking. For the timed refinement checking with an untimed specification, two kinds of anti-chains are utilized to reduce the state space. For the timed refinement checking with a timed specification, we show that anti-chain can terminate the algorithm in certain situations although the problem is undecidable. More importantly, in the verification of many real-life systems as shown in

the evaluation, anti-chain improves the performance significantly. For probabilistic refinement checking with an untimed specification, although the state space cannot be reduced, anti-chain can speed up the iterative probability calculation.

As for the future work, we would like to explore the refinement relation between probabilistic systems, which may also be motivated by the anti-chain approach. In addition, we will investigate the timed trace refinement checking problem with the assumption of non-Zenoness to enhance the refinement checking.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61602412, 61103044, U1509214, 61402406) and Natural Science Foundation of Zhejiang Province of China (Grant No. LY16F020035).

References

1 Roscoe A W. Model-Checking CSP. Upper Saddle River: Prentice-Hall, 1994
2 Baier C, Katoen J P. Principles of Model Checking. Cambridge: The MIT Press, 2008
3 Li W, Li N. A formal semantics for program debugging. Sci China Inf Sci, 2012, 55: 133–148
4 Li H, Luo J, Li W. A formal semantics for debugging synchronous message passing-based concurrent programs. Sci China Inf Sci, 2014, 57: 128101
5 Che X P, Maag S. Testing protocols in internet of things by a formal passive technique. Sci China Inf Sci, 2014, 57: 032101
6 Hoare C A R. Communicating sequential processes. In: The Origin of Concurrent Programming. Berlin: Springer, 1985. 413–443
7 Roscoe A W. On the expressive power of CSP refinement. Form Asp Comput, 2005, 17: 93–112
8 Sun J, Song S, Liu Y. Model checking hierarchical probabilistic systems. In: Proceedings of the 12th International Conference on Formal Engineering Methods (ICFEM), Shanghai, 2010. 388–403
9 Abdulla P A, Ouaknine J, Quaas K, et al. Zone-based universality analysis for single-clock timed automata. In: Proceedings of International Conference on Fundamentals of Software Engineering (FSE), Luxembourg, 2007. 98–112
10 Baier C, Bertrand N, Bouyer P, et al. When are timed automata determinizable? In: Proceedings of International Colloquium on Automata, Languages, and Programming (ICALP), Rhodes, 2009. 43–54
11 Ouaknine J, Worrell J. On the language inclusion problem for timed automata: closing a decidability gap. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS), Turku, 2004. 54–63
12 Wulf M D, Doyen L, Henzinger T A, et al. Antichains: a new algorithm for checking universality of finite automata. In: Proceedings of the 18th International Conference on Computer Aided Verification (CAV), Seattle, 2006. 17–30
13 Bengtsson J, Yi W. Timed automata: semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. Berlin: Springer, 2004. 87–124
14 Wang T, Sun J, Liu Y, et al. Are timed automata bad for a specification language? Language inclusion checking for timed automata. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Grenoble, 2014. 310–325
15 Liu Y, Sun J, Dong J S. Developing model checkers using PAT. In: Proceedings of the 8th International Symposium on Automated Technology for Verification and Analysis (ATVA), Singapore, 2010. 371–377
16 Abdulla P A, Chen Y F, Holk L, et al. When simulation meets antichains. In: Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Paphos, 2010. 158–174
17 Henzinger T A, Nicollin X, Sifakis J, et al. Symbolic model checking for real-time systems. J Inform Comput, 1994, 111: 193–244
18 Bouyer P. Forward analysis of updatable timed automata. Form Meth Syst Des, 2004, 24: 281–320
19 Rokicki T G. Representing and modeling digital circuits. Dissertation for Ph.D. Degree. San Francisco: Stanford University, 1993
20 Tripakis S. Checking timed buchi automata emptiness on simulation graphs. ACM Trans Comput Logic, 2009, 10: 1–19
21 Behrmann G, Bouyer P, Larsen K G, et al. Lower and upper bounds in zonebased abstractions of timed automata. Int J Softw Tools Technol Trans, 2004, 8: 204–215
22 Puterman M L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Hoboken: John Wiley and Sons, 1994
23 Vereijken J J. Fischer's Protocol in Timed Process Algebra. Technical Report. 1994
24 Lynch N, Shavit N. Timing-based mutual exclusion. In: Proceedings of Real-Time Systems Symposium (RTSS), Phoenix, 1992. 2–11
25 Behrmann G, David R, Larsen K G. A tutorial on uppaal. In: Formal Methods for the Design of Real-Time Systems. Berlin: Springer, 2004. 200–236
26 Daws C, Tripakis S. Model checking of real-time reachability properties using abstractions. In: Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lisbon, 1998. 313–329

27 Duflot M, Fribourg L, Herault T, et al. Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC. Electron Notes Theor Comput Sci, 2005, 128: 195–214
28 Gruhn V, Laue R. Patterns for timed property specifications. Electron Notes Theor Comput Sci, 2006, 153: 117–133
29 Treiber R K. Systems Programming: Coping with Parallelism. Technical Report, IBM Almaden Research Center. 1986
30 Attiya H, Welch J. Distributed Computing: Fundamentals, Simulations, and Advanced Topics. 2nd ed. Oxford: The Oxford University Press, 2004
31 Doyen L, Raskin J F. Antichains for the automata-based approach to model checking. Logical Meth Comput Sci, 2009, 5: 1–20
32 Wulf M D, Doyen L, Maquet N, et al. Antichains: alternative algorithms for LTL satisfiability and model-checking. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Budapest, 2008. 63–77
33 Bouajjani A, Habermehl P, Holk L, et al. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In: Proceedings of the 13th International Conference on Implementation and Application of Automata (CIAA), San Francisco, 2008. 57–67
34 Filiot E, Jin N, Raskin J F. An antichain algorithm for LTL realizability. In: Proceedings of the 21st International Conference on Computer Aided Verification (CAV), Grenoble, 2009. 263–277
35 Alur R, Dill D L. A theory of timed automata. Theory Comput Sci, 1994, 126: 183–235
36 Suman P V, Pandya P K, Krishna S N, et al. Timed automata with integer resets: language inclusion and expressiveness. In: Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), Saint-Malo, 2008. 78–92
37 Alur R, Fix L, Henzinger T A. Event-clock automata: a determinizable class of timed automata. Theor Comput Sci, 1999, 211: 253–273
38 Larsen K G, Petterson P, Wang Y. UPPAAL in a nutshell. J Softw Tools Technol Trans, 1997, 1: 134–152
39 Yovine S. Kronos: a verification tool for real-time systems. J Softw Tools Technol Trans, 1997, 1: 123–133

Appendix A Proofs for all the sections

Theorem 1. Given a timed automaton $\mathcal{A}$ and an LTS $\mathcal{L}$, $\text{traces}_{\text{ZG}}(\mathcal{A}) \subseteq \text{traces}(\mathcal{L})$ iff there is no reachable state $(s, \delta, \emptyset)$ in ZG_{AL}.

Proof. (**if**) Let $|\pi|$ be the length of a trace π. If $\text{traces}_{\text{ZG}}(\mathcal{A}) \subsetneq \text{traces}(\mathcal{L})$, there is a trace π_A in $\text{traces}_{\text{ZG}}(\mathcal{A})$ and a trace π_L in $\text{traces}(\mathcal{L})$, such that π_L is a maximal prefix of π_A satisfying (1) $|\pi_L| = |\pi_A| - 1$; (2) π_A is not in $\text{traces}(\mathcal{L})$. Let (s, δ, W) be the abstract configuration, where (s, δ) is a state in $\text{ZG}(\mathcal{A})$ with which a run of $\text{ZG}(\mathcal{A})$ exhibiting π_L ends, and W is a state in $\det(\mathcal{L})$ with which a run of $\det(\mathcal{L})$ exhibiting π_L ends. Notice that W is not empty. Next, (s, δ) can perform an event and transit to (s', δ') (exhibiting π_A), but none of y in W can perform the same event because π_A is not in $\text{traces}(\mathcal{L})$. Thus, an abstract configuration $(s', \delta', \emptyset)$ is reachable. As a result, if there is no reachable state $(s, \delta, \emptyset)$ in ZG_{AL}, $\text{traces}_{\text{ZG}}(\mathcal{A}) \subseteq \text{traces}(\mathcal{L})$ holds.

(**only-if**) It is proved straightforwardly. If $\text{traces}_{\text{ZG}}(\mathcal{A}) \subseteq \text{traces}(\mathcal{L})$, then any trace π performed by $\text{ZG}(\mathcal{A})$ can also be performed by $\mathcal{L}$. Thus $(s, \delta, \emptyset)$ is not reachable.

Lemma 2. Let (s, δ, W) and (s, δ', W') be two abstract configurations in ZG_{AL}. If $(s, \delta, W) \lesssim (s, \delta', W')$, then a TATR-witness state is reachable from (s, δ, W) implies a TATR-witness state is reachable from (s, δ', W').

Proof. By induction. (1) First, the base case is that (s, δ, W) is a TATR-witness state (that is, $W = \emptyset$), because $W' \subseteq W$, $W = \emptyset$ and therefore (s, δ', W') is also a TATR-witness state. (2) For the induction step, suppose that $(s, \delta, W) \lesssim (s, \delta', W')$, and a TATR-witness state is reachable from (s, δ, W) implies a TATR-witness state is reachable from (s, δ', W'). Given that $((s_0, \delta_0, W_0), e, (s, \delta, W))$ is a transition in ZG_{AL}, and (s_0, δ_0', W_0') satisfies $\delta_0 \subseteq \delta_0'$ and $W_0' \subseteq W_0$ (i.e., $(s_0, \delta_0, W_0) \lesssim (s_0, \delta_0', W_0')$). Then there is also a transition labelled with e from (s_0, δ_0', W_0'), since the first parts of (s_0, δ_0, W_0) and (s_0, δ_0', W_0') are the same. Thus we can get $((s_0, \delta_0', W_0'), e, (s, \delta', W'))$ is a transition in ZG_{AL}, where $\delta \subseteq \delta'$ and $W' \subseteq W$, because by the same transition and event from s_0, $W_0' \subseteq W_0$ implies $W' \subseteq W$, and $\delta_0 \subseteq \delta_0'$ implies $\delta \subseteq \delta'$. Therefore, the induction step holds.

When exploring ZG_{AL}, (s, δ, W) is found and there exists a configuration (s, δ', W') (which has been visited before) satisfying $(s, \delta, W) \lesssim (s, \delta', W')$, then we do not need to search from (s, δ, W). The above induction will end until (s, δ, W) and (s, δ', W') have been reached. As a summary, the lemma holds.

Theorem 2. Algorithm 1 returns true iff $\text{traces}_{\text{ZG}}(\mathcal{A}) \subseteq \text{traces}(\mathcal{L})$.

Proof. Because the zones are finite (with normalization), the number of product states is finite and the algorithm eventually terminates. Let ps be an abstract configuration in ZG_{AL}. Define $\text{Dist(ps)} \in N \cup \{\infty\}$ as the length of the shortest TATR-witness trace from ps (if a TATR-witness state is not reachable from ps, $\text{Dist(ps)} = \infty$). For a set of states S, if $S = \emptyset$, $\text{Dist}(S) = \infty$, otherwise, $\text{Dist}(S) = \min_{s \in S} \text{Dist}(s)$. We prove the correctness of the algorithm using the following invariants which are quite similar to [16].

- If there exists a TATR-witness state ps in working ∪ anti, then ps is reachable from a state in Init.
- If there is a reachable TATR-witness state, then Dist(anti) > Dist(working).

Algorithm 1 returns false only if the set W is empty on line 8. In this case, (s, δ, W) is a TATR-witness state, and hence there exists a TATR-witness state in working ∪ anti. By the first invariant, $\mathcal{A}$ cannot refine $\mathcal{L}$ in trace semantics. Algorithm 1 returns true only when working is empty, which implies that Dist(anti)>Dist(working) is not true. By the second invariant, there is no reachable TATR-witness state and $\mathcal{A}$ refines $\mathcal{L}$ in trace semantics.

Theorem 4. tmtraces($\mathcal{P}$) $\subseteq$ tmtraces($\mathcal{Q}$) iff there is no reachable state $(s_p, \emptyset, \delta)$ in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ where δ is not false.

Proof. (**if**) For a timed trace $\mathrm{tt} = \langle (d_1, e_1), (d_2, e_2), \ldots, (d_n, e_n) \rangle$, let $|\mathrm{tt}| = n$ be the length of tt. If tmtraces($\mathcal{P}$) $\subsetneq$ tmtraces($\mathcal{Q}$), there is a timed trace $\mathrm{tt}_\mathcal{P}$ in tmtraces($\mathcal{P}$) and a timed trace $\mathrm{tt}_\mathcal{Q}$ in tmtraces($\mathcal{Q}$), such that $\mathrm{tt}_\mathcal{Q}$ is a maximal prefix of $\mathrm{tt}_\mathcal{P}$ satisfying (1) $|\mathrm{tt}_\mathcal{Q}| = |\mathrm{tt}_\mathcal{P}| - 1$; (2) $\mathrm{tt}_\mathcal{P}$ is not in tmtraces($\mathcal{Q}$). Let $\mathrm{tt}_\mathcal{P} = \langle (d_1, e_1), (d_2, e_2), \ldots, (d_{n-1}, e_{n-1}), (d_n, e_n) \rangle$, and $\mathrm{tt}_\mathcal{Q} = \langle (d_1, e_1), (d_2, e_2), \ldots, (d_{n-1}, e_{n-1}) \rangle$. Let $((s_p, v_p), X)$ be the state in $\mathcal{P} \otimes \mathcal{Q}$ with which a concrete run of $\mathcal{P} \otimes \mathcal{Q}$ exhibiting $\mathrm{tt}_\mathcal{Q}$ ends. Notice that X must not be empty. Next, (s_p, v_p) can perform a timed event (d_n, e_n), but no x in X can perform the same timed event. Because zone abstraction preserves reachability, the concrete run is abstracted by an abstract run in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$, and there exists an abstract state (s_p, X', δ) in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ such that (s_p, X', v) is a concrete configuration of (s_p, X', δ) and it is the same as $((s_p, v_p), X)$. With (s_p, X', v), since (d_n, e_n) can be performed by $\mathcal{P}$, $v + d_n$ satisfies the guard g of a transition labeled with e_n from s_p. Since $\mathcal{Q}$ cannot perform (d, e_n), $v + d_n$ must satisfy negCons. Therefore $\delta \wedge g \wedge$ negCons cannot be false, and hence there is a successor (s'_p, X'', δ') such that δ' is not false. Because negCons is the conjunction of negations of all e_n-transitions from any state in X', X'' is $\emptyset$. Thus, $(s'_p, \emptyset, \delta')$ where δ' is not false is reachable. As a result, if there is no reachable state $(s_p, \emptyset, \delta)$ in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ where δ is not false, tmtraces($\mathcal{P}$) $\subseteq$ tmtraces($\mathcal{Q}$) holds.

(**only-if**) It is proved straightforwardly. If tmtraces($\mathcal{P}$) $\subseteq$ tmtraces($\mathcal{Q}$), then any timed trace performed by $\mathcal{P}$ can also be performed by $\mathcal{Q}$. Thus $(s_p, \emptyset, \delta)$ is not reachable.

Lemma 3. Let (s_p, X_q, δ) and (s_p, X'_q, δ') be two abstract configurations in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$. If $(s_p, X'_q, \delta') \preccurlyeq (s_p, X_q, \delta)$, then a TTR-witness state is reachable from (s_p, X'_q, δ') implies a TTR-witness state is reachable from (s_p, X_q, δ).

Proof. By induction. (1) First, the base case is that if (s_p, X'_q, δ') is a TTR-witness state (that is, $X'_q = \emptyset$), because $(s_p, X'_q, \delta') \preccurlyeq (s_p, X_q, \delta)$ (and $X_q \subseteq_m X'_q$), $X_q = \emptyset$ and (s_p, X_q, δ) is also a TTR-witness state. (2) For the induction step, we show that if $(s, X_0, \delta_0) \preccurlyeq (s, X_1, \delta_1)$, then for all e, such that $((s, X_0, \delta_0), e, (s', X'_0, \delta'_0))$ is a transition in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$, there exists (s', X'_1, δ'_1) such that $((s, X_1, \delta_1), e, (s', X'_1, \delta'_1))$ is a transition in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ and $(s', X'_0, \delta'_0) \preccurlyeq (s', X'_1, \delta'_1)$. For simplicity of the proof, a state $(s, C_{q\oplus})$ in X_q is written as (s, A).

Let $(s, X_0, v_0) \in (s, X_0, \delta_0)$ be a concrete configuration. Since $(s, X_0, \delta_0) \preccurlyeq (s, X_1, \delta_1)$ by definition, there is a concrete configuration $(s, X_1, v_1) \in (s, X_1, \delta_1)$ such that $v_0[\mathrm{maps}(m)] = m(v_1)$ for some m. Let $((s, v_0[C_p]), (d, e), (s', v'_0))$ be a concrete transition in $\mathcal{P}$. For each state (s_1, A_1) in X_1 such that $(((s_1, A_1), v_1[A_1]), (d, e), ((s'_1, A'_1), v'_1))$ is a concrete transition of $\mathcal{Q}$, there is a corresponding state $(s'_0, A'_0) \in X_0$ such that $(((s_0, A_0), v_0[A_0]), (d, e), ((s'_0, A'_0), v'_0))$ is a transition of $\mathcal{Q}$ where $s_0 = s_1$, $s'_0 = s'_1$, and v'_0 and v'_1 (A'_0 and A'_1, respectively) are equivalent up to clock renaming. Since (s, X_0, v_0) is arbitrary, $X'_1 \subseteq_m X'_0$ holds for some mapping m. Since $\delta_0 \subseteq_m \delta_1$ for some m, $\delta'_0 \subseteq_m \delta'_1$ holds for some m and thus δ'_1 is not empty if δ'_0 is not. Therefore, we conclude that if $((s, X_0, \delta_0), e, (s', X'_0, \delta'_0))$ is a transition in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$, then $((s_1, X_1, \delta_1), e, (s'_1, X'_1, \delta'_1))$ is a transition in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$ and $(s'_0, X'_0, \delta'_0) \preccurlyeq (s'_1, X'_1, \delta'_1)$. Thus, the induction step holds.

When exploring $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$, (s_p, X'_q, δ') is found and there exists a configuration (s_p, X_q, δ) (which has been visited before) satisfying $(s_p, X'_q, \delta') \preccurlyeq (s_p, X_q, \delta)$, then we don not need to search from (s_p, X'_q, δ'). The above induction will end until (s_p, X'_q, δ') and (s_p, X_q, δ) have been reached. As a summary, the lemma holds.

Theorem 5. Algorithm 2 returns true iff tmtraces($\mathcal{P}$) $\subseteq$ tmtraces($\mathcal{Q}$).

Proof. Let ps be an abstract configuration in $\mathcal{Z}(\mathcal{P} \otimes \mathcal{Q})$. We prove the correctness of the algorithm using the following invariants which are quite similar to Theorem 2.

- If there exists a TTR-witness state ps in working $\cup$ anti, then ps is reachable from a state in Init.
- If there is a reachable TTR-witness state, then Dist(anti) $>$ Dist(working).

Lemma 4. Given an MDP $\mathcal{M}$, an LTS $\mathcal{L}$. Let $\mathcal{D} = \mathcal{M} \times \det(\mathcal{L})$, and the set G_t containing all the PR-witness states of $\mathcal{D}$. For any product states (s_1, N_1) and (s_2, N_2) of $\mathcal{D}$ s.t. $(s_2, N_2) \preceq (s_1, N_1)$, $\Pr_{\max}(\mathcal{D}, (s_1, N_1), G_t) \geqslant \Pr_{\max}(\mathcal{D}, (s_2, N_2), G_t)$ and $\Pr_{\min}(\mathcal{D}, (s_1, N_1), G_t) \geqslant \Pr_{\min}(\mathcal{D}, (s_2, N_2), G_t)$.

Proof. By induction. (1) First, the base case is that (s_2, N_2) is in G_t. Since for any $n_1 \in N_1$, there is $n_2 \in N_2$ where $n_1 \prec n_2$, then (s_1, N_1) should be in G_t. So the lemma holds since the maximal/minimal reachability probability of a state in G_t is always 1.

(2) For the induction step, suppose that (s'_1, N'_1) and (s'_2, N'_2) can satisfy the lemma above, i.e., for any (s'_1, N'_1) such that $(s'_2, N'_2) \preceq (s'_1, N'_1)$, the lemma holds. Given a state (s_2, N_2) and any state (s_1, N_1) such that $(s_2, N_2) \preceq (s_1, N_1)$, if there is a distribution μ_2 from (s_2, N_2), there must exist a distribution μ_1 from (s_1, N_1) by Definition 10 such that: for any $\mu_2((s'_2, N'_2)) > 0$, there exists (s'_1, N'_1) such that $\mu_2((s'_2, N'_2)) = \mu_1((s'_1, N'_1))$, and $(s'_2, N'_2) \preceq (s'_1, N'_1)$. By induction hypothesis, we have $\Pr_{\max}(\mathcal{D}, (s'_1, N'_1), G_t) \geqslant \Pr_{\max}(\mathcal{D}, (s'_2, N'_2), G_t)$ and $\Pr_{\min}(\mathcal{D}, (s'_1, N'_1), G_t) \geqslant \Pr_{\min}(\mathcal{D}, (s'_2, N'_2), G_t)$. Thus $\Pr_{\max}(\mathcal{D}, (s_1, N_1), G_t) \geqslant \Pr_{\max}(\mathcal{D}, (s_2, N_2), G_t)$ and $\Pr_{\min}(\mathcal{D}, (s_1, N_1), G_t) \geqslant \Pr_{\min}(\mathcal{D}, (s_2, N_2), G_t)$ hold. Therefore the lemma holds.

When exploring $\mathcal{D}$, (s_2, N_2) is found and there exists a configuration (s_1, N_1) (which has been visited before) satisfying $(s_2, N_2) \preceq (s_1, N_1)$, then (s_2, N_2) is added into $(s_1, N_1).ac$. The above induction will end until (s_1, N_1) and (s_2, N_2) have been reached. As a summary, the lemma holds.